Algorithmen und Datenstrukturen
Tutorium V

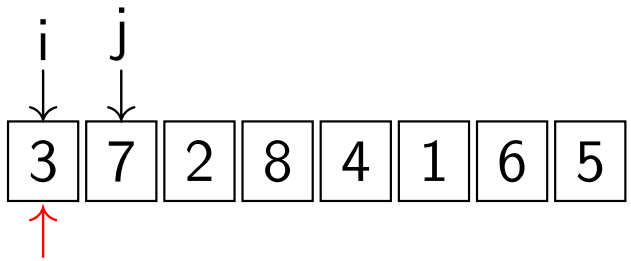
Michael R. Jung

18. - 23. 05. 2016

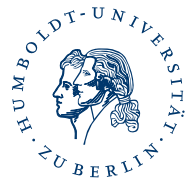
An folgender Beispielinstanz werden einige Sortierverfahren veranschaulicht:

3	7	2	8	4	1	6	5

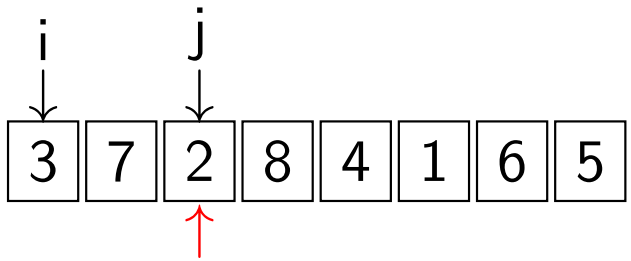
```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



↑...minpos

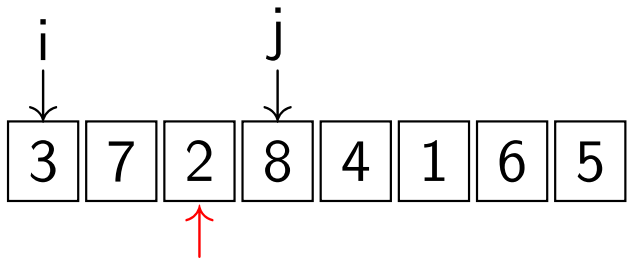


```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



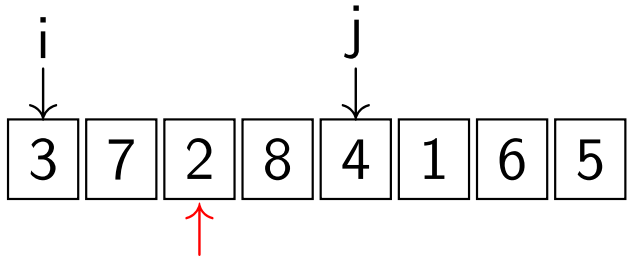
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



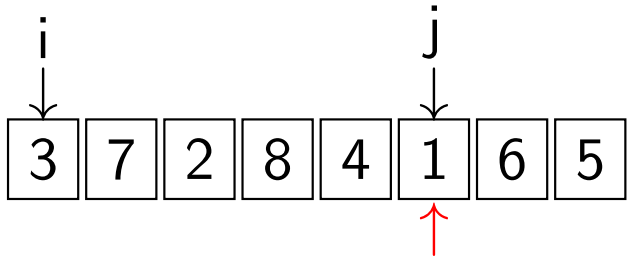
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



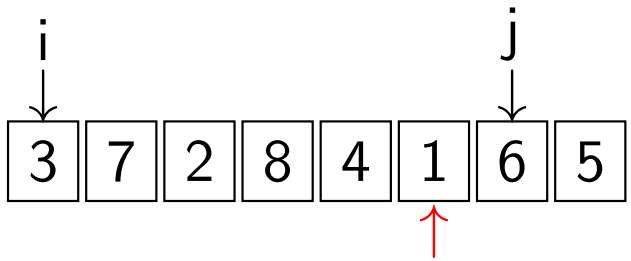
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



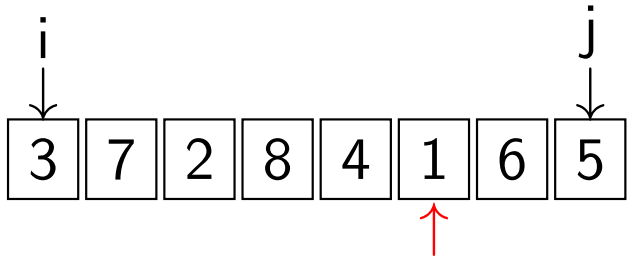
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



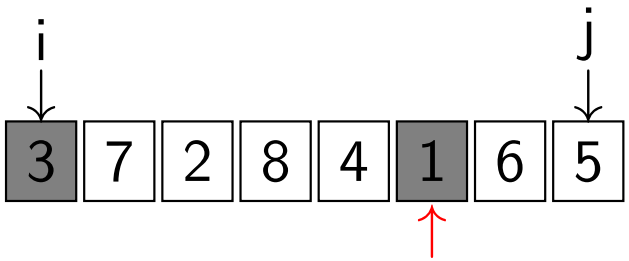
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



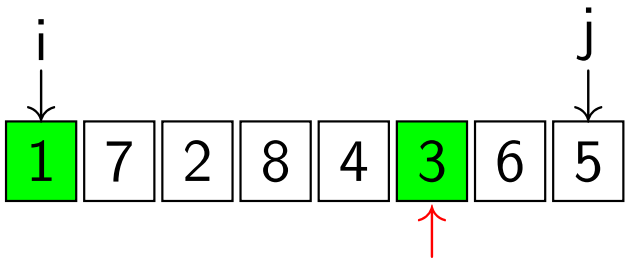
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



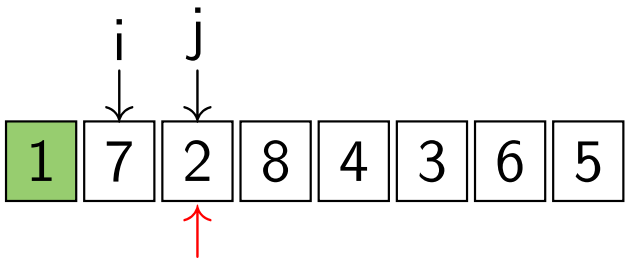
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



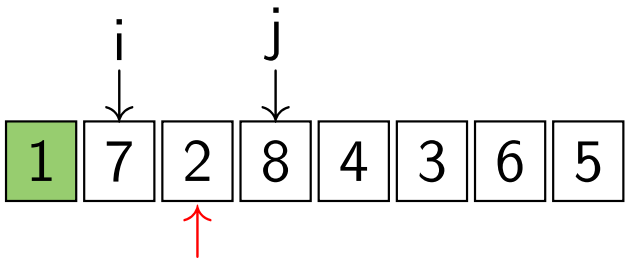
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



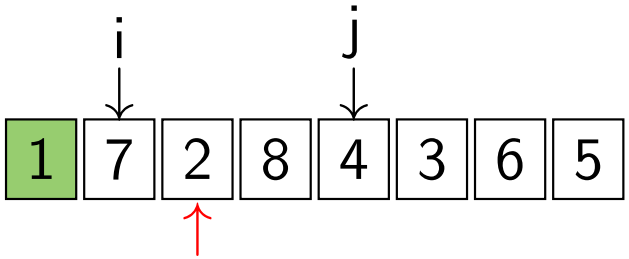
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



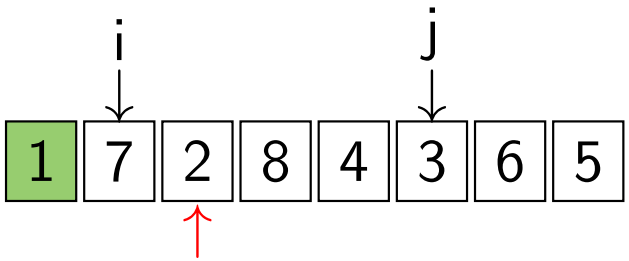
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



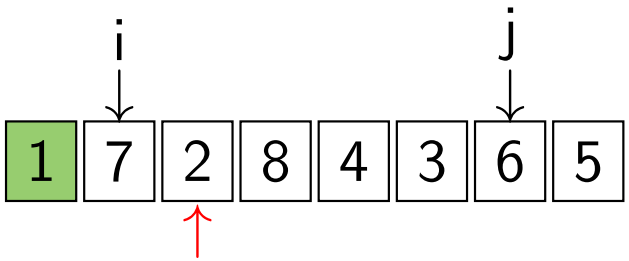
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



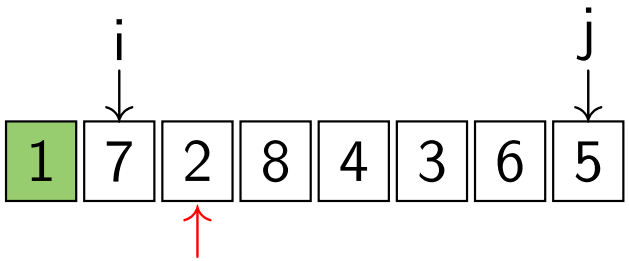
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



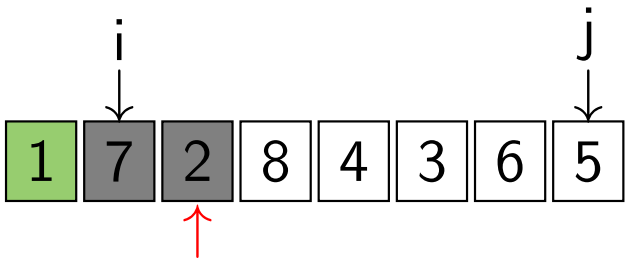
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



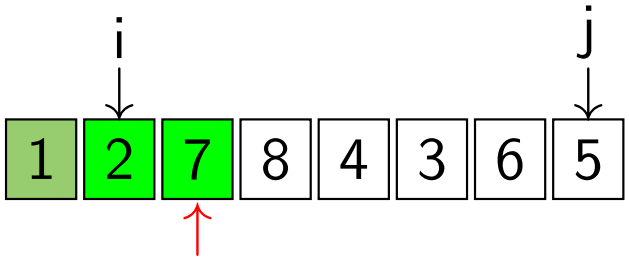
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



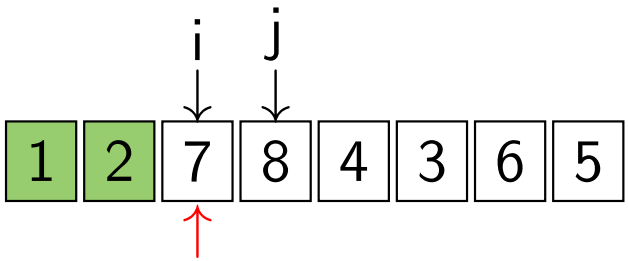
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



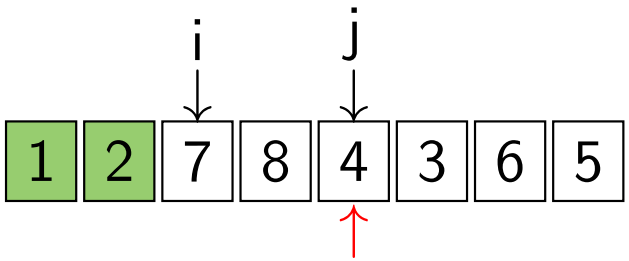
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



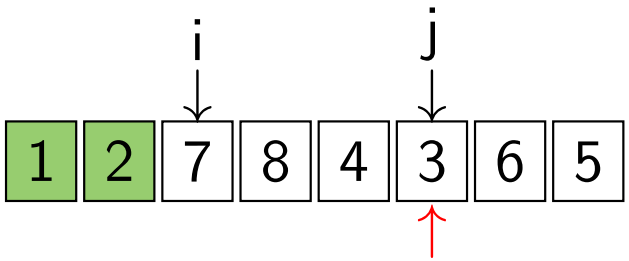
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



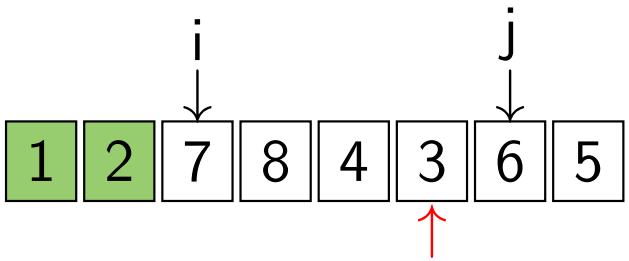
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



