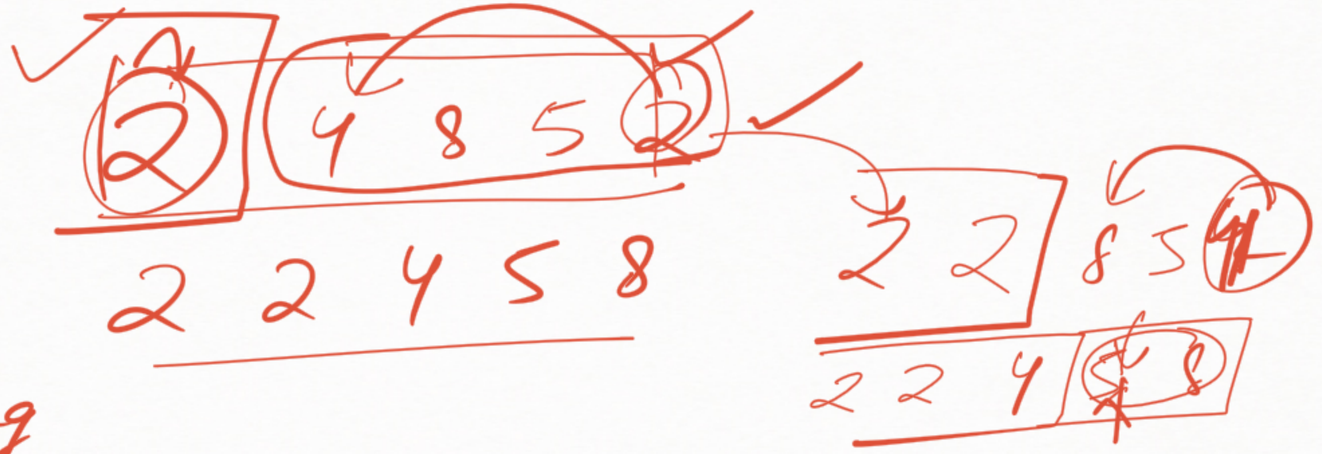
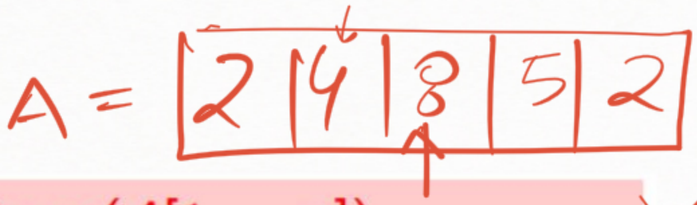
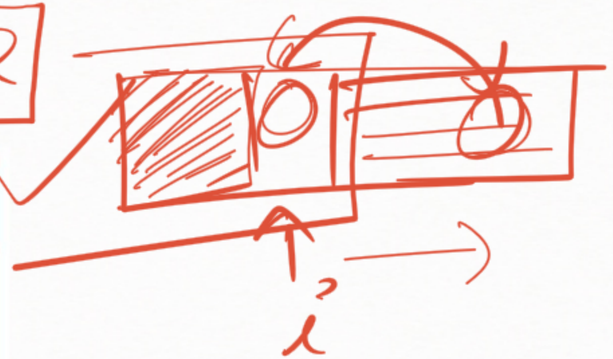
Sorting



selection sorting

1. Find a minimum by scanning the array.
2. Swap it with the first element
3. Repeat with the remaining part of the array.





