Data Structures & Algorithms

23: Quick Sort

(Divide & Conquer: Partition-Exchange Sort)

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

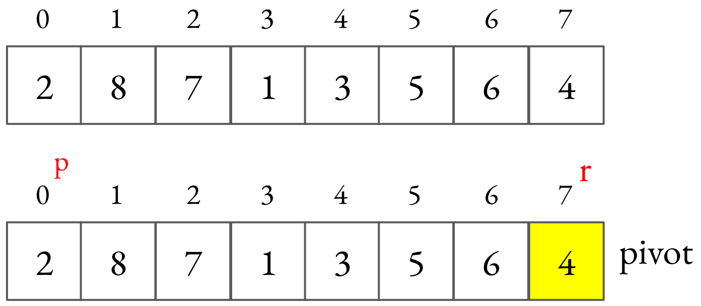
ram.krish@saiuniversity.edu.in

Quick Sort

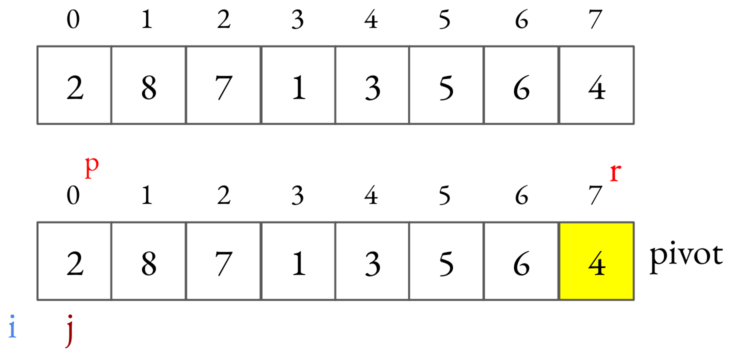
Quick Sort

0	1	2	3	4	5	6	7
2	8	7	1	3	5	6	4

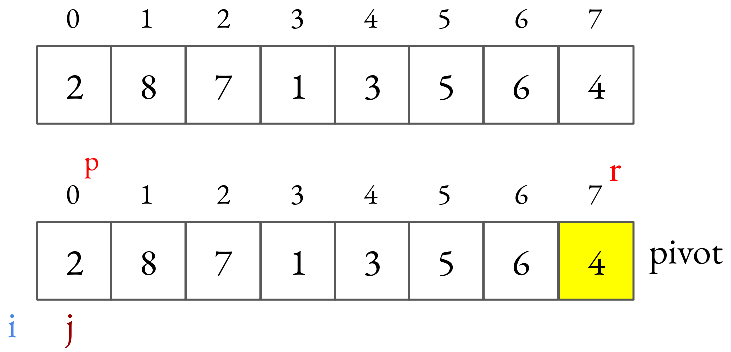
Quick Sort

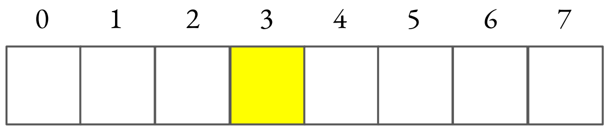


Quick Sort

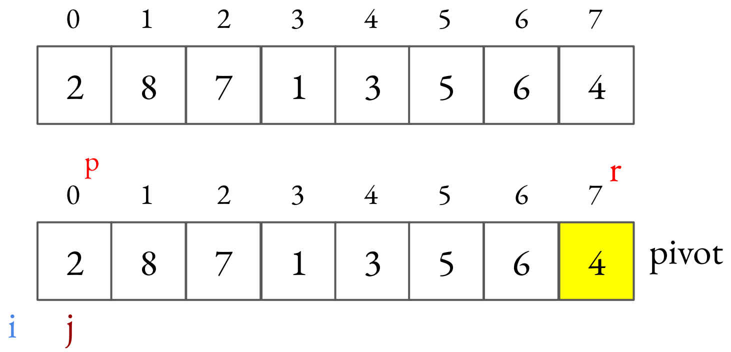


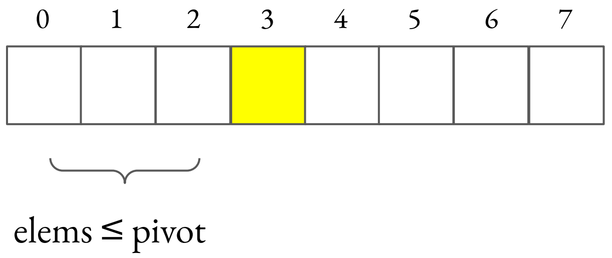
Quick Sort



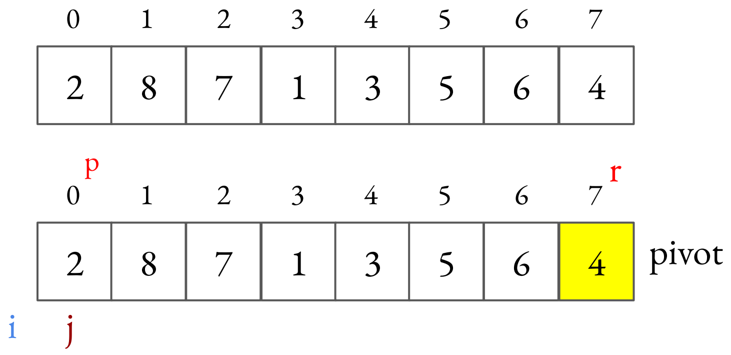


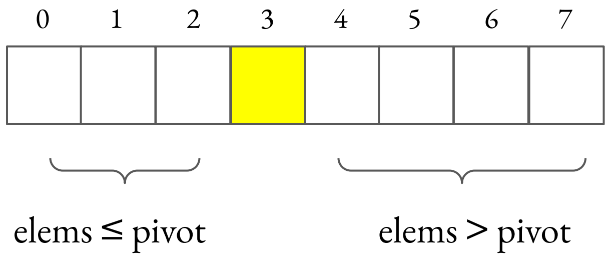
Quick Sort


0 1 2 3 4 5 6 7
2 8 7 1 3 5 6 4
p