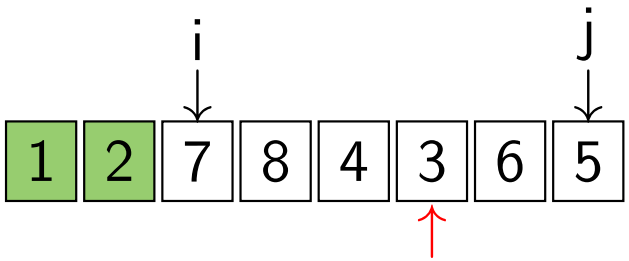
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



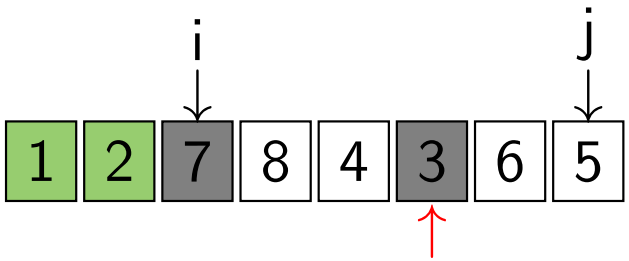
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



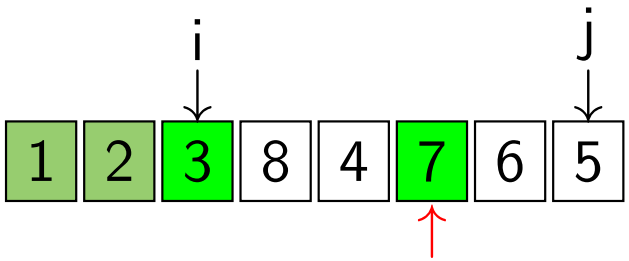
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



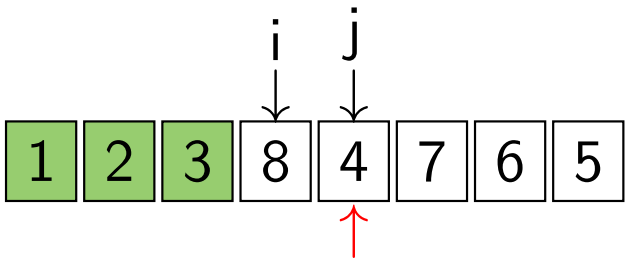
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



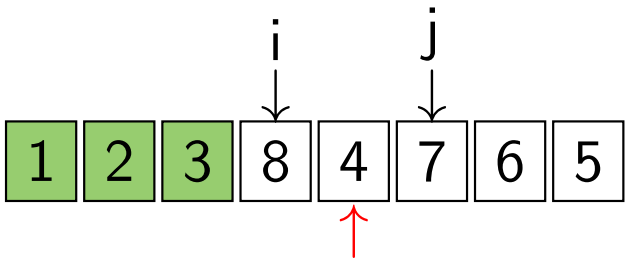
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



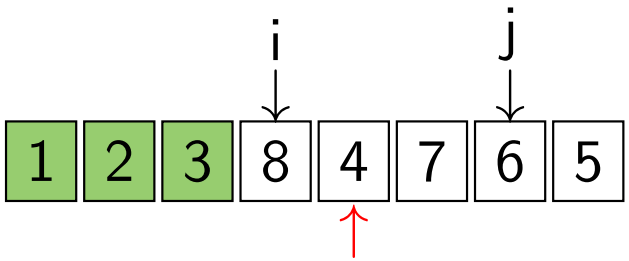
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



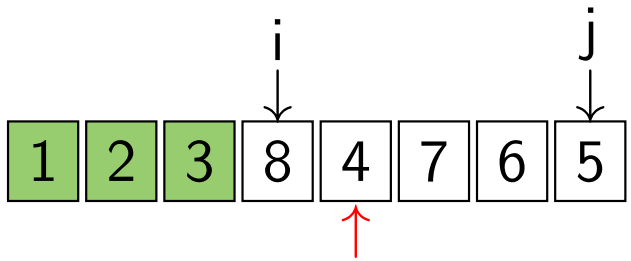
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



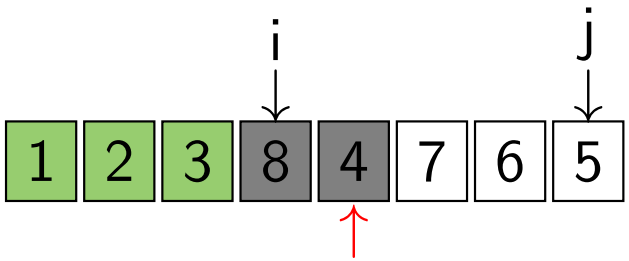
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



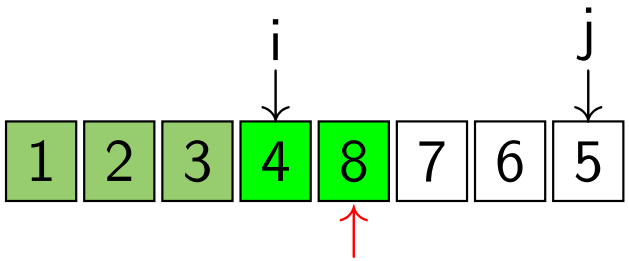
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



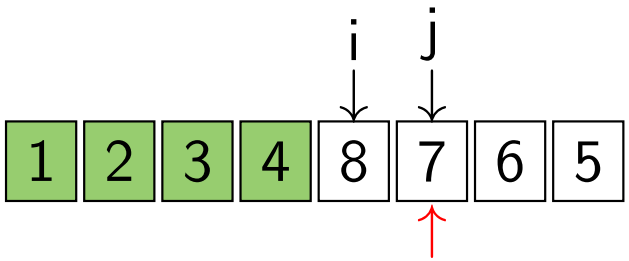
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



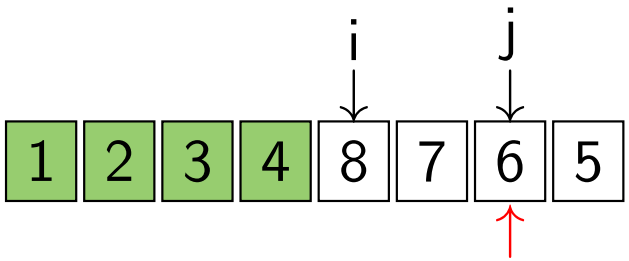
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



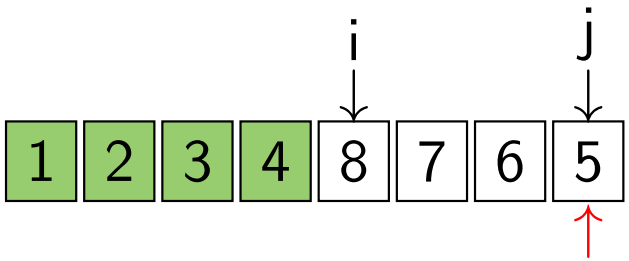
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



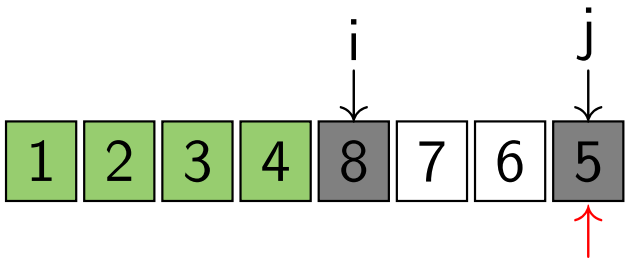
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



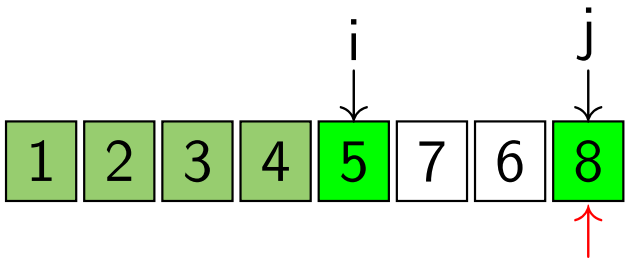
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



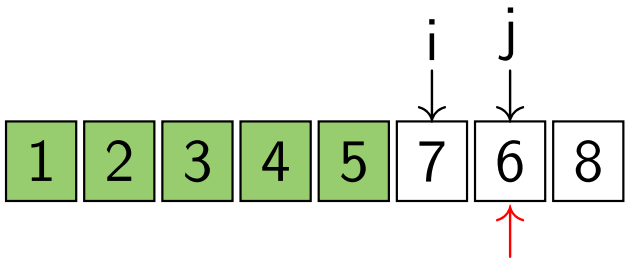
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



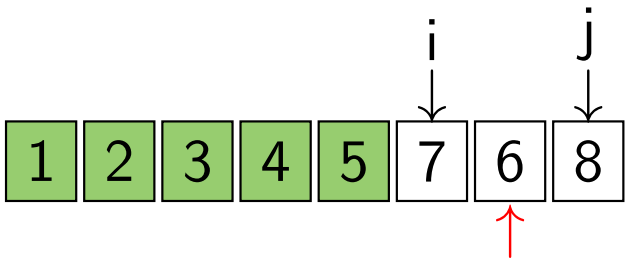
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



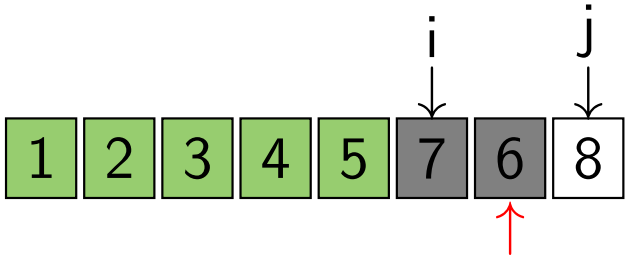
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



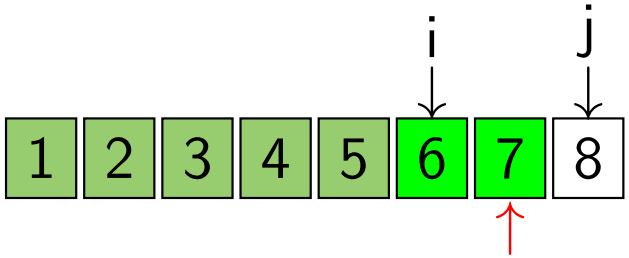
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



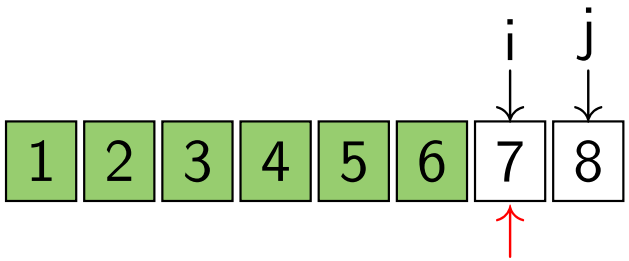
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



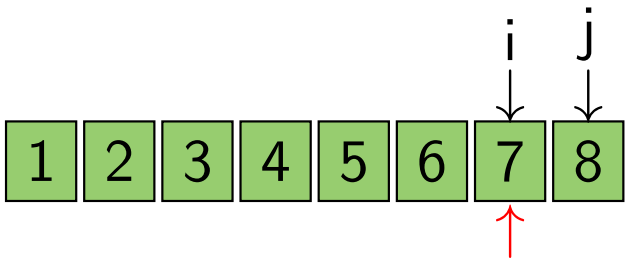
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```



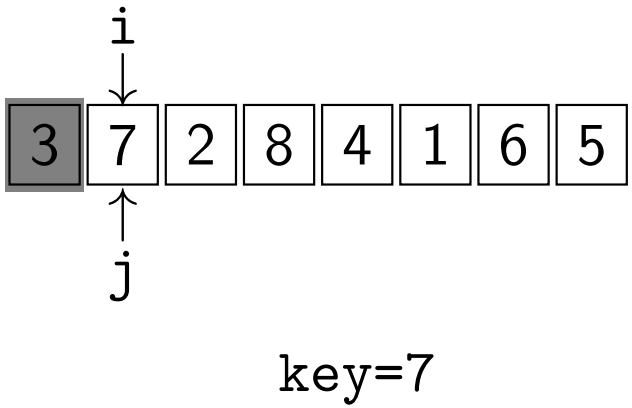
↑...minpos

```
S: array_of_names;
n := |S|
for i = 1..n-1 do
  min_pos := i;
  for j = i+1..n do
    if S[min_pos]>S[j] then
      min_pos := j;
    end if;
  end for;
  tmp := S[i];
  S[i] := S[min_pos];
  S[min_pos] := tmp;
end for;
```