SelectionSort($A[1 \ldots n]$)

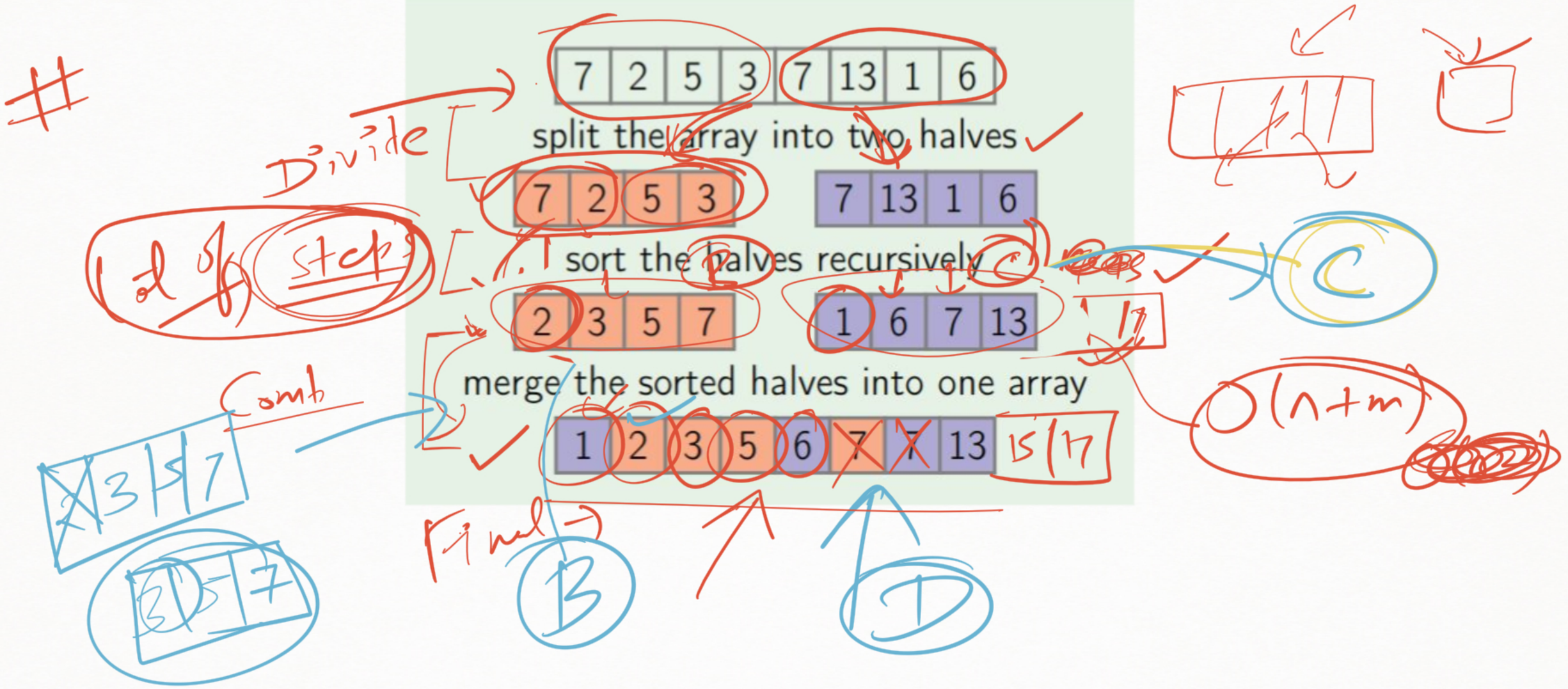
```
for i from 1 to n:
  minIndex ← i
  for j from i + 1 to n:
    if A[j] < A[minIndex]:
      minIndex ← j
  {A[minIndex] = min A[i ... n]}
  swap(A[i], A[minIndex])
  {A[1 ... i] is in final position}
```