r
0 1 2 3 4 5 6 7
2 8 7 1 3 5 6 4
pivot
i
j

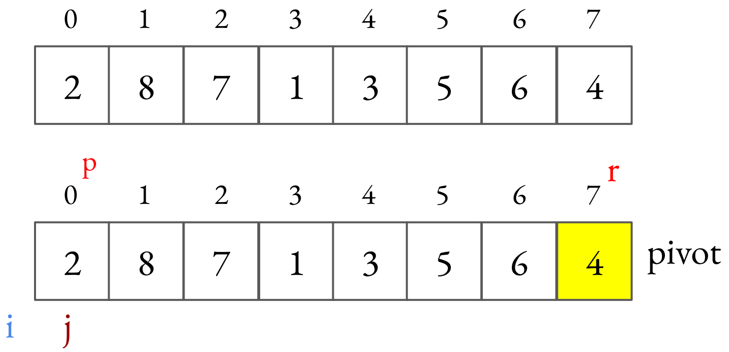

0 1 2 3 4 5 6 7
elems ≤ pivot

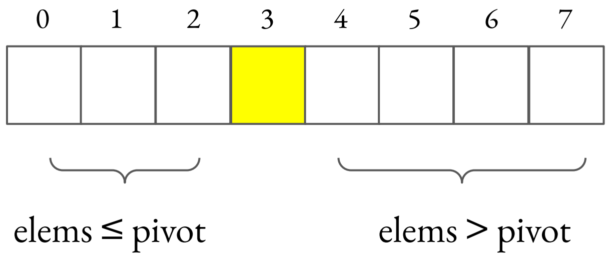
Quick Sort


0
1
2
3
4
5
6
7
2
8
7
1
3
5
6
4
p
r
0
1
2
3
4
5
6
7
2
8
7
1
3
5
6
4
pivot
i
j


0
1
2
3
4
5
6
7
elems ≤ pivot
elems > pivot

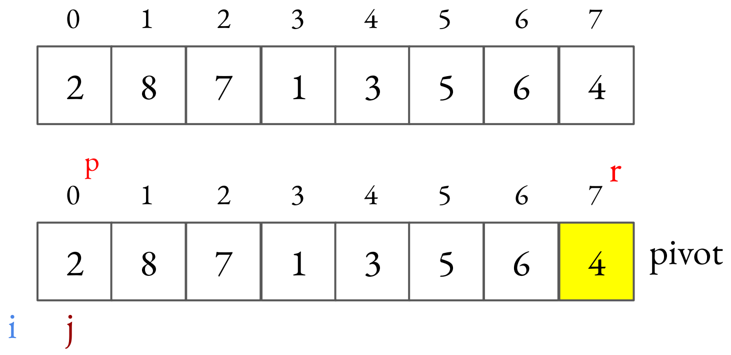
Quick Sort

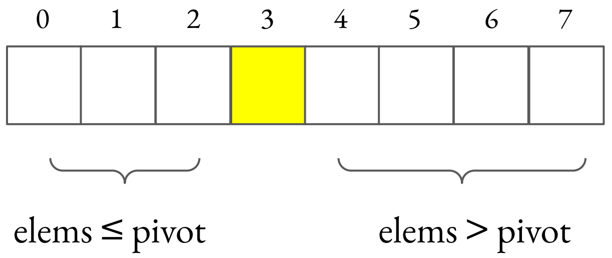




```
x = A[r]
x = 4
```

Quick Sort

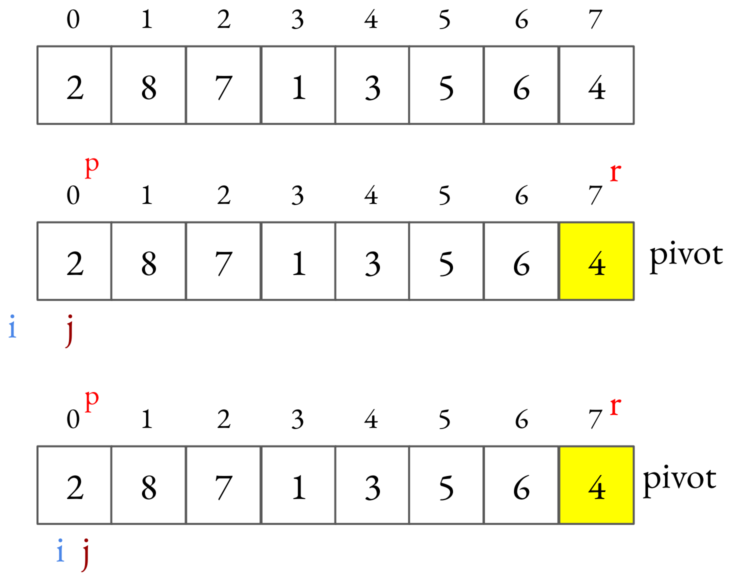


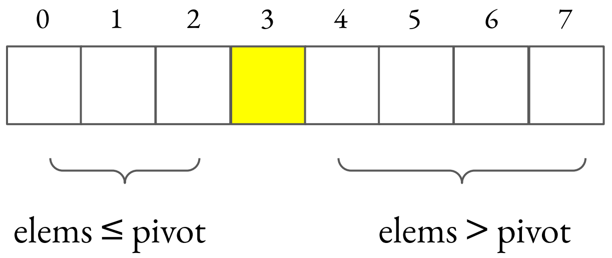


```
        x = A[r]
        x = 4

for j = p to r-1
        if(A[j] ≤ x)
                i = i + 1
                swap(A[i], A[j])
```