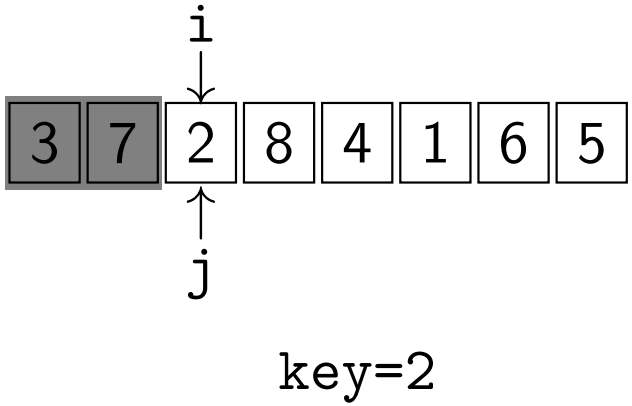


↑...minpos

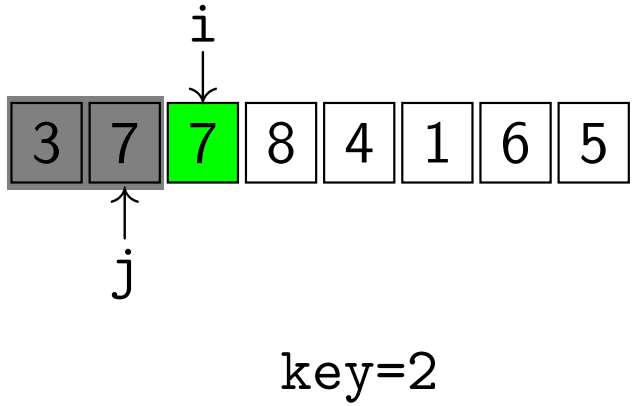
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

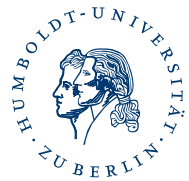


```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

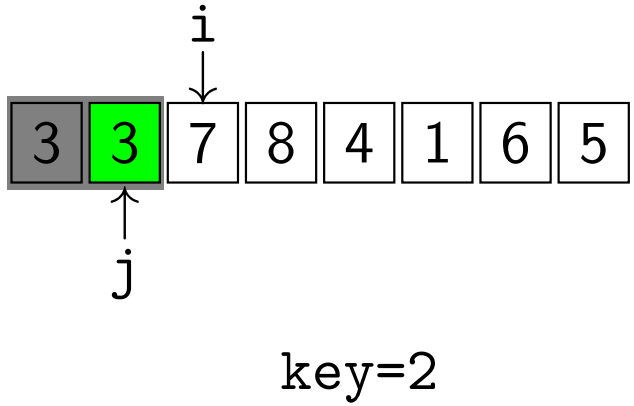



```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

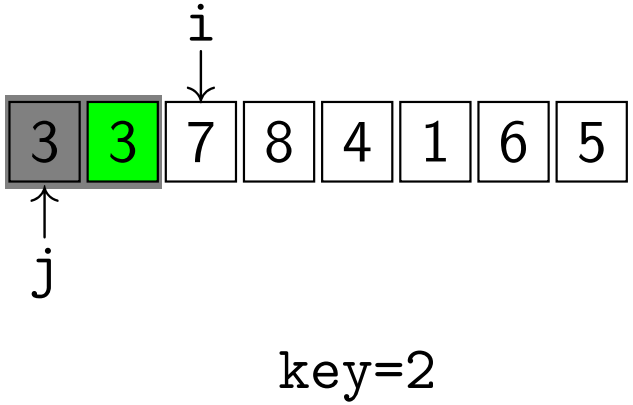




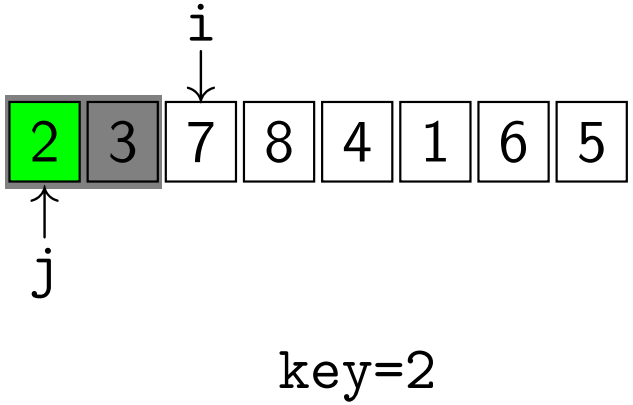

```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



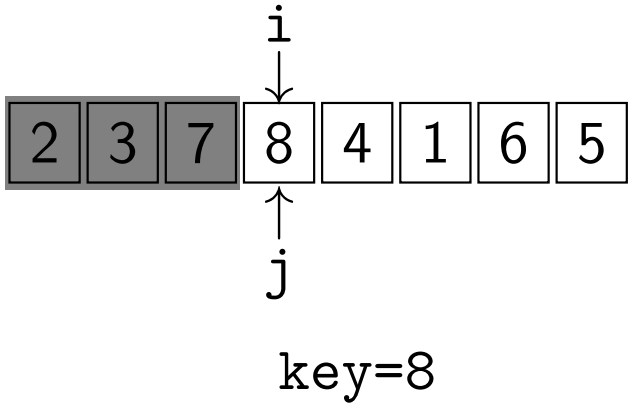
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



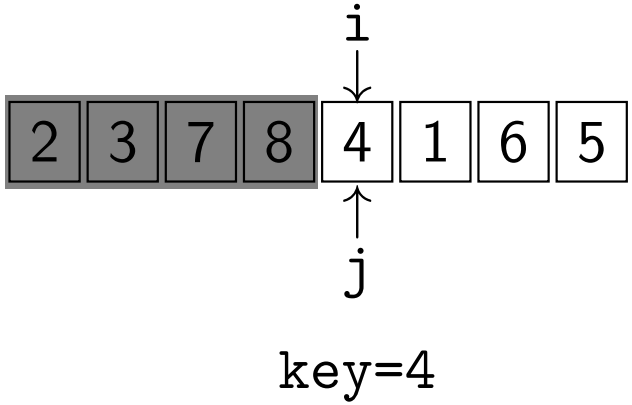
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



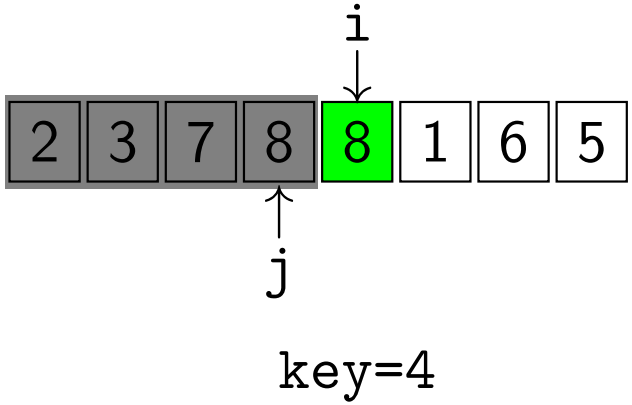
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



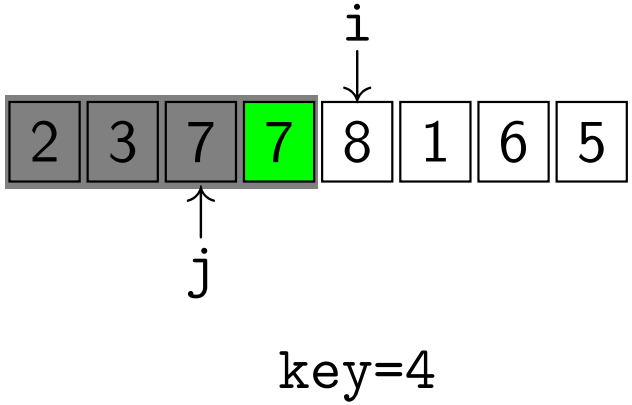
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



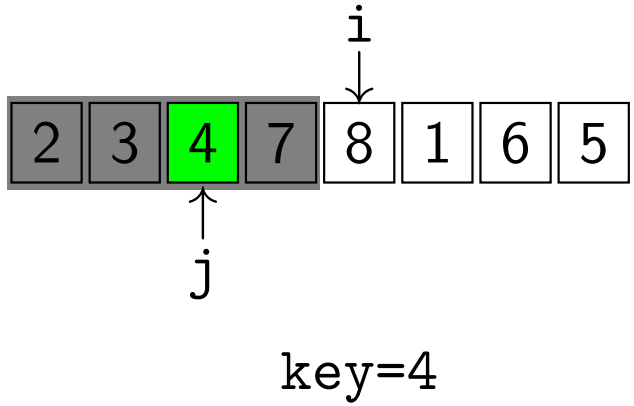
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

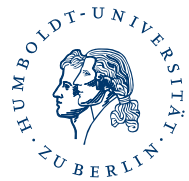


```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

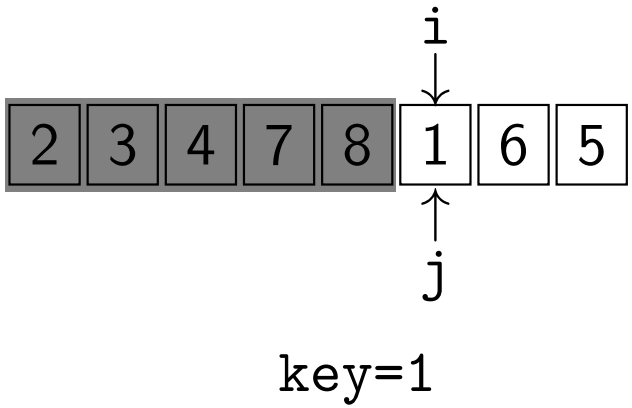



```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

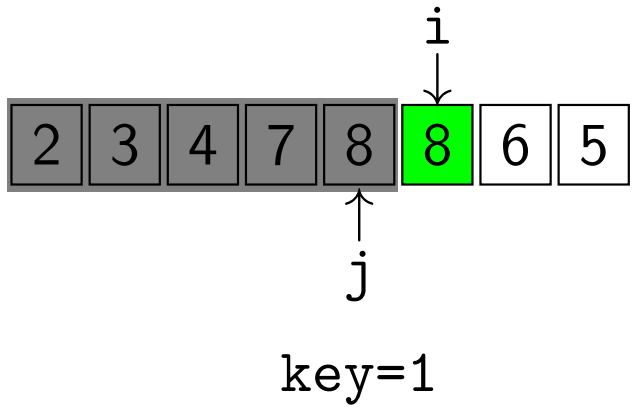




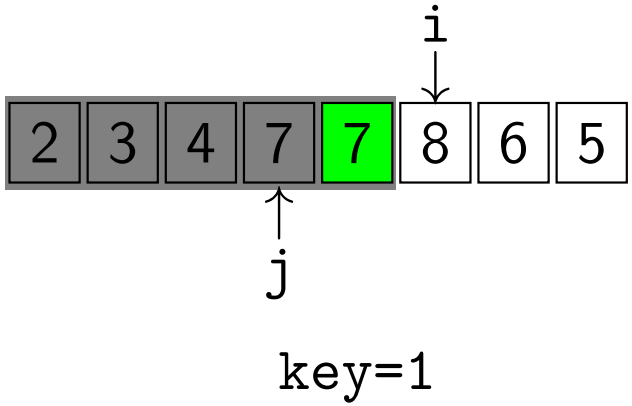
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



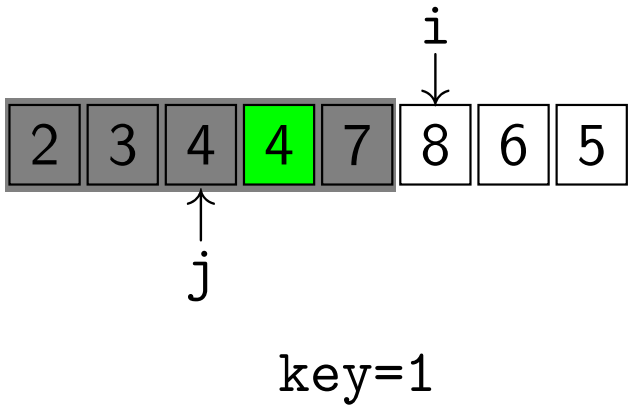