$n \times$

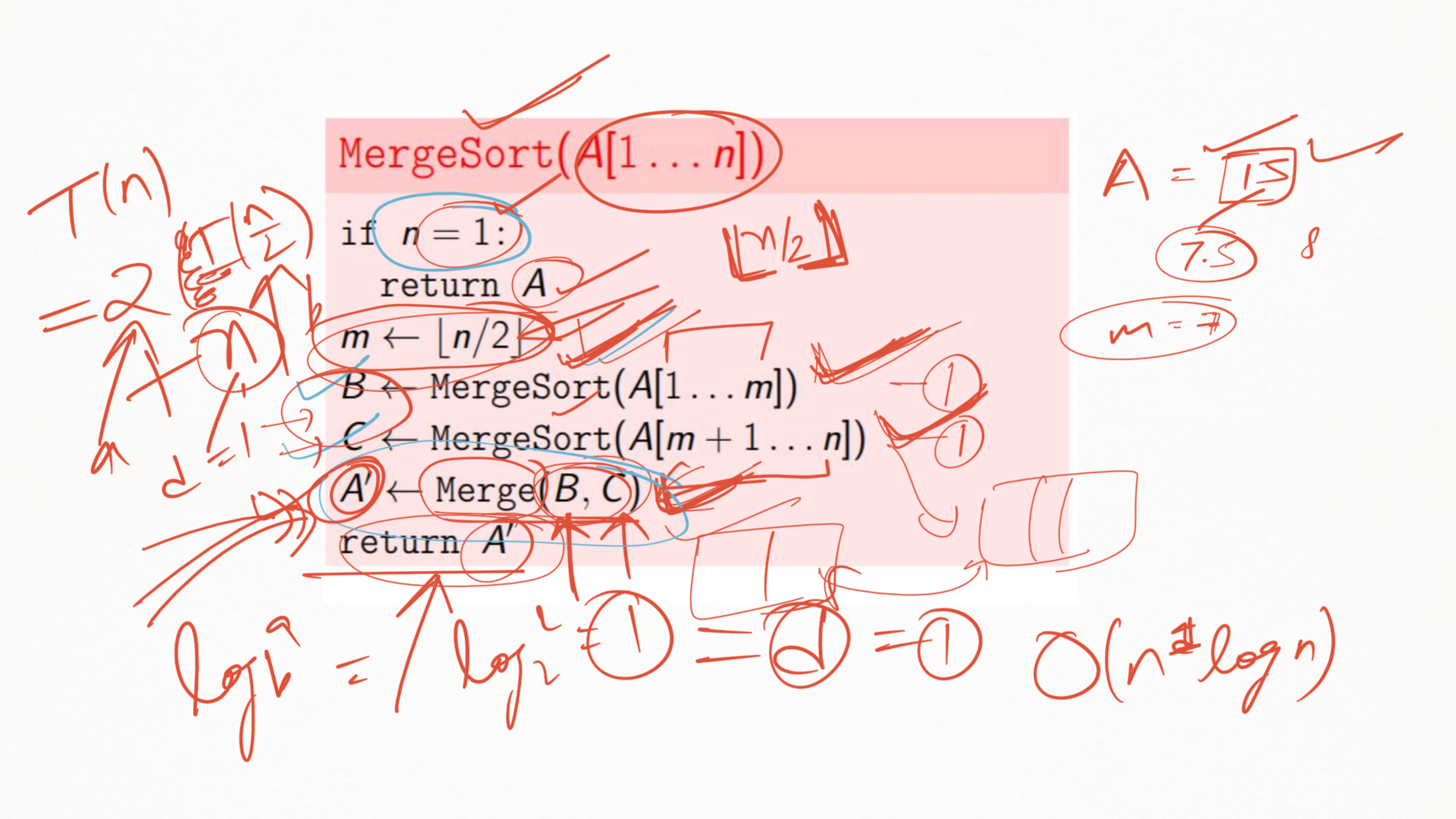
$n + n - 1, n - 2$

$O(n^2)$ $\frac{n^2}{2}$

Example: merge sort



```
MergeSort(A[1...n])
if n = 1:
    return A
m ← ⌊n/2⌋
B ← MergeSort(A[1...m])
C ← MergeSort(A[m+1...n])
A' ← Merge(B, C)
return A'
```