Quick Sort

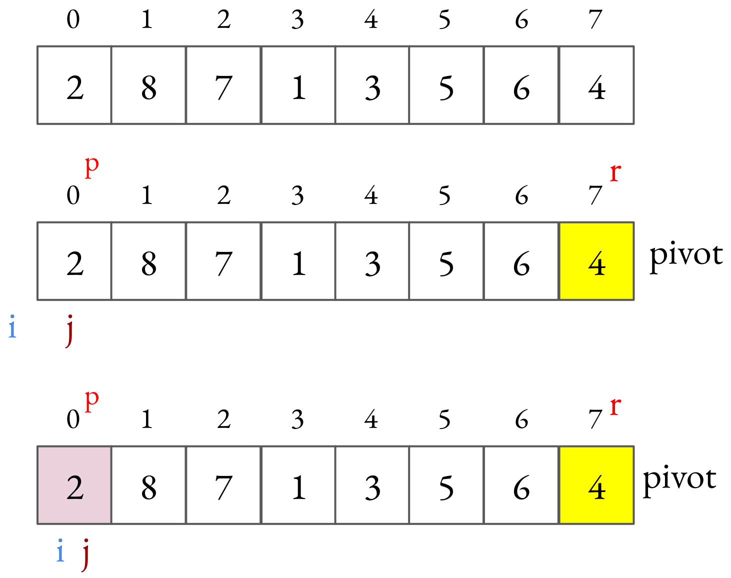


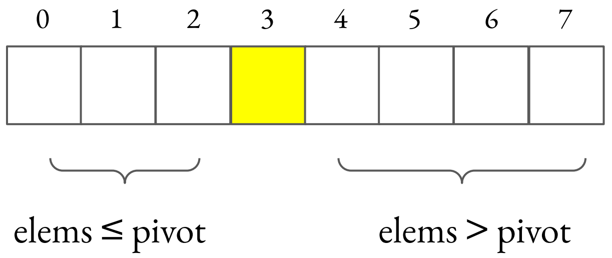


```
x = A[r]
x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

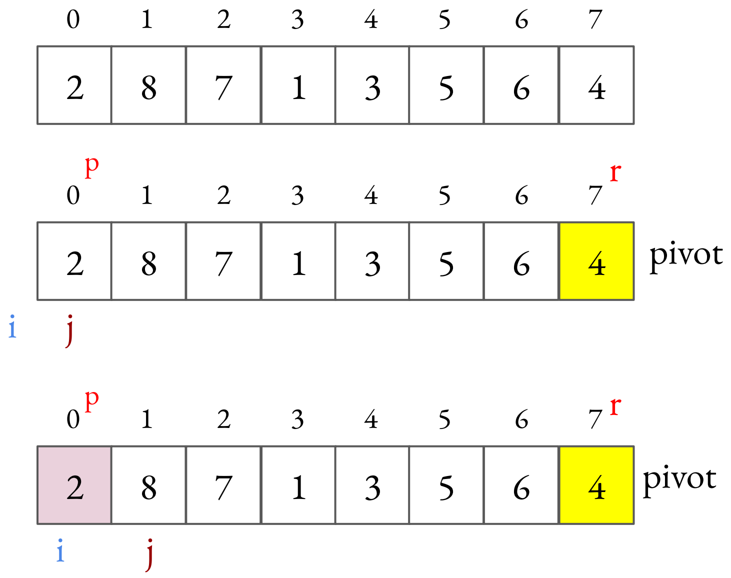


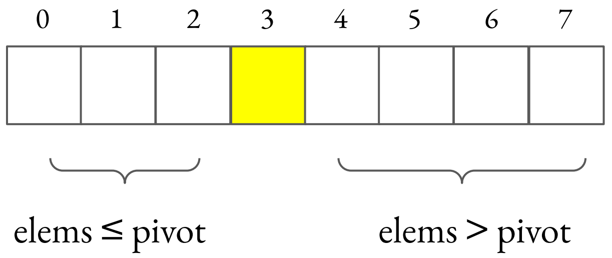


```
      x = A[r]
      x = 4

for j = p to r-1
      if(A[j] ≤ x)
            i = i + 1
            swap(A[i], A[j])
```

Quick Sort

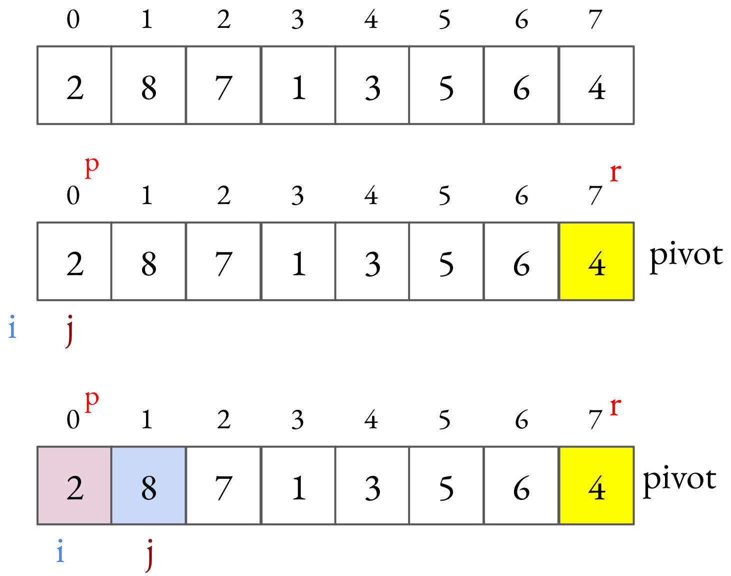


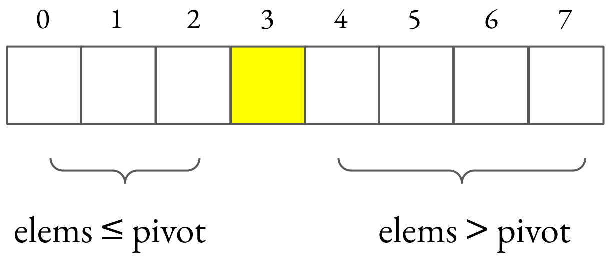