```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



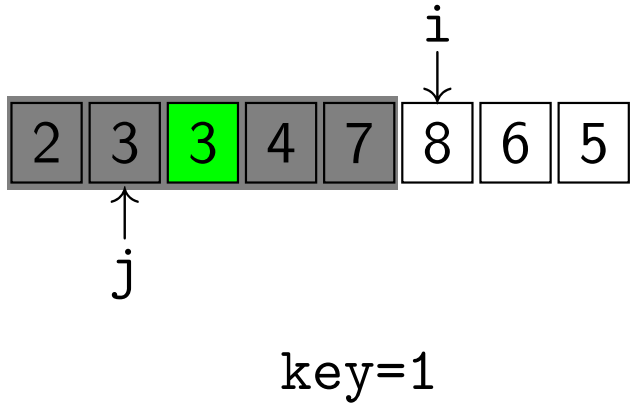

```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

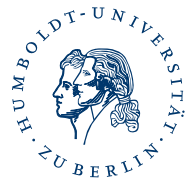


```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

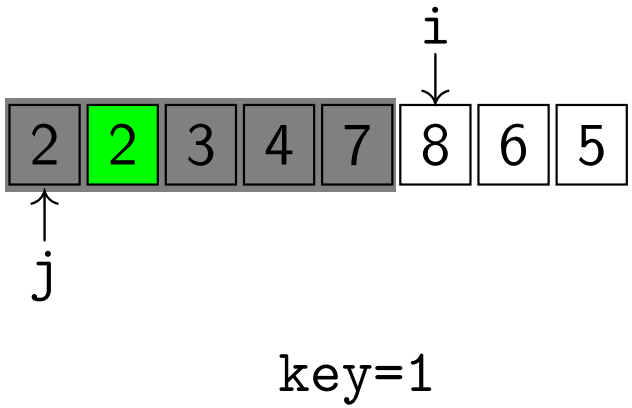



```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

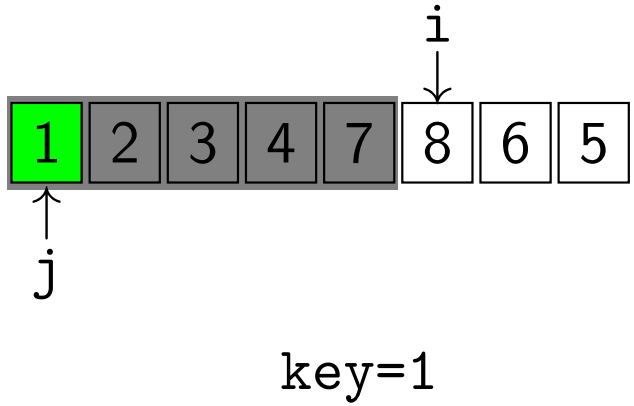


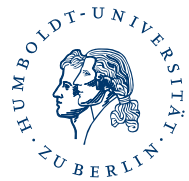


```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

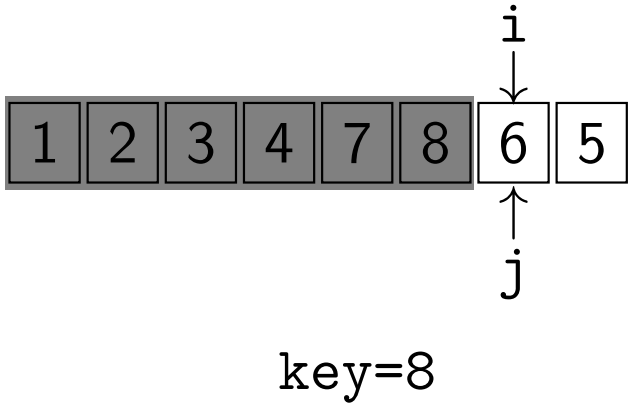



```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

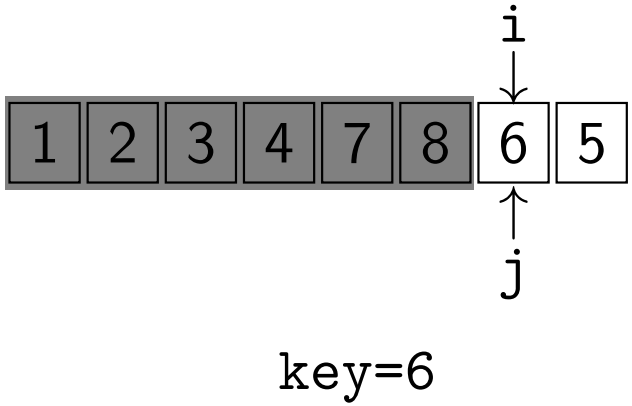




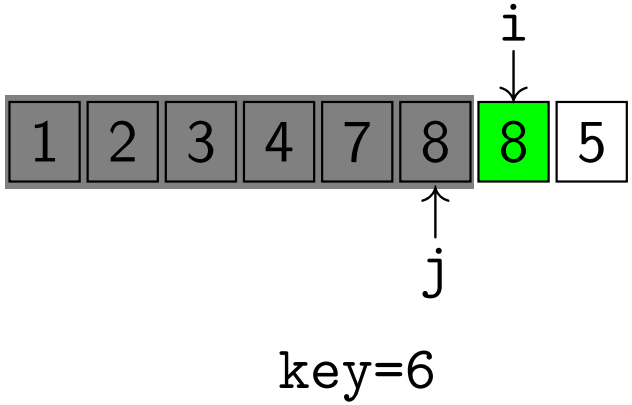
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



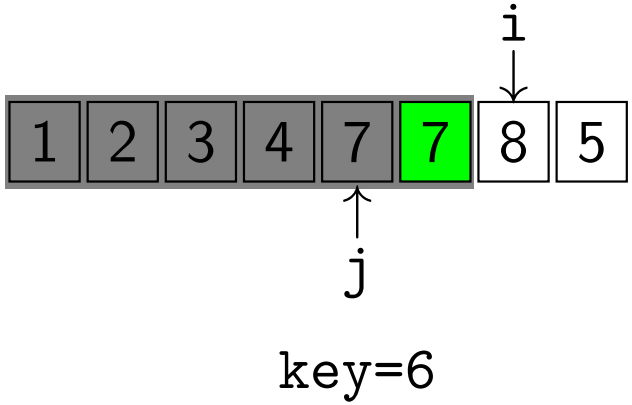
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



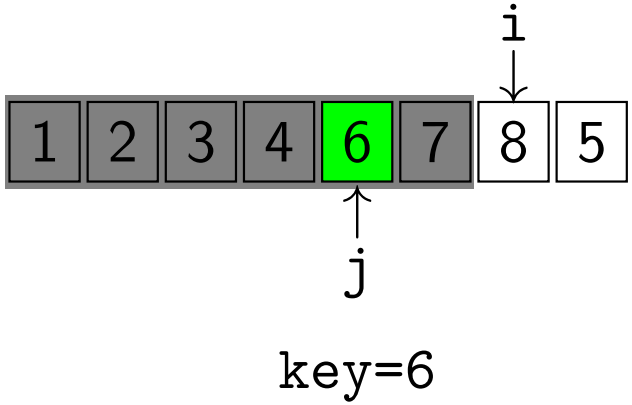
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

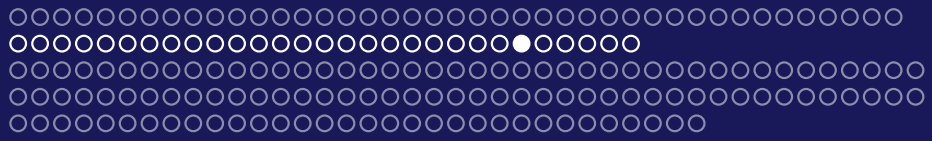


```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

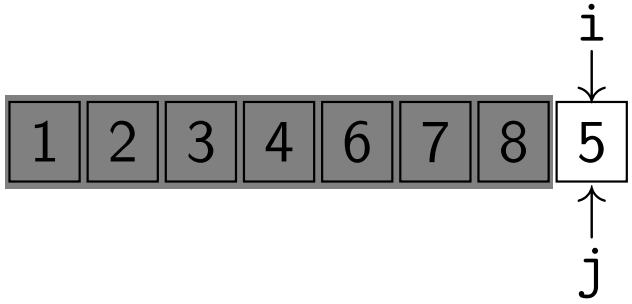


```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```

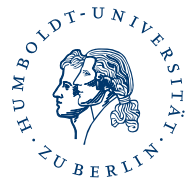




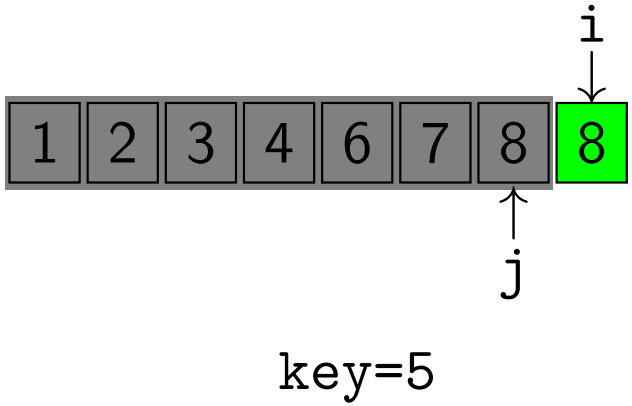

```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



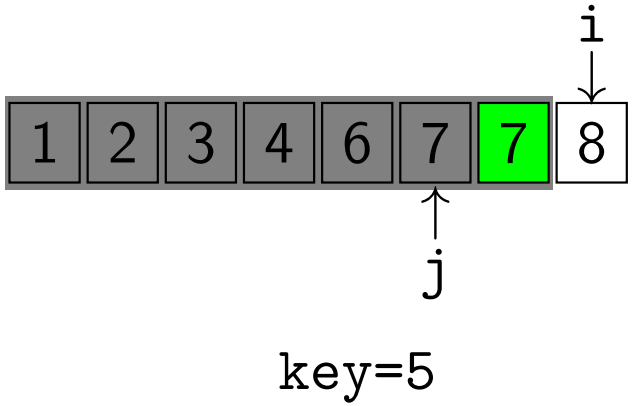
key=5



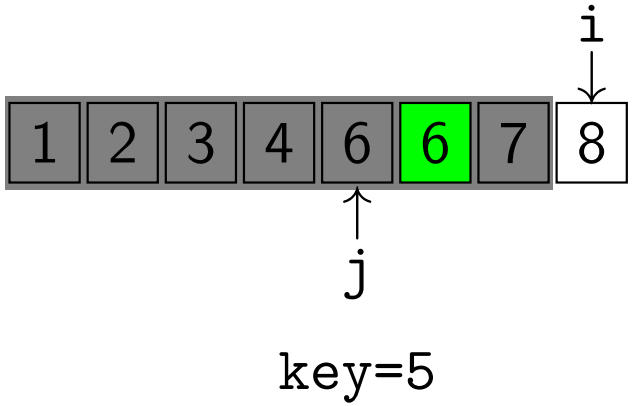
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



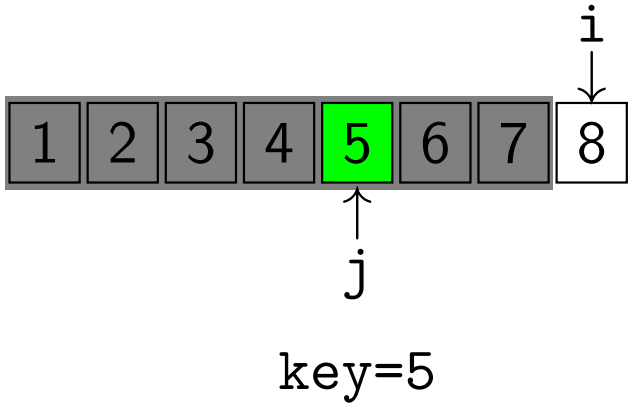
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