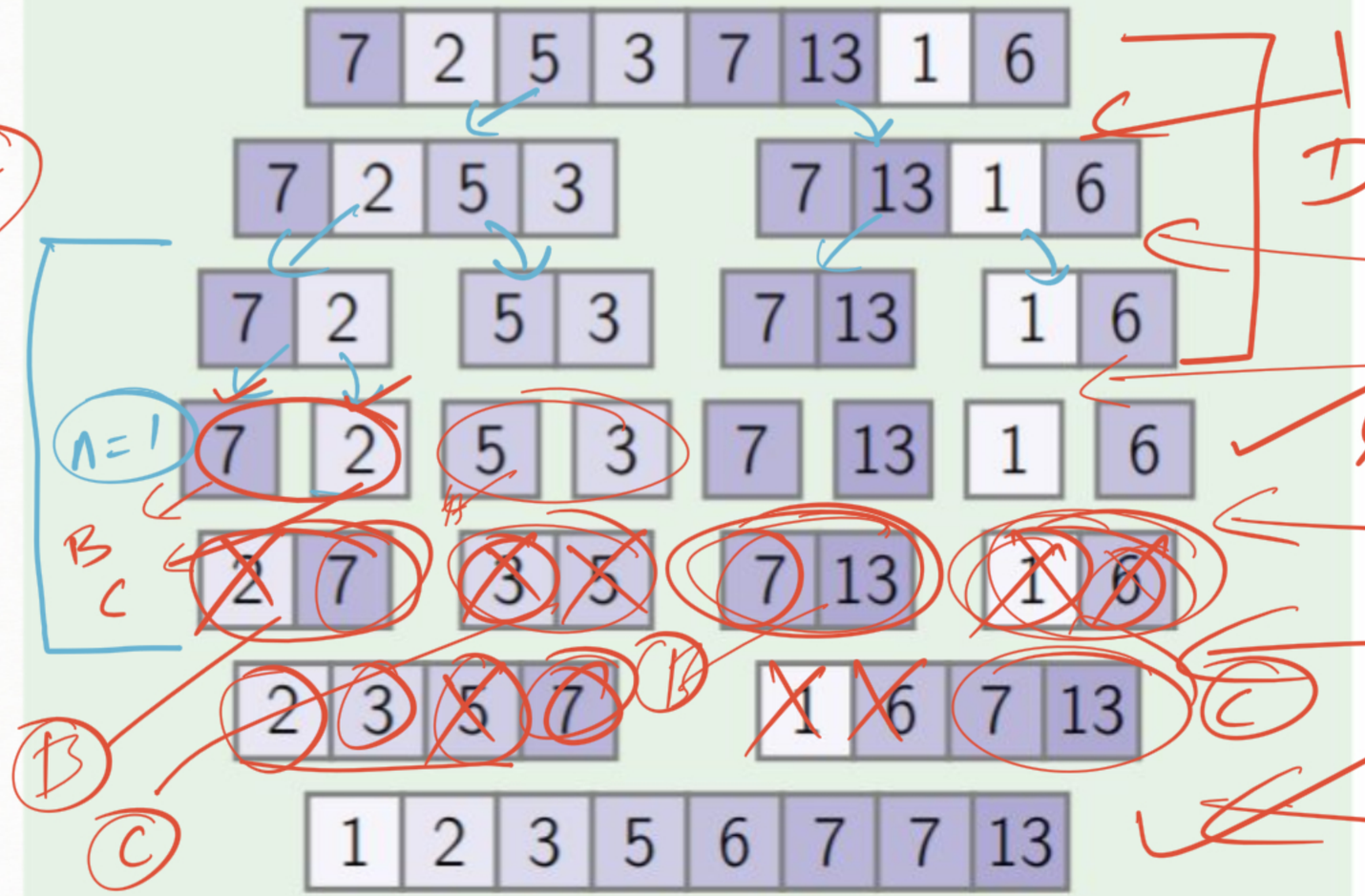


Merging Two Sorted Arrays


```
Merge(B[1...p], C[1...q])
{B and C are sorted}
D ← empty array of size p + q
while B and C are both non-empty:
  b ← the first element of B
  c ← the first element of C
  if b ≤ c:
    move b from B to the end of D
  else:
    move c from C to the end of D
move the rest of B and C to the end of D
return D
```