```
x = A[r]
x = 4
```

```
for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

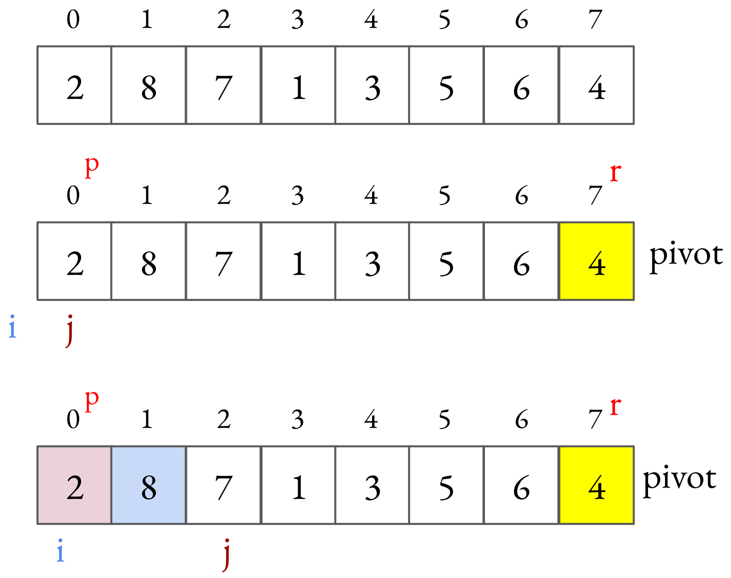


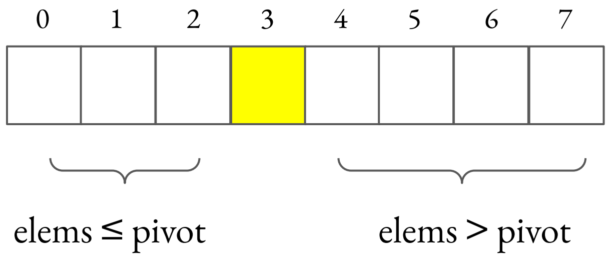


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

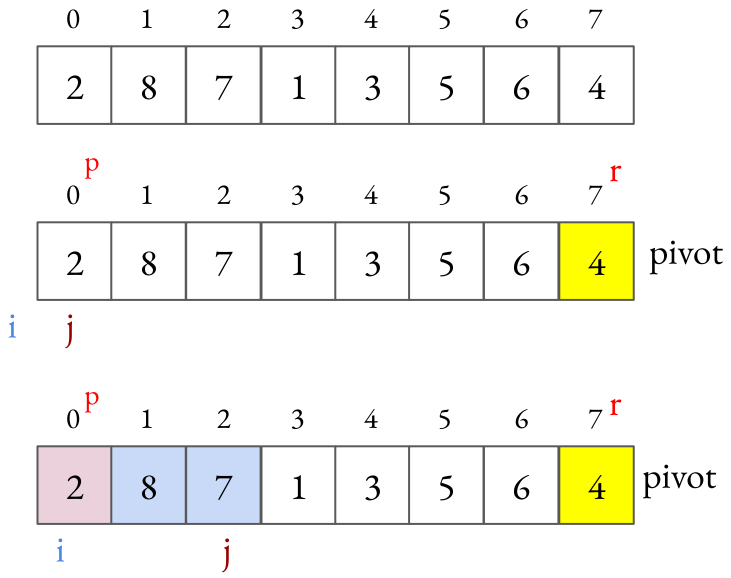


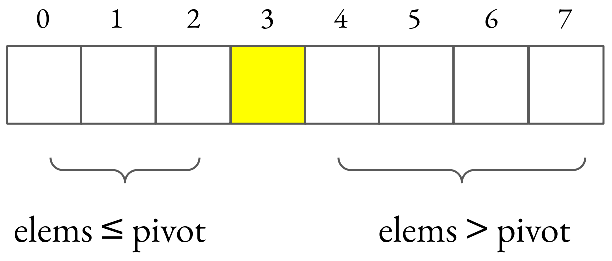


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

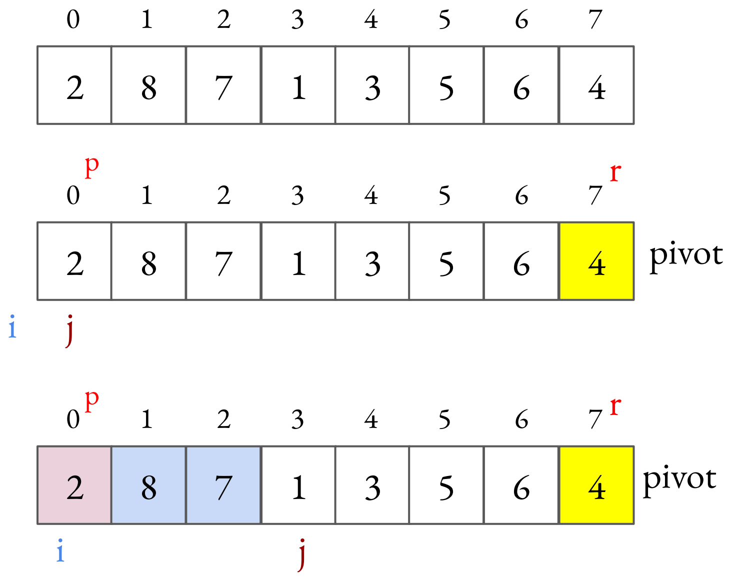


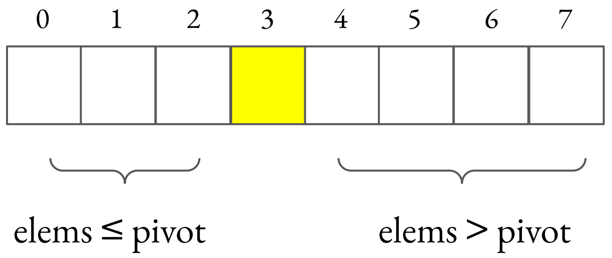