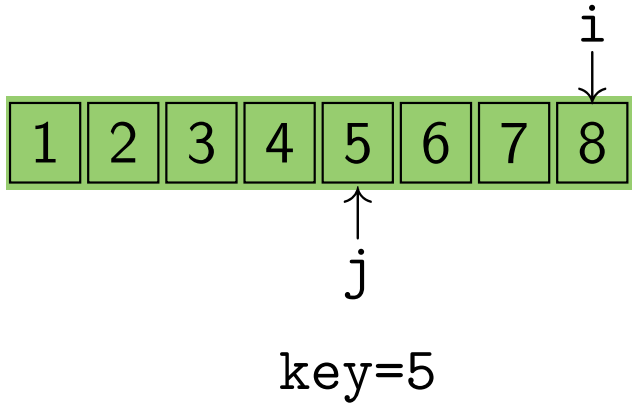
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



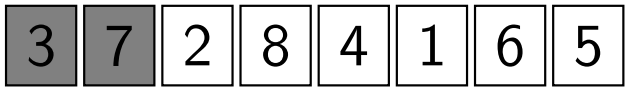

```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



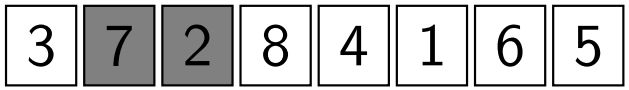
```
S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;
```



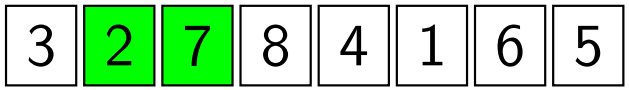
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



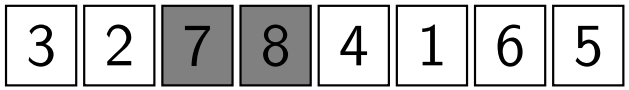
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

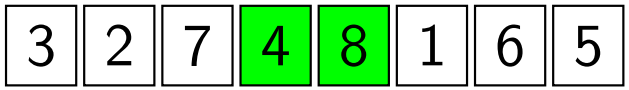


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

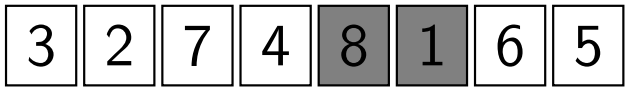


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

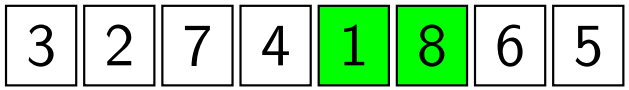
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



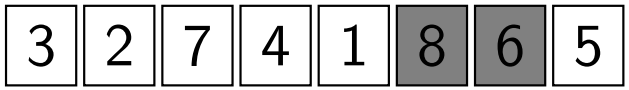
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



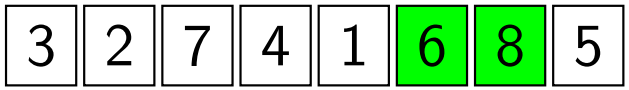
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



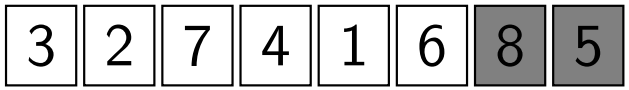
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



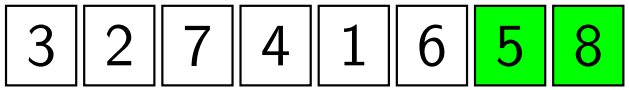
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



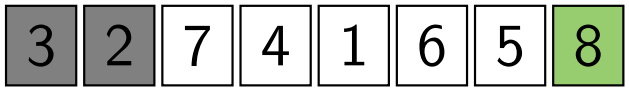
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



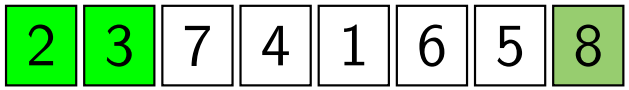
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



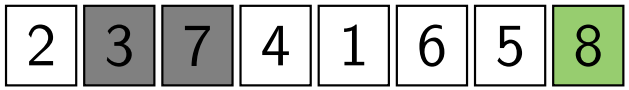
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



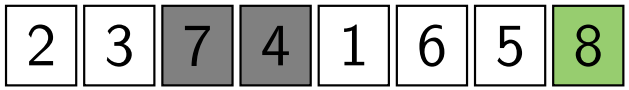
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



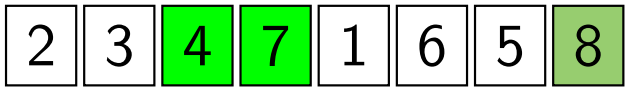
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

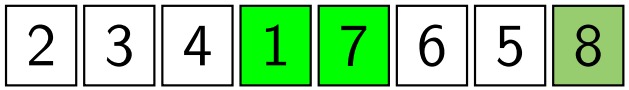


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

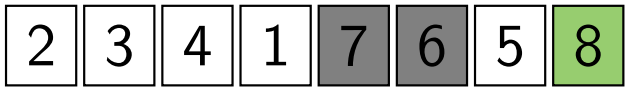


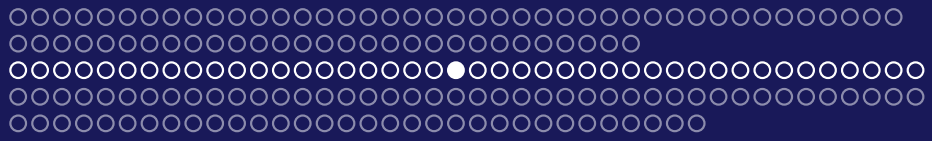
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

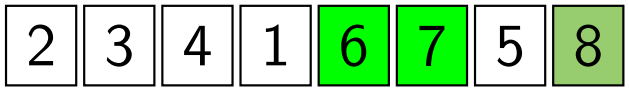


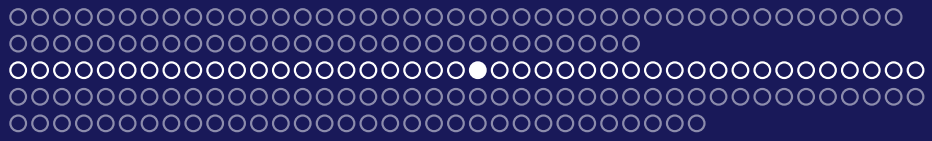
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



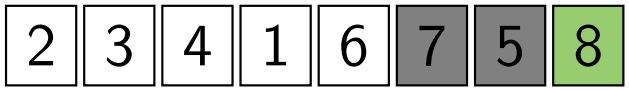


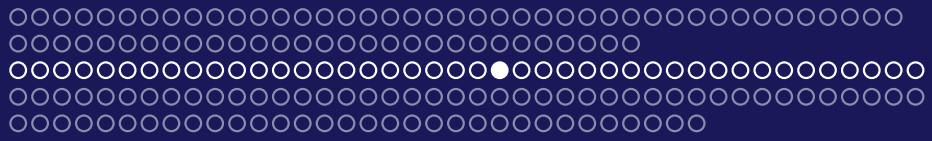
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



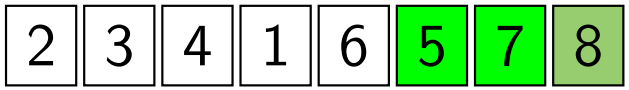


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

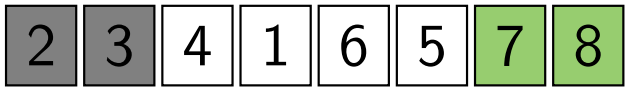




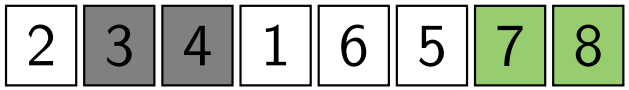
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



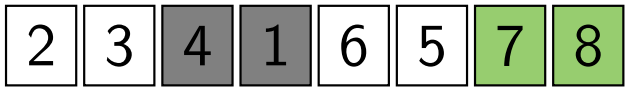
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



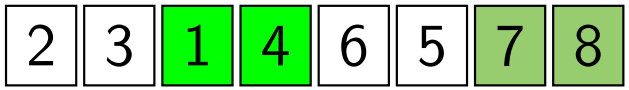
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

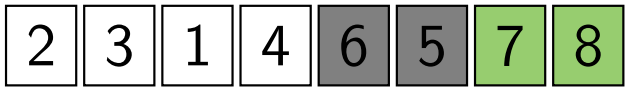


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



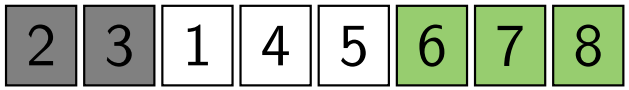
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

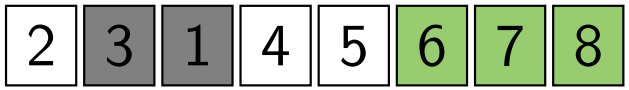


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

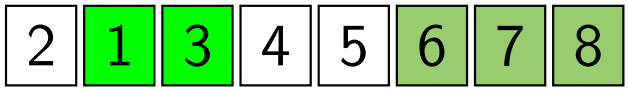
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



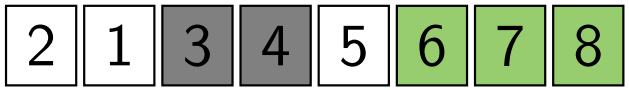
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



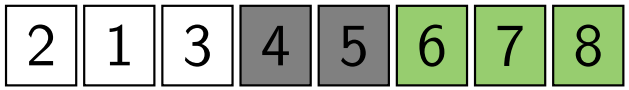
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



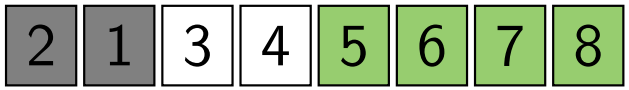
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



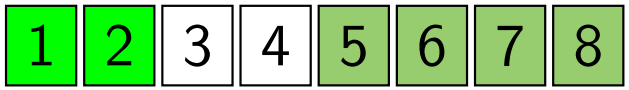
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



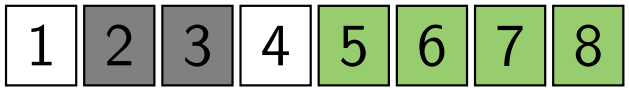
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

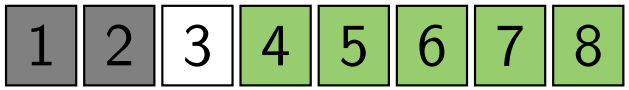


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

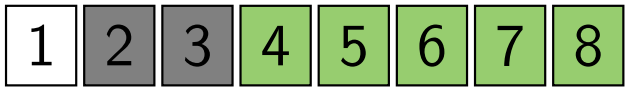


- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

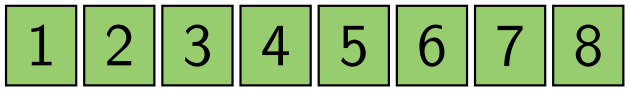
- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



- Go through array again and again
- Compare all direct neighbors
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

3	7	2	8	4	1	6	5

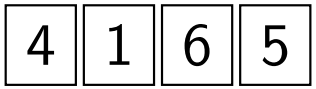
l=0

r=7

m=3


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



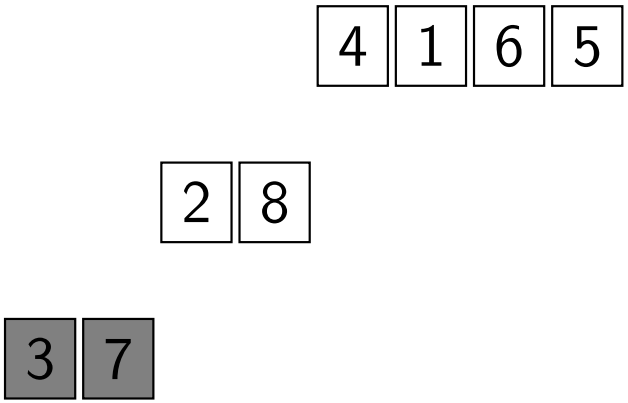
l=0

r=3

m=1


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



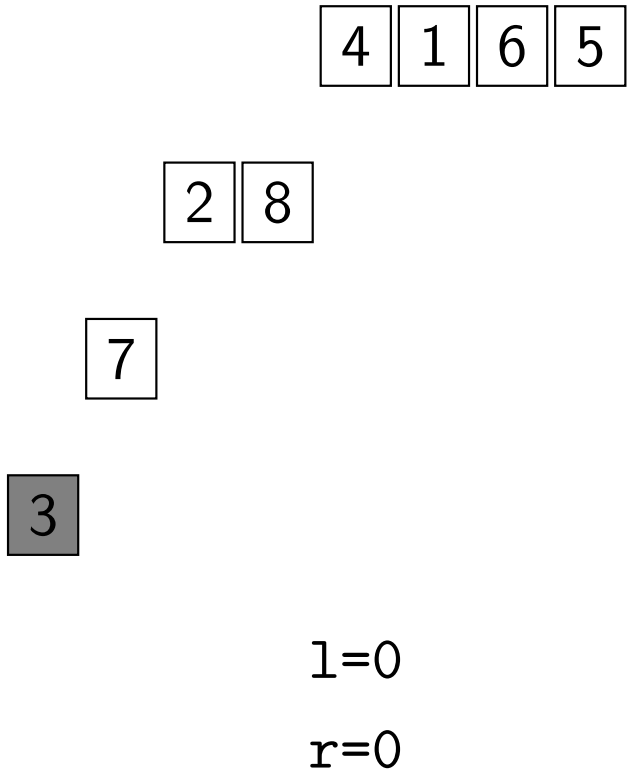
l=0

r=1

m=0


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

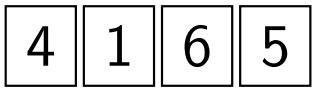
```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