Outside the while

Merge sort: example



$O(n \log n)$

$\log_2 n$ steps

not shown earlier

1st Divide

Divide

2nd D

3rd D

Quit

1st merge

2nd merge

3rd merge

Quick Sort $\Theta(n \log n)$ ✓

$O(n^2)$

Example: quick sort

indexed

6	4	8	2	9	3	9	4	7	6	1

partition with respect to $x = A[1]$

in particular, x is in its final position

1	4	2	3	4	6	6	9	7	8	9

sort the two parts recursively

1	2	3	4	4	6	6	7	8	9	9

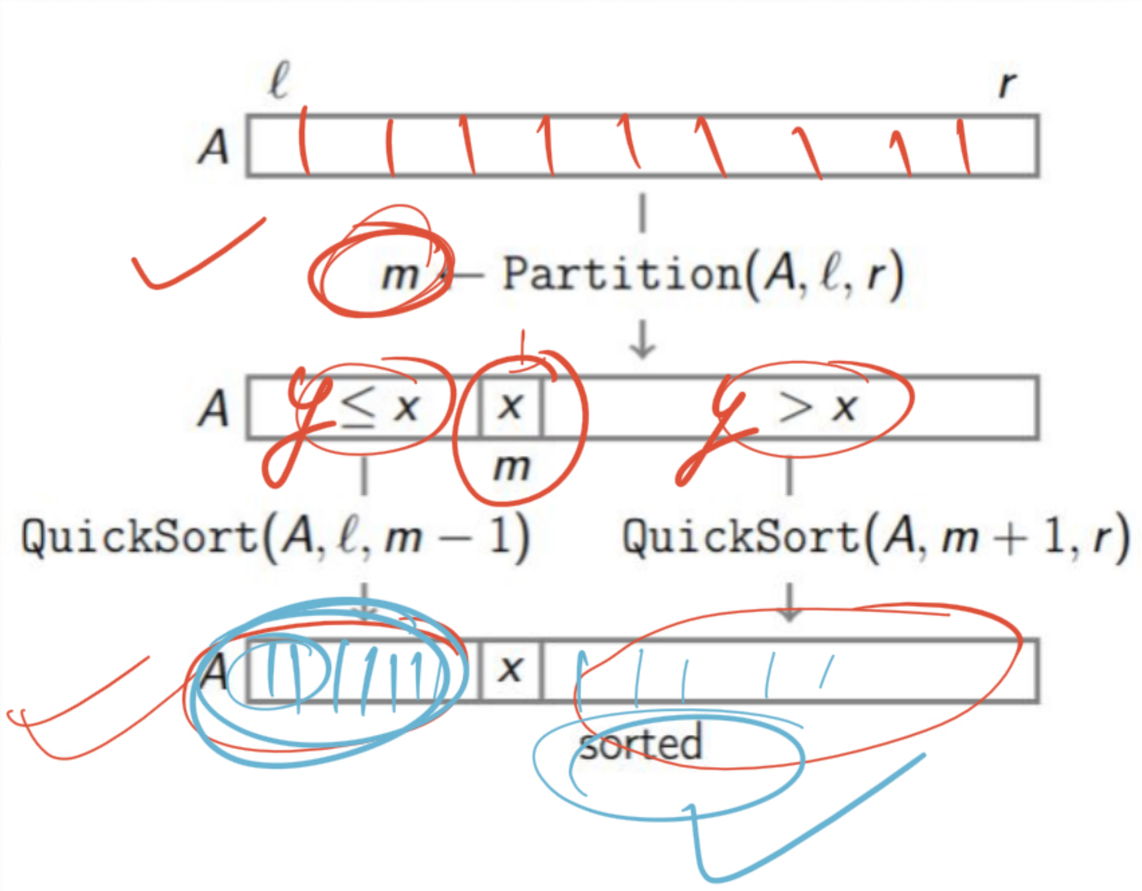
step

steps

$l \to m-1$

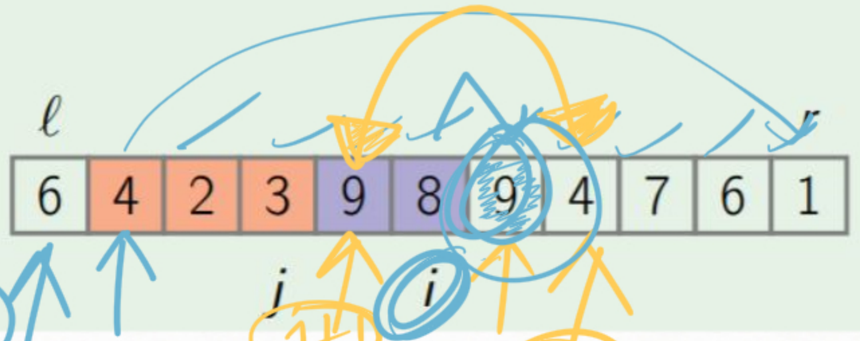
$m+1 \to r$

```
QuickSort(A, ℓ, r)
if ℓ ≥ r:
    return
m ← Partition(A, ℓ, r)
{A[m] is in the final position}
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```


ℓ
r
A
m ← Partition(A, ℓ, r)