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

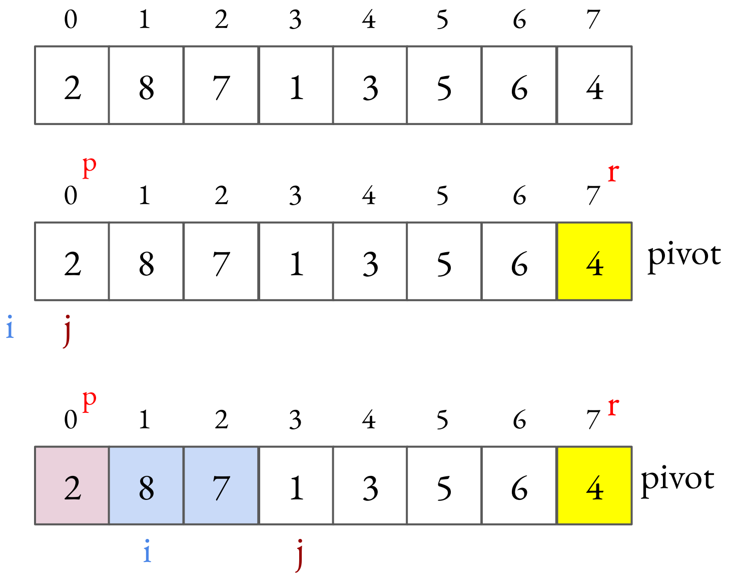


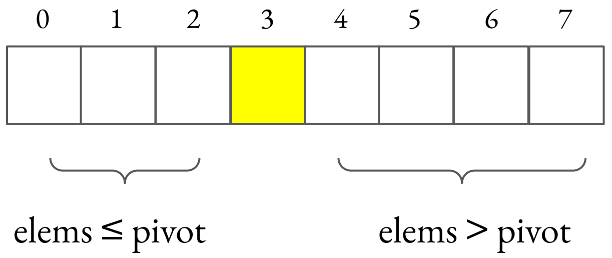


```
x = A[r]
x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

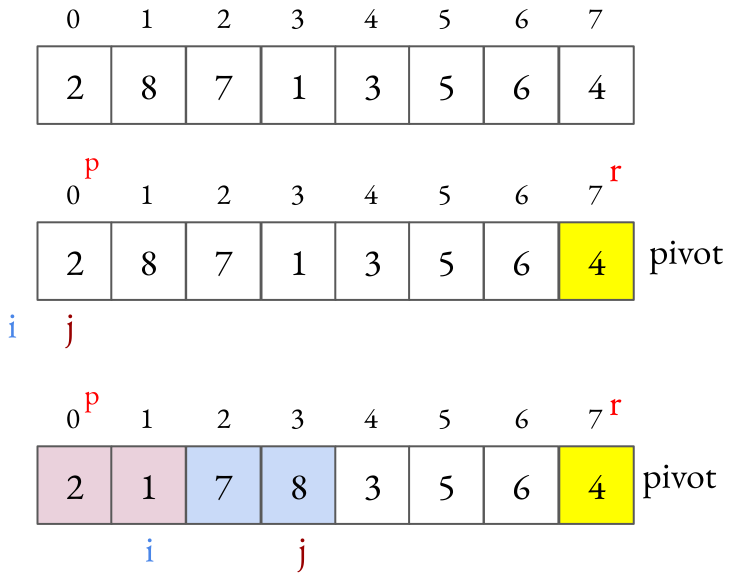


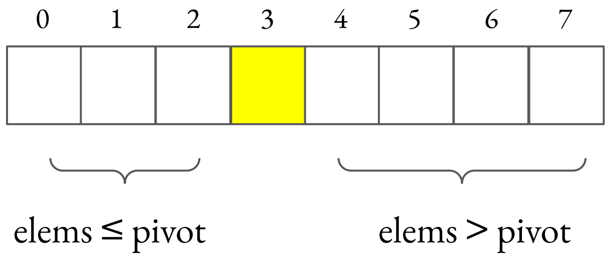


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

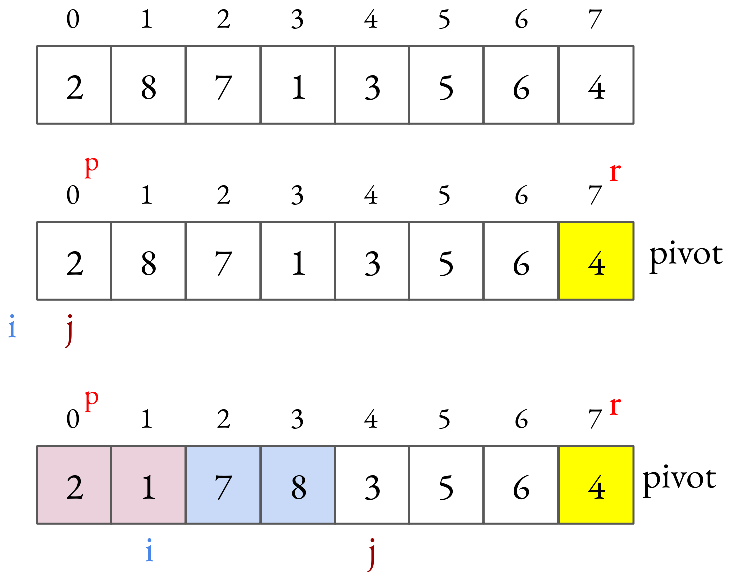


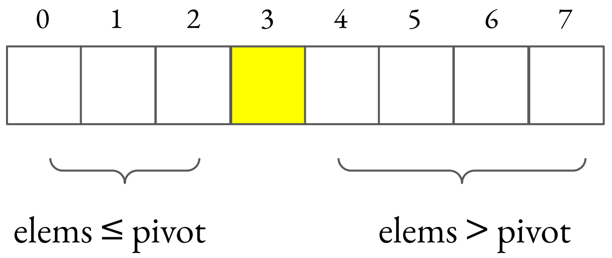