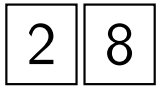


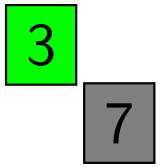
MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```







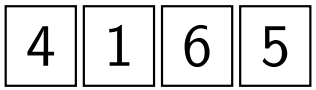
l=1

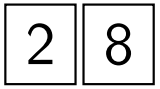
r=1

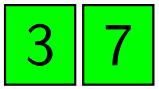
MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





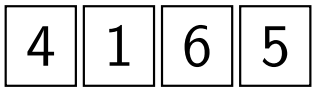


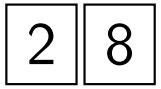
l=0

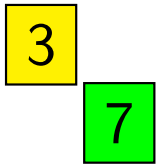
r=1


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```







l=0

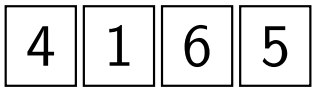
r=1

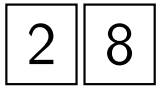
m=0

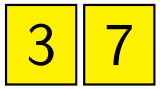
MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```







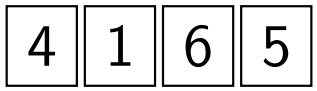
l=0

r=1

m=0


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=2

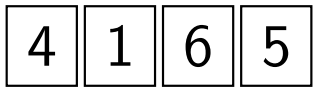
r=3

m=0

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

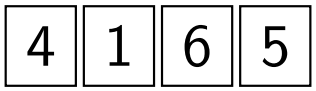


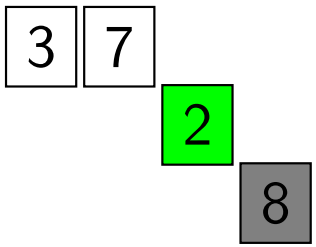
l=2

r=2


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



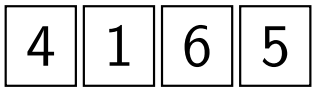


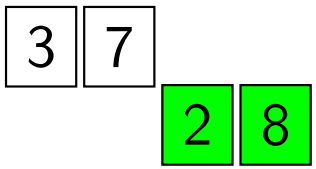
l=3

r=3


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



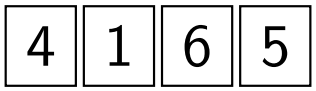


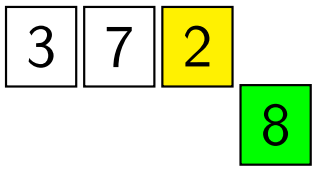
l=2

r=3


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





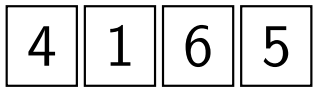
l=2

r=3

m=0


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

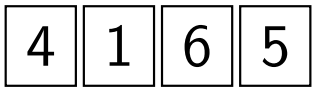
r=3

m=1

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



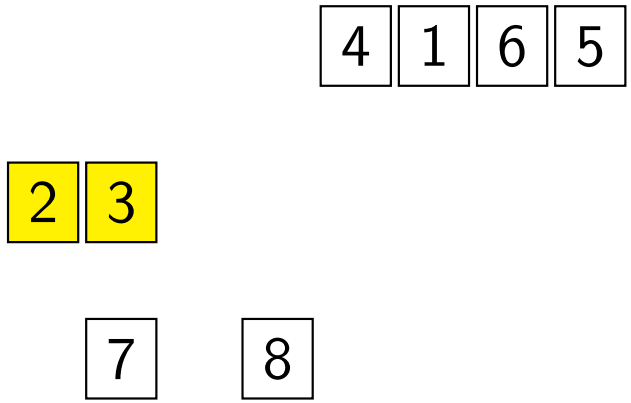
l=0

r=3

m=1


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



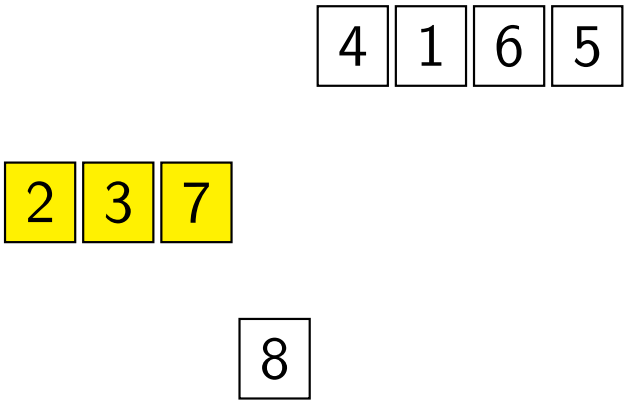
l=0

r=3

m=1

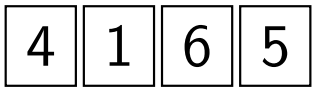

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```




```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

r=3

m=1


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

l=4

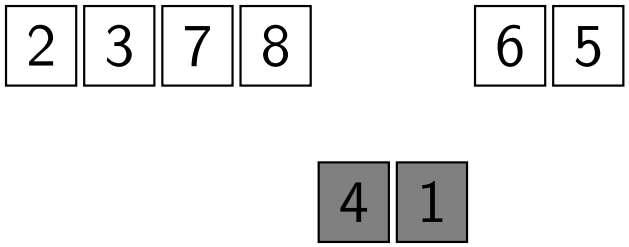
r=7

m=1

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

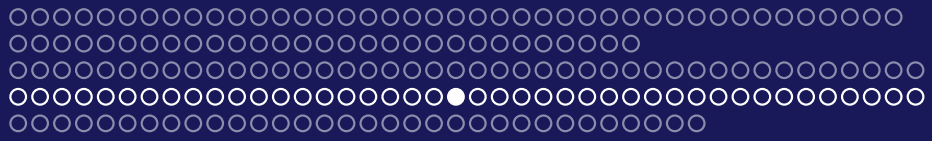
```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=4

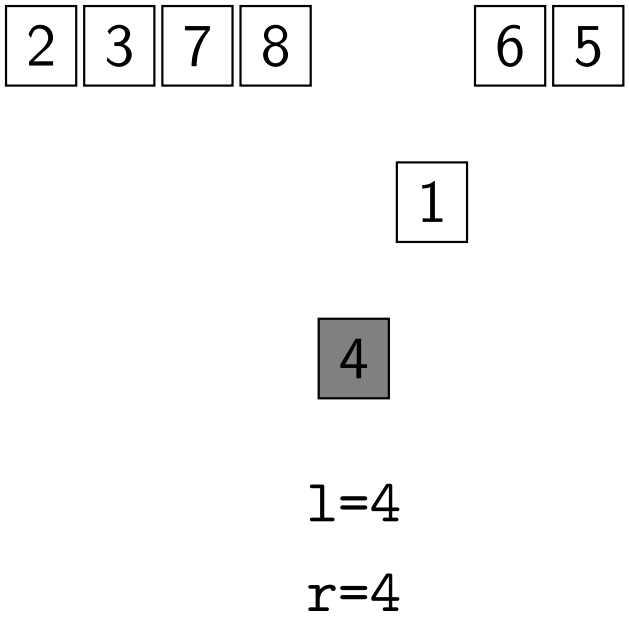
r=5

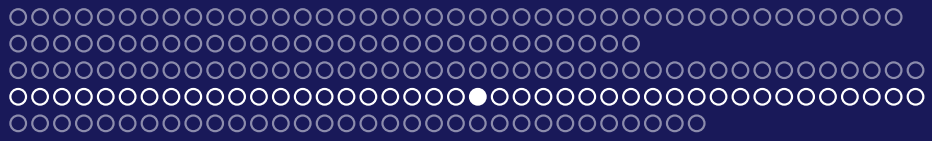
m=0




```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

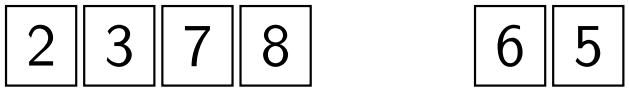
```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

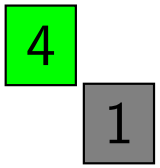





```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

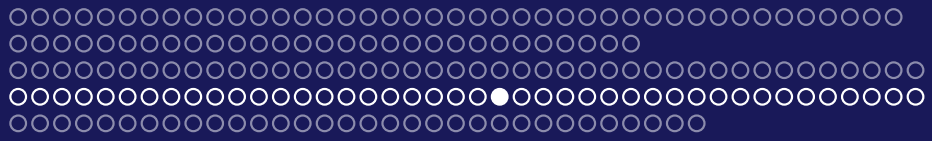
```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





