COMP 2611, DATA STRUCTURES
LECTURE 22

SORTING (Conclusion)

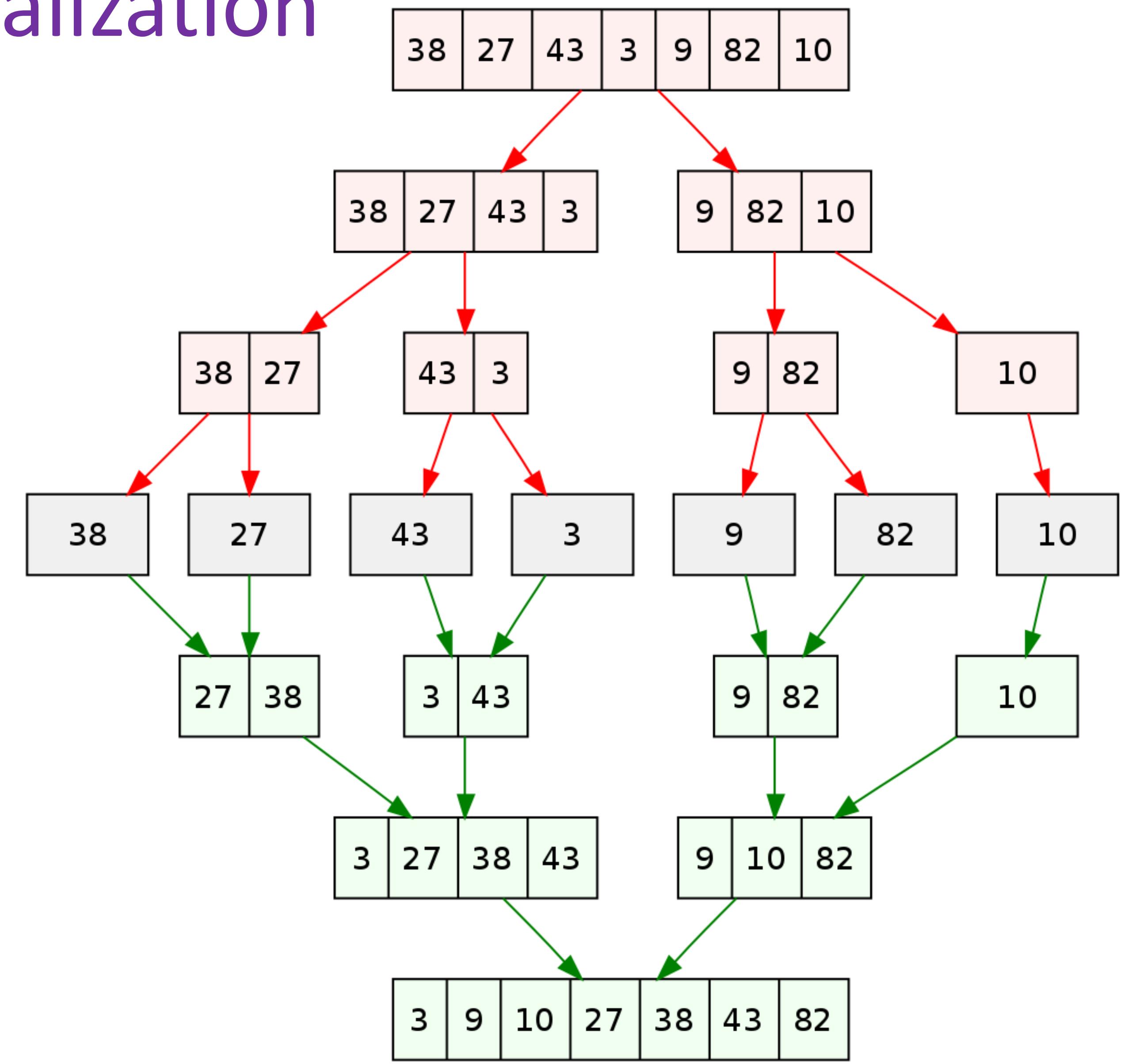
Sorting Algorithms

- Selection sort
- Bubble sort
- Insertion sort
- Heap sort
- Merge sort
- Quick sort