```
x = A[r]
x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

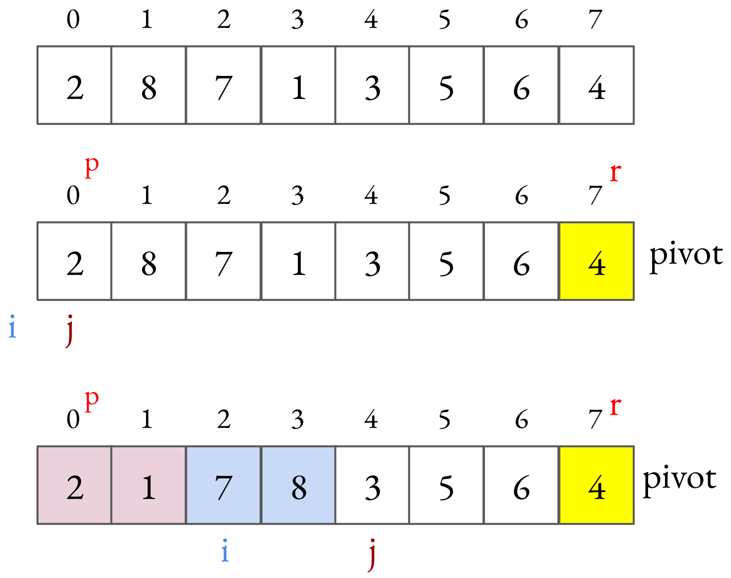


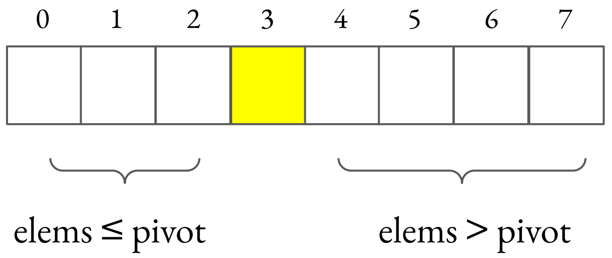


```
x = A[r]
x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

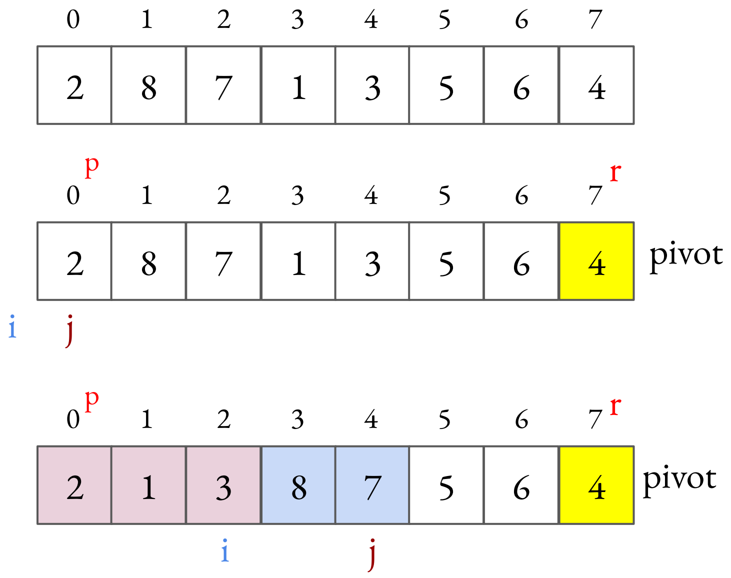


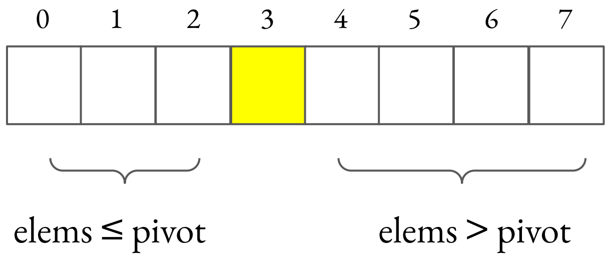


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

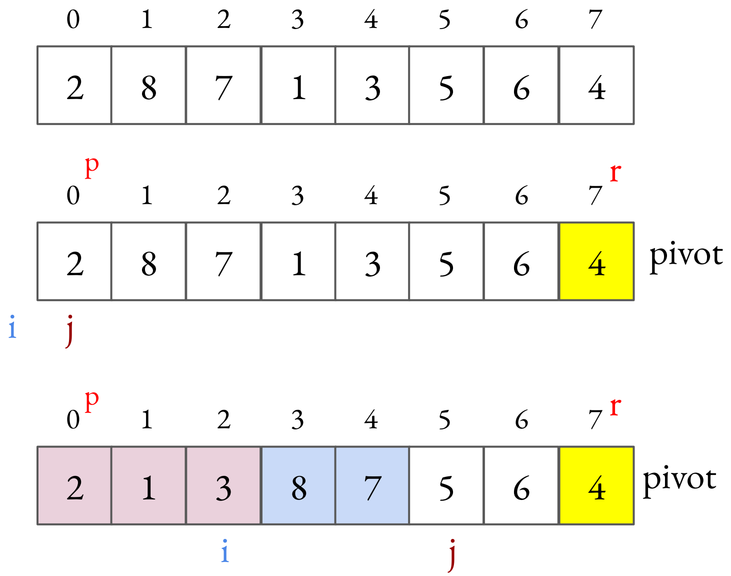


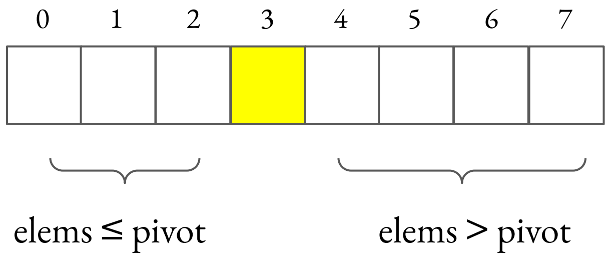