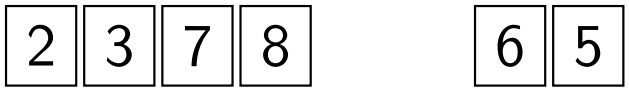
l=5

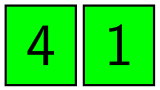
r=5




```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





l=4

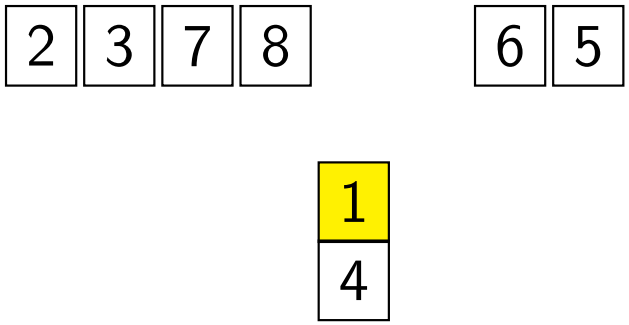
r=5

m=0

MergeSort

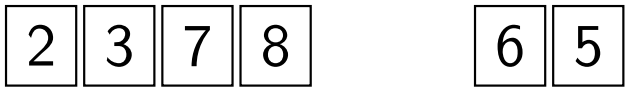
```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```




```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



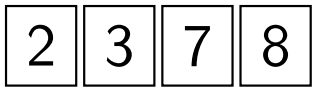
l=4

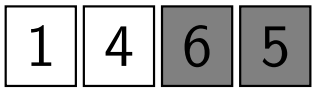
r=5

m=0


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





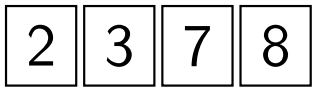
l=6

r=7

m=0


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

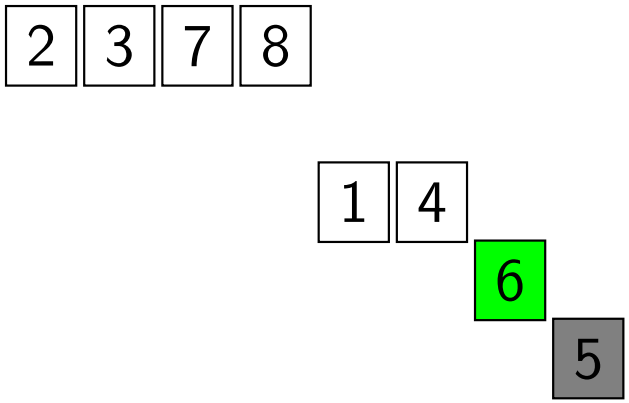


l=6

r=6


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

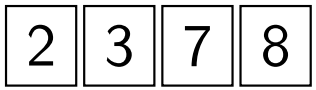


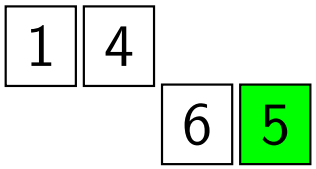
l=7

r=7


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





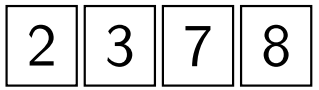
l=7

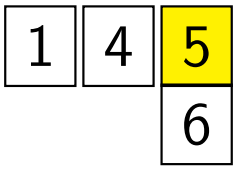
r=7

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





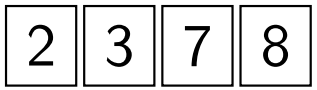
l=6

r=7

m=0


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=6

r=7

m=0


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

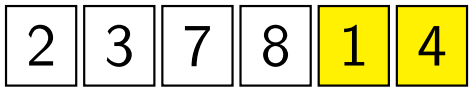
l=4

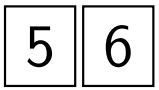
r=7

m=1


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```





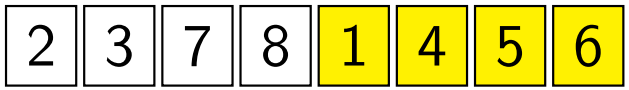
l=4

r=7

m=1


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



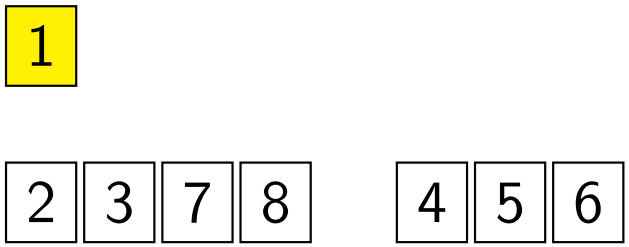
l=4

r=7

m=1


```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

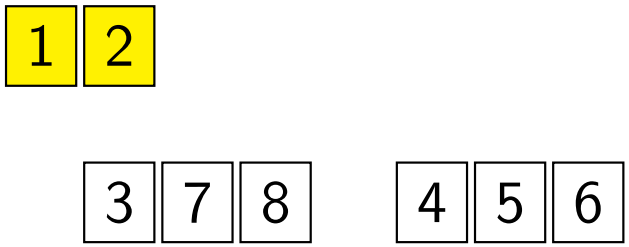
r=7

m=3

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

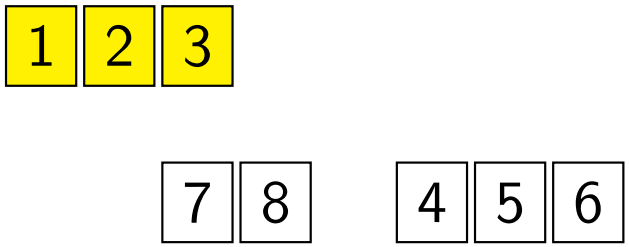
r=7

m=3

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

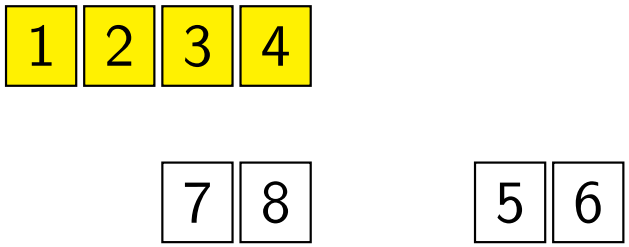
r=7

m=3

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

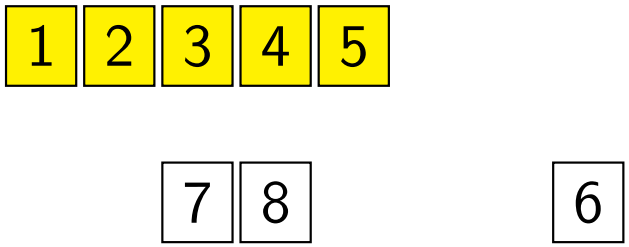
r=7

m=3

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

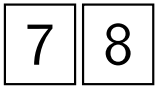
r=7

m=3

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



l=0

r=7

m=3

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```

l=0

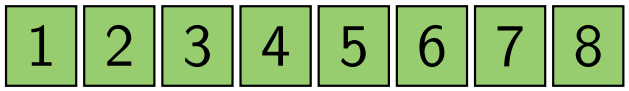
r=7

m=3

MergeSort

```
function void mergesort(S array;
                        l,r integer) {
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
```

```
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;         # Start of 1st list
  j := m+1;       # Start of 2nd list
  k := 1;         # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;       # Next target
  end while;
  if i>m then     # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}
```



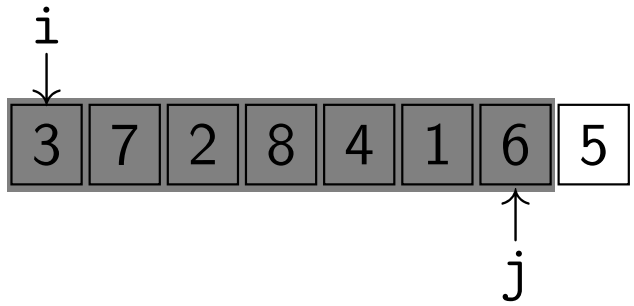
l=0

r=7

m=3

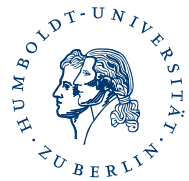
```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



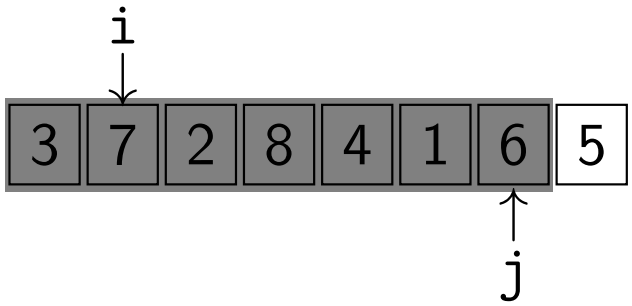
val=5

pos=



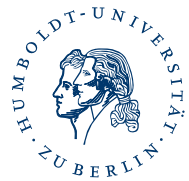

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



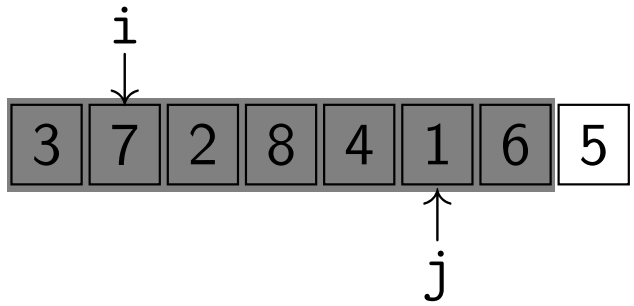
val=5

pos=



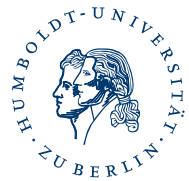
```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



val=5

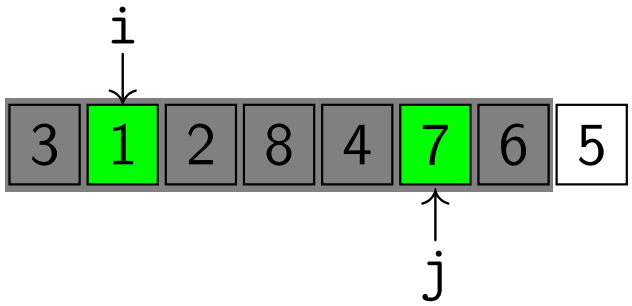
pos=



QuickSort

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

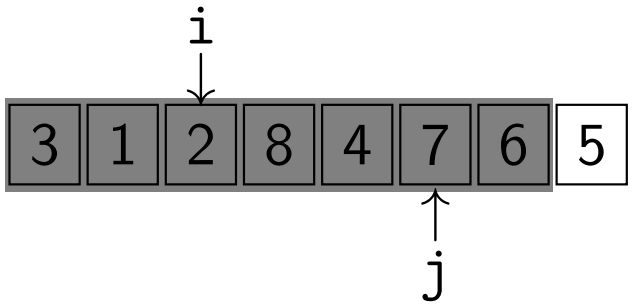


val=5

pos=

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

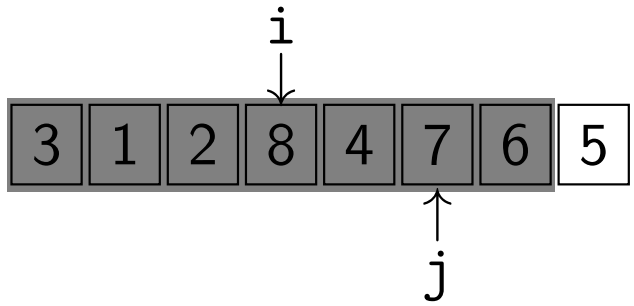


val=5

pos=

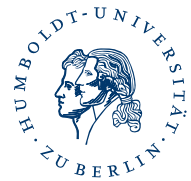
```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



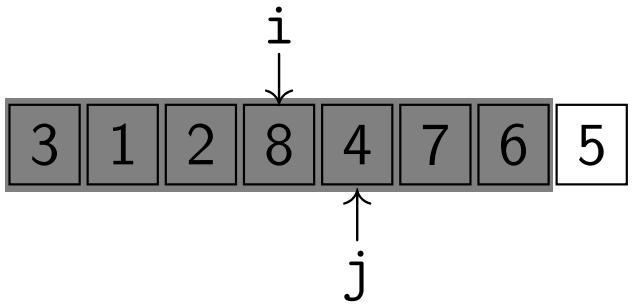
val=5

pos=



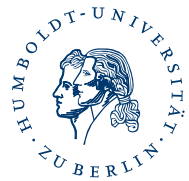
```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



val=5

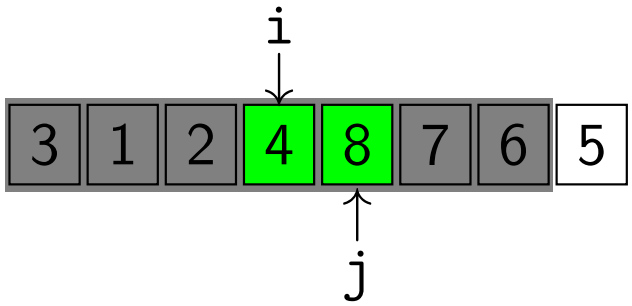
pos=



QuickSort

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

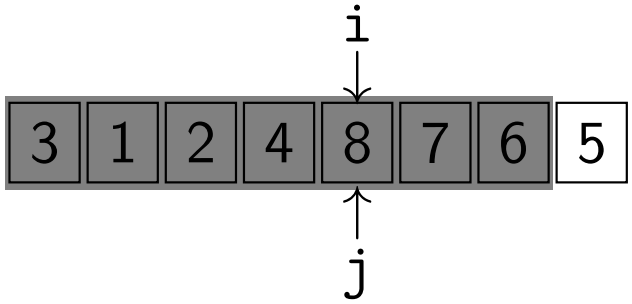


val=5

pos=

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

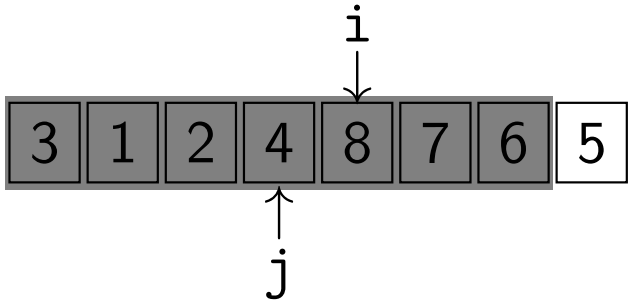


val=5

pos=

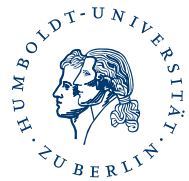
```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



val=5

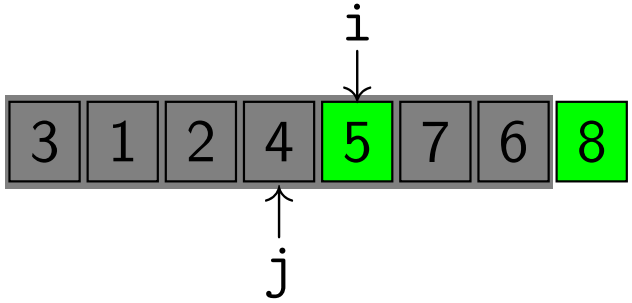
pos=



QuickSort

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

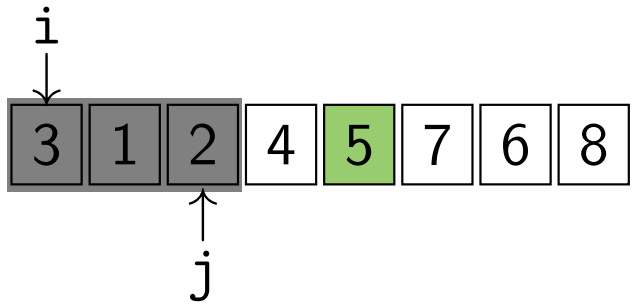


val=5

pos=5