Mergesort Function

```
void mergeSort (int A[], int start, int end) {
   int mid;

   if (start < end) {
      mid = (start + end) / 2;
      mergeSort (A, start, mid);
      mergeSort (A, mid+1, end);
      merge (A, start, mid, end);
   }

}
```

Mergesort Visualization



Mergesort Animation

0	1	2	3	4	5	6
38	27	43	3	9	82	10

0	1	2	3
38	27	43	3

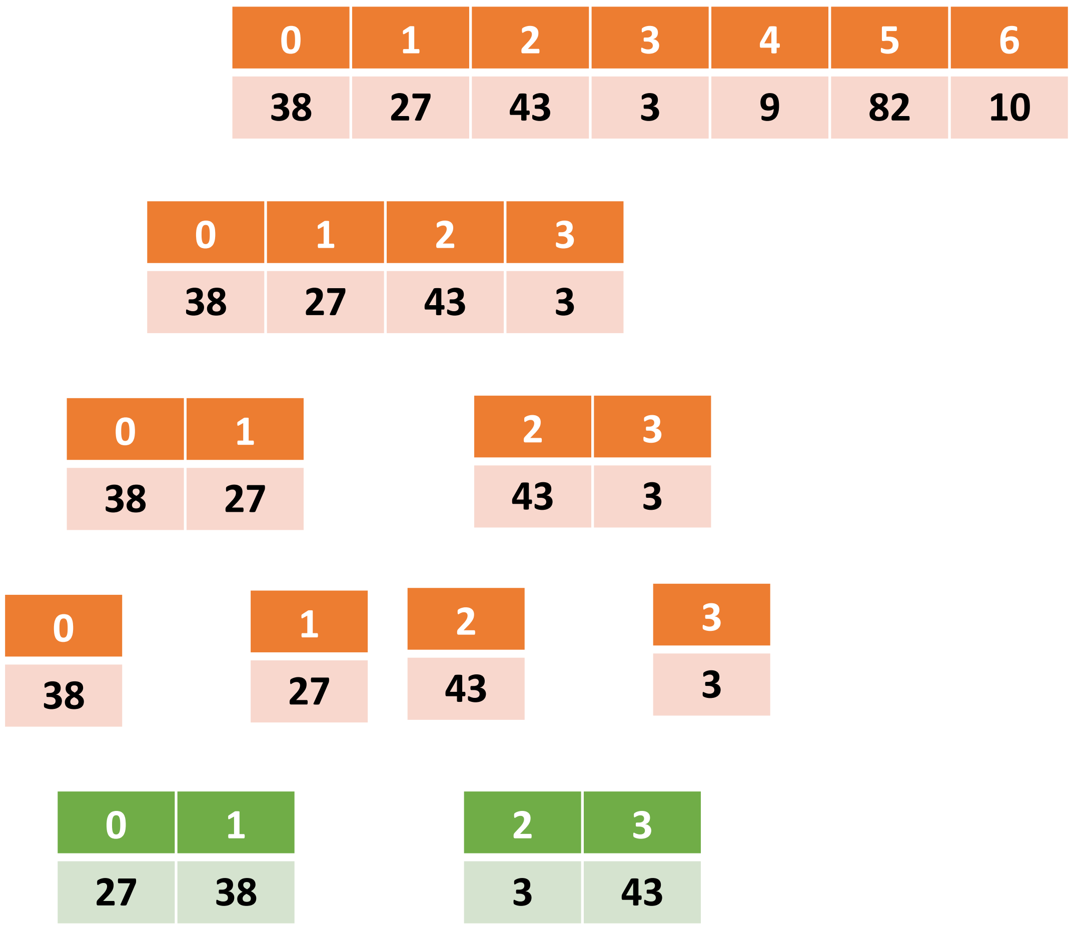
0	1
38	27

0
38

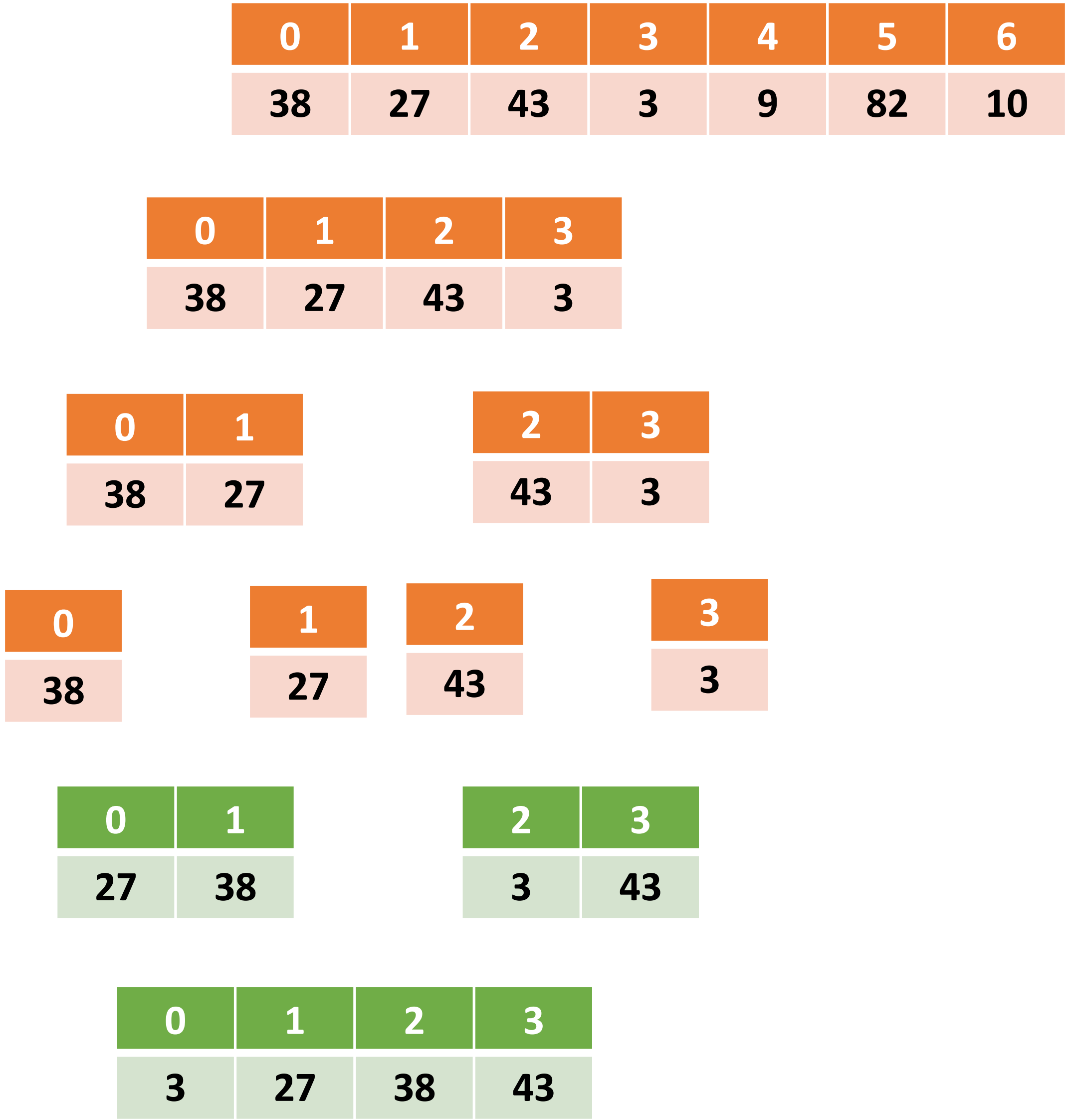
1
27

0	1
27	38

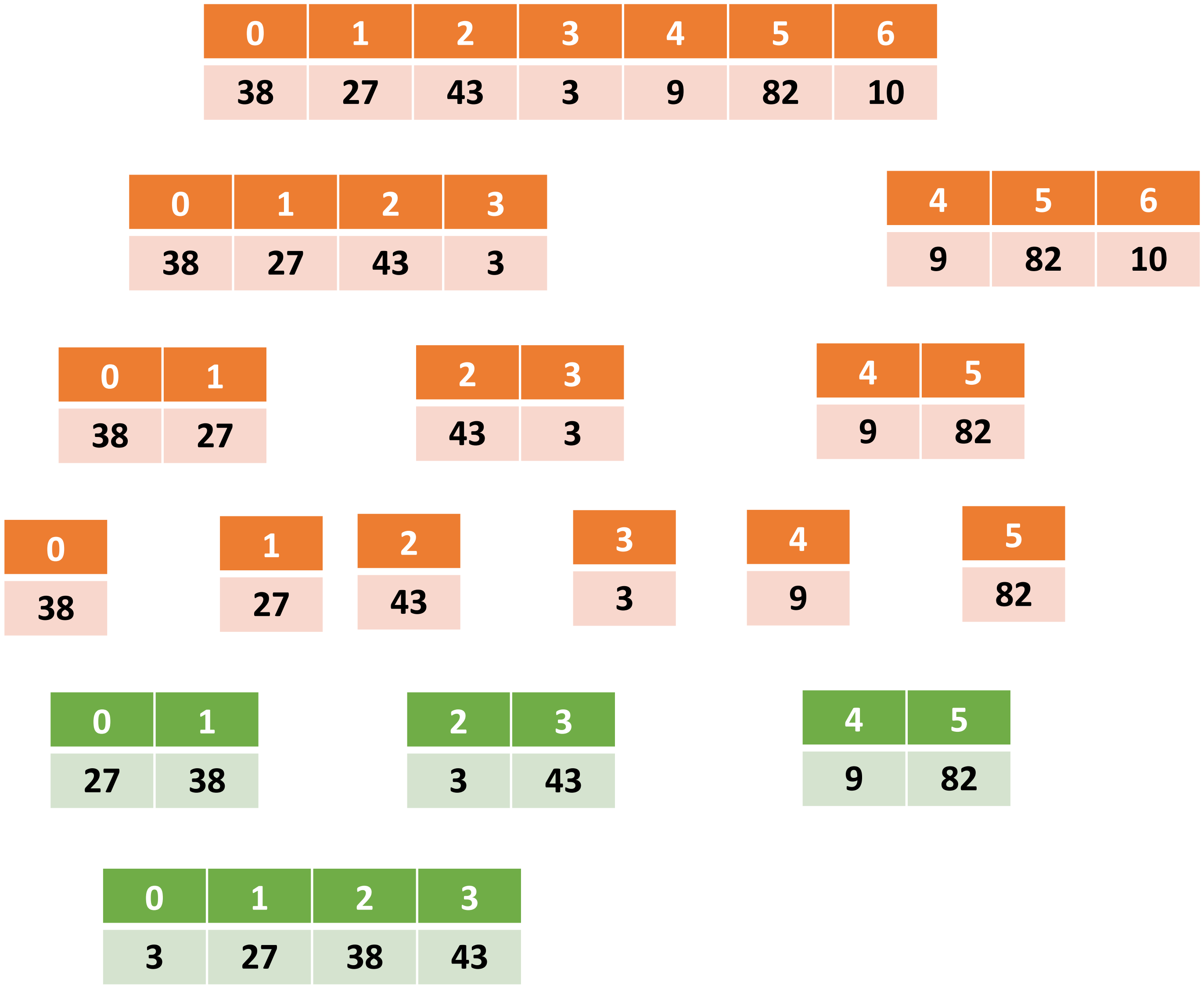
```
mergeSort (A, 0, 6):

mid = (0 + 6) / 2 = 3
mergeSort (A, 0, 3):

mid = (0 + 3) / 2 = 1
mergeSort (A, 0, 1):

mid = (0 + 1) / 2 = 0
mergeSort (A, 0, 0):
  terminates!
mergeSort (A, 1, 1):
  terminates!
merge (A, 0, 0, 1)
```