```
x = A[r]
x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

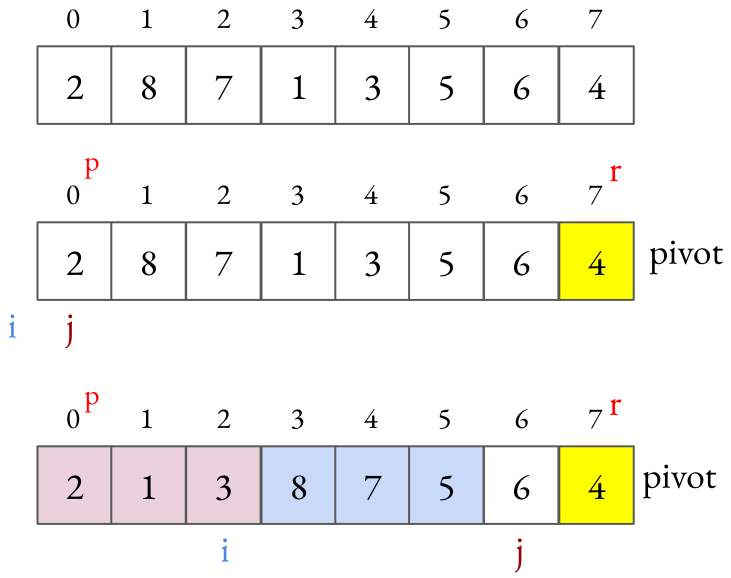


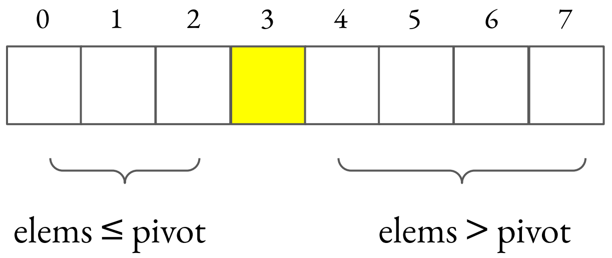


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

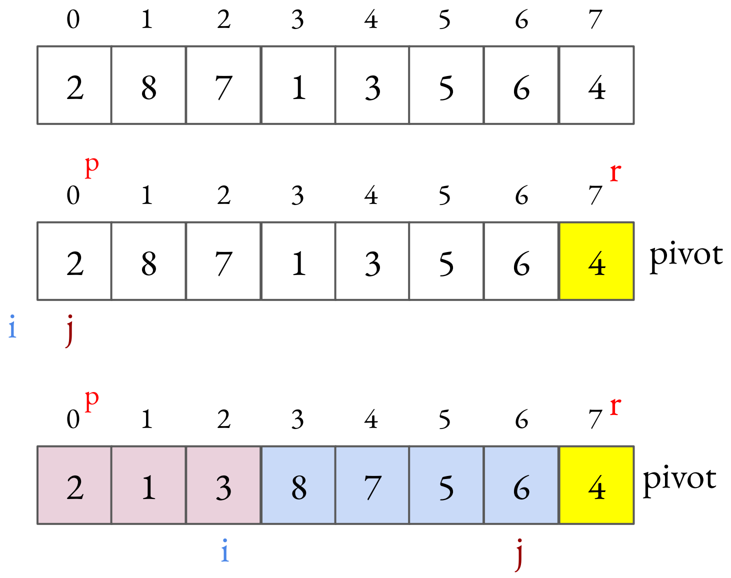


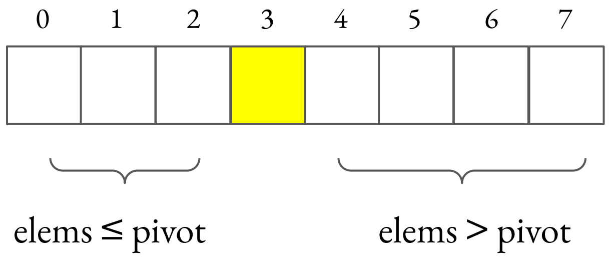


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort

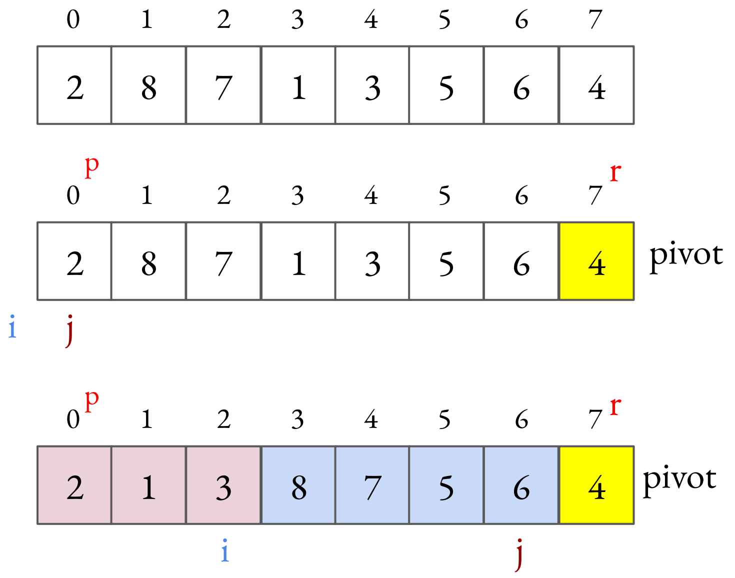


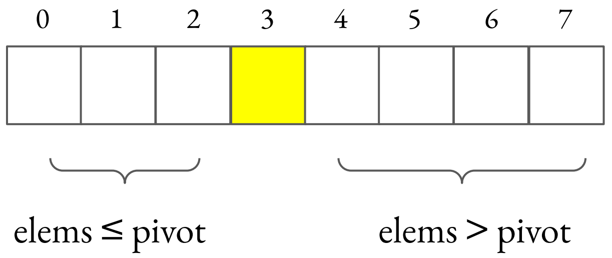


```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])
```