A
≤ x
x
> x
m
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
A
x
sorted

Partitioning: example

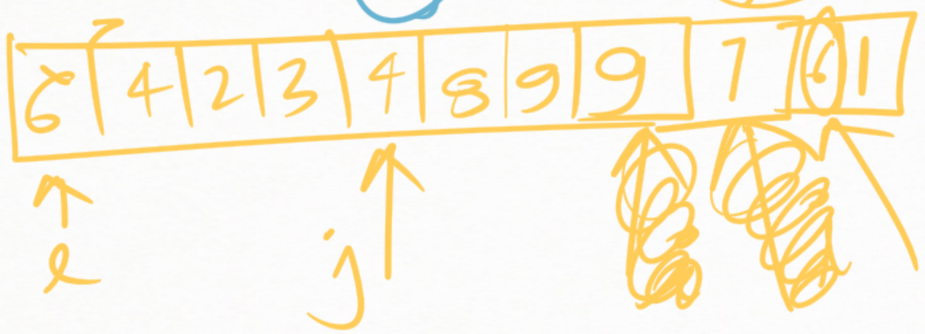
- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



one of middle step in quicksort

Algo

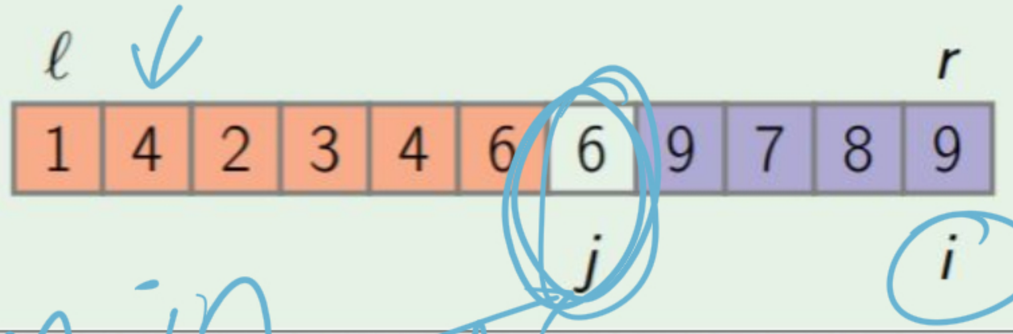
After swapping



l+1 to j
Red Area
j+1 to i
Blue Area

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$
- in the end, move $A[\ell]$ to its final place



Partition(A, ℓ, r)

$x \leftarrow A[\ell]$ {pivot}
$j \leftarrow \ell$
for i from $\ell + 1$ to r:
  if $A[i] \leq x$:
    $j \leftarrow j + 1$
    swap $A[j]$ and $A[i]$
  $\{A[\ell + 1 \ldots j] \leq x,\ A[j + 1 \ldots i] > x\}$
swap $A[\ell]$ and $A[j]$
return j

m in QuickSort