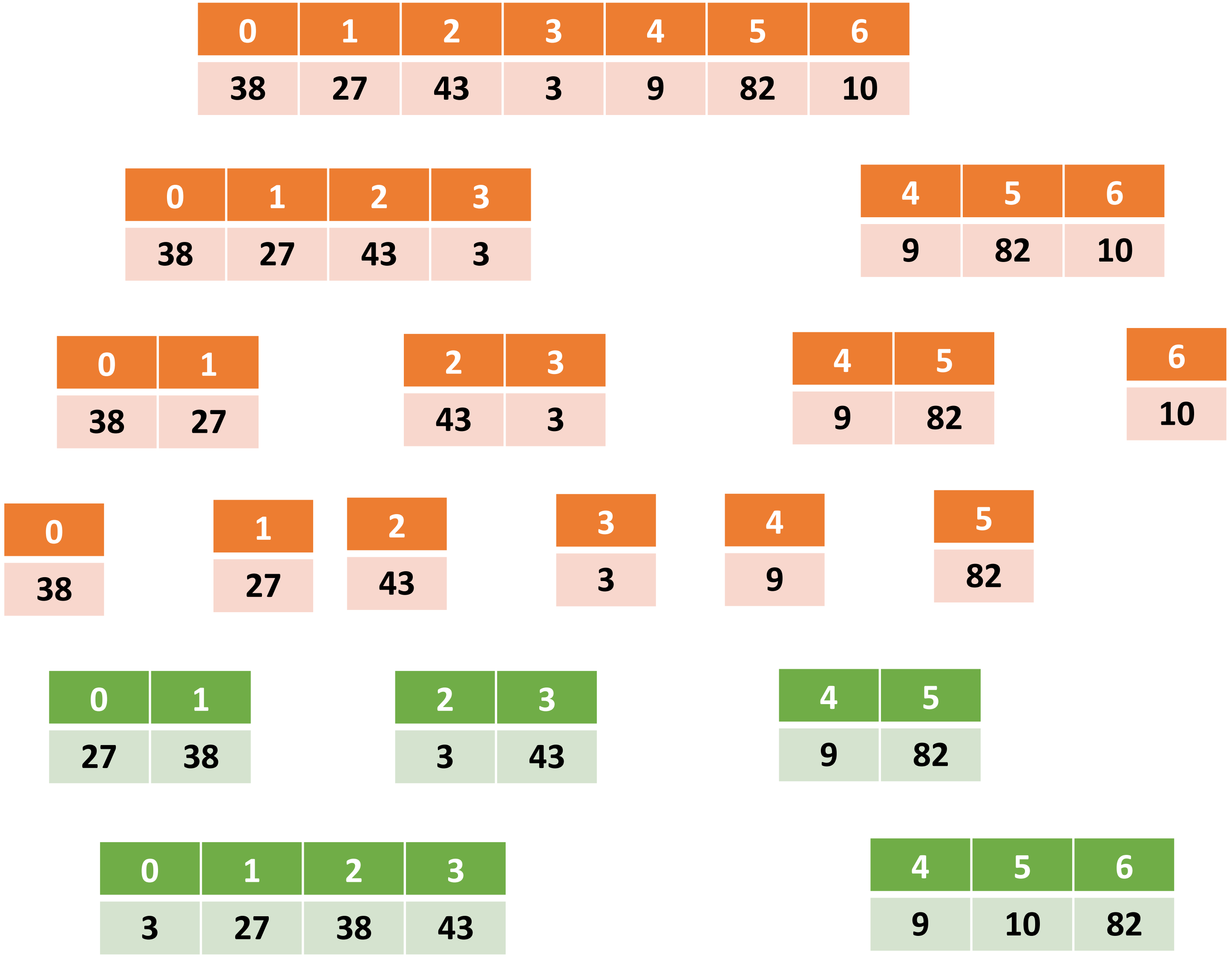

```
mergeSort (A, 0, 6):

mid = (0 + 6) / 2 = 3
mergeSort (A, 0, 3):

mid = (0 + 3) / 2 = 1
mergeSort (A, 0, 1)
mergeSort (A, 2, 3):

mid = (2 + 3) / 2 = 2
mergeSort (A, 2, 2):
  terminates!
mergeSort (A, 3, 3):
  terminates!

merge (A, 2, 2, 3)
```