Quick Sort



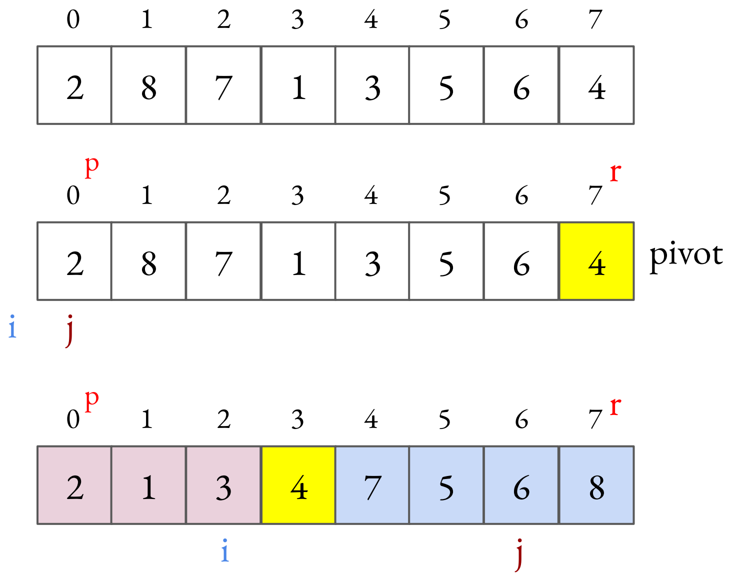


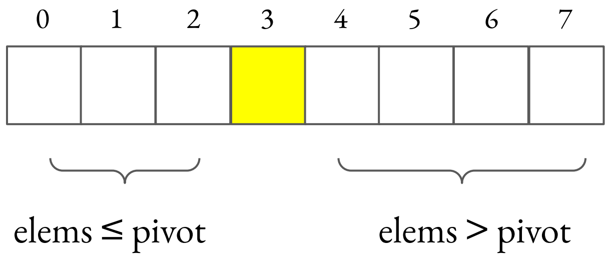
```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])

swap(A[i+1], A[r])
```

Quick Sort



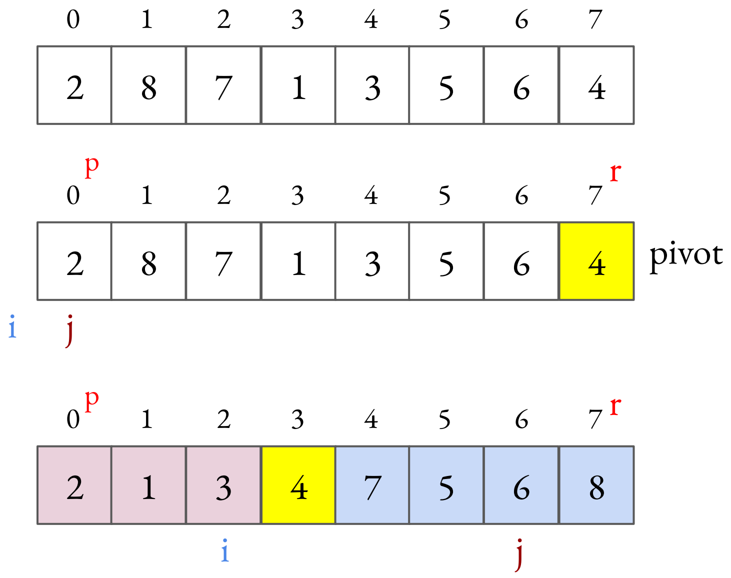


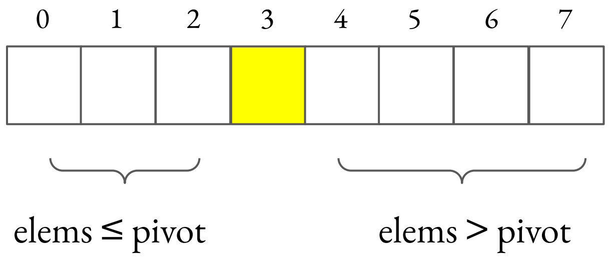
```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])

swap(A[i+1], A[r])
```

Quick Sort





```
    x = A[r]
    x = 4

for j = p to r-1
    if(A[j] ≤ x)
        i = i + 1
        swap(A[i], A[j])

swap(A[i+1], A[r])
return i+1
```

Quick Sort

Quicksort is based on the three-step process of divide-and-conquer.

- To sort the subarray $A[p:r]$:

  Divide: Partition $A[p:r]$, into two (possibly empty) subarrays $A[p:q-1]$ and $A[q+1:r]$, such that each element in the first subarray $A[p:q-1]$ is $\leq A[q]$ and $A[q]$ is $\leq$ each element in the second subarray $A[q+1:r]$.

  Conquer: Sort the two subarrays by recursive calls to QUICKSORT.

  Combine: No work is needed to combine the subarrays, because they are sorted in place.

- Perform the divide step by a procedure PARTITION, which returns the index q that marks the position separating the subarrays.

Quick Sort

```
PARTITION$(A, p, r)$
  $x = A[r]$                            // the pivot
  $i = p - 1$                           // highest index into the low side
  for $j = p$ to $r - 1$                // process each element other than the pivot
      if $A[j] \leq x$                  // does this element belong on the low side?
          $i = i + 1$                   // index of a new slot in the low side
          exchange $A[i]$ with $A[j]$   // put this element there
  exchange $A[i + 1]$ with $A[r]$       // pivot goes just to the right of the low side
  return $i + 1$                        // new index of the pivot
```

QUICK SORT

```
QUICKSORT($A, p, r$)
  if $p < r$
      // Partition the subarray around the pivot, which ends up in $A[q]$.
      $q$ = PARTITION($A, p, r$)
      QUICKSORT($A, p, q - 1$)   // recursively sort the low side
      QUICKSORT($A, q + 1, r$)   // recursively sort the high side
```

Initial call is QUICKSORT($A, 1, n$).

Best case:	$\Omega(n \, log \, n)$
Worst case:	$O(n^2)$
Average case:	$\Theta(n \, log \, n)$