```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



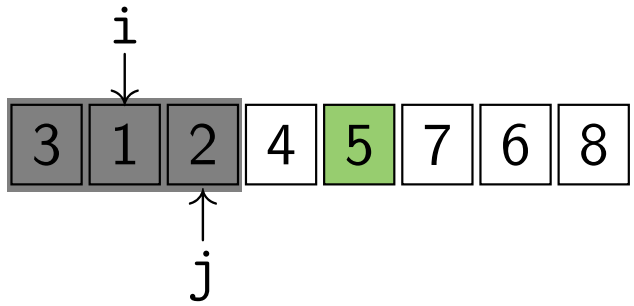
val=4

pos=

QuickSort

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

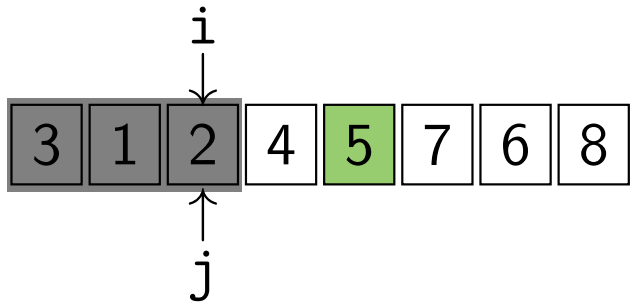


val=4

pos=

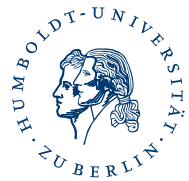

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



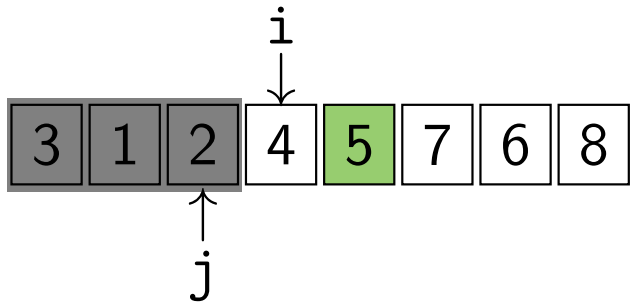
val=4

pos=



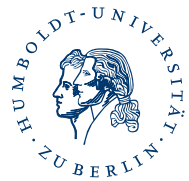

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



val=4

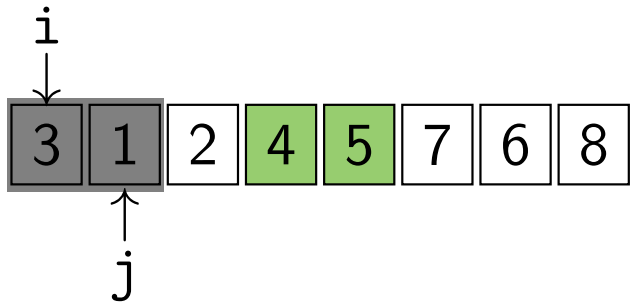
pos=



QuickSort

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

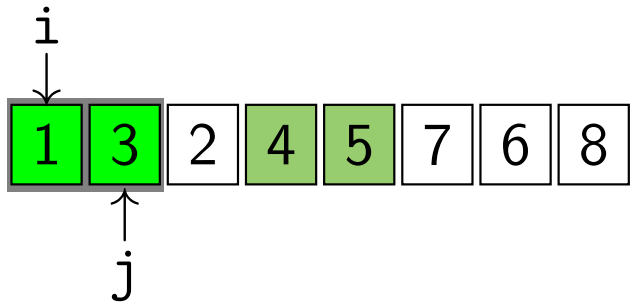


val=2

pos=


```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



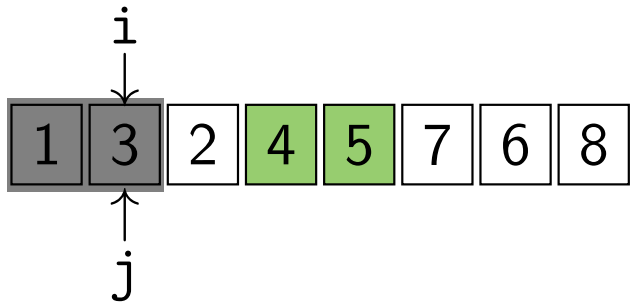
val=2

pos=

QuickSort

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

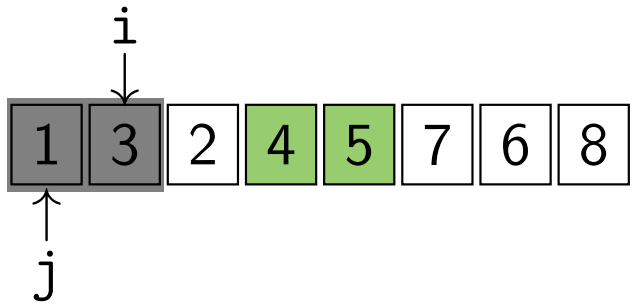


val=2

pos=

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

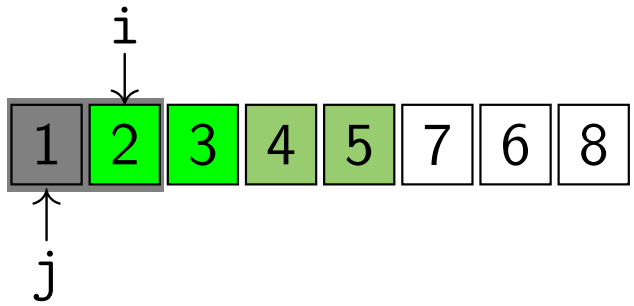


val=2

pos=2


```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



val=

pos=


```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

1	2	3	4	5	7	6	8

val=

pos=

QuickSort

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

1	2	3	4	5	7	6	8

val=

pos=


```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

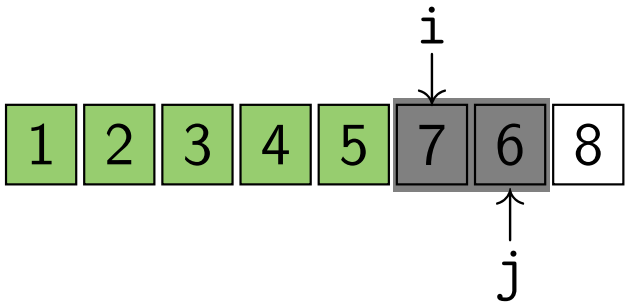
1	2	3	4	5	7	6	8

val=

pos=

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

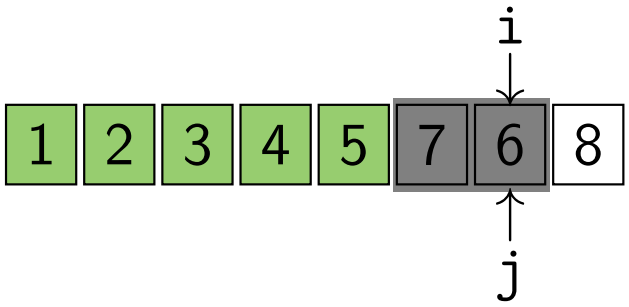


val=8

pos=


```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

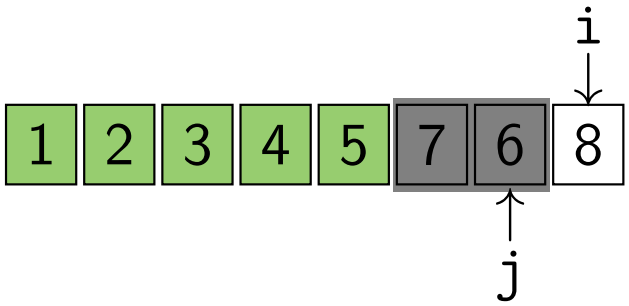


val=8

pos=

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

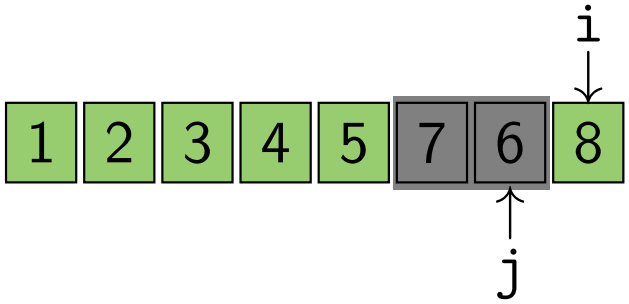


val=8

pos=

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

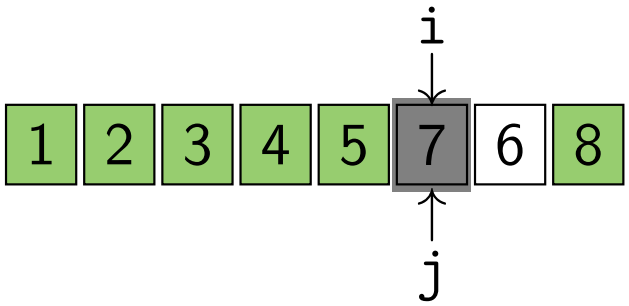


val=8

pos=8

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

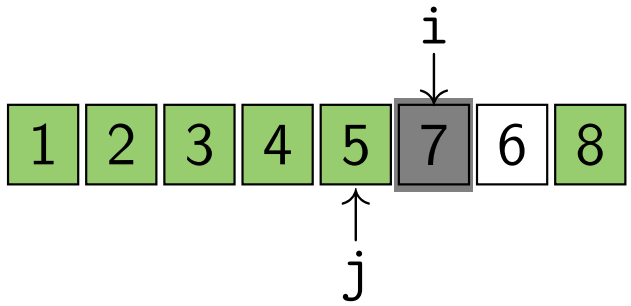


val=6

pos=

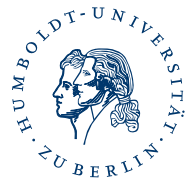

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



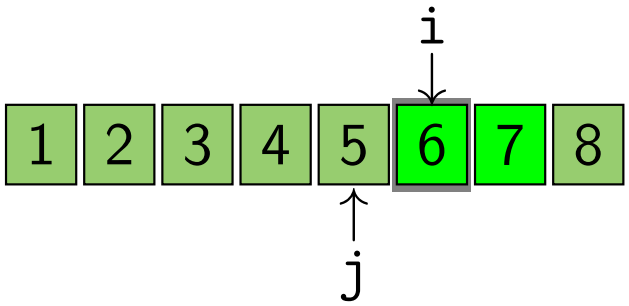
val=6

pos=



```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```



val=6

pos=

```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

1	2	3	4	5	6	7	8

val=

pos=


```
1. func void qsort(S array;
2.                 l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                 l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i>=j;
17.  swap( S[i], S[r]);
18.  return i;
19.}
```

1	2	3	4	5	6	7	8

val=

pos=