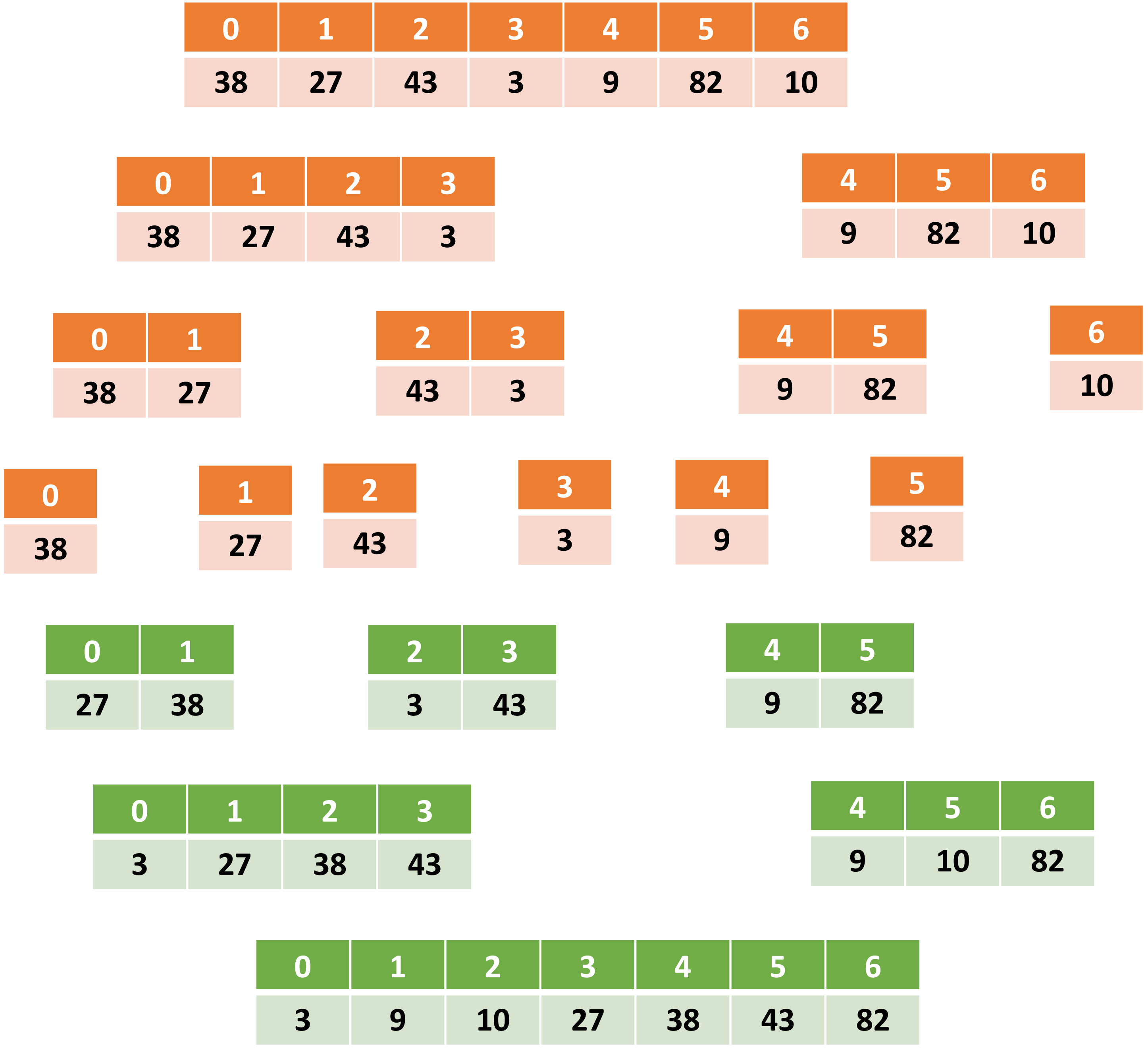

```
mergeSort (A, 0, 6):

mid = (0 + 6) / 2 = 3
mergeSort (A, 0, 3):

    mid = (0 + 3) / 2 = 1
    mergeSort (A, 0, 1)
    mergeSort (A, 2, 3)
    merge (A, 0, 1, 3)
```




```
mergeSort (A, 0, 6):

mid = (0 + 6) / 2 = 3
mergeSort (A, 0, 3)
mergeSort (A, 4, 6):

mid = (4 + 6) / 2 = 5
mergeSort (A, 4, 5):

mid = (4 + 5) / 2 = 4
mergeSort (A, 4, 4):
  terminates!

mergeSort (A, 5, 5):
  terminates!
merge (A, 4, 4, 5)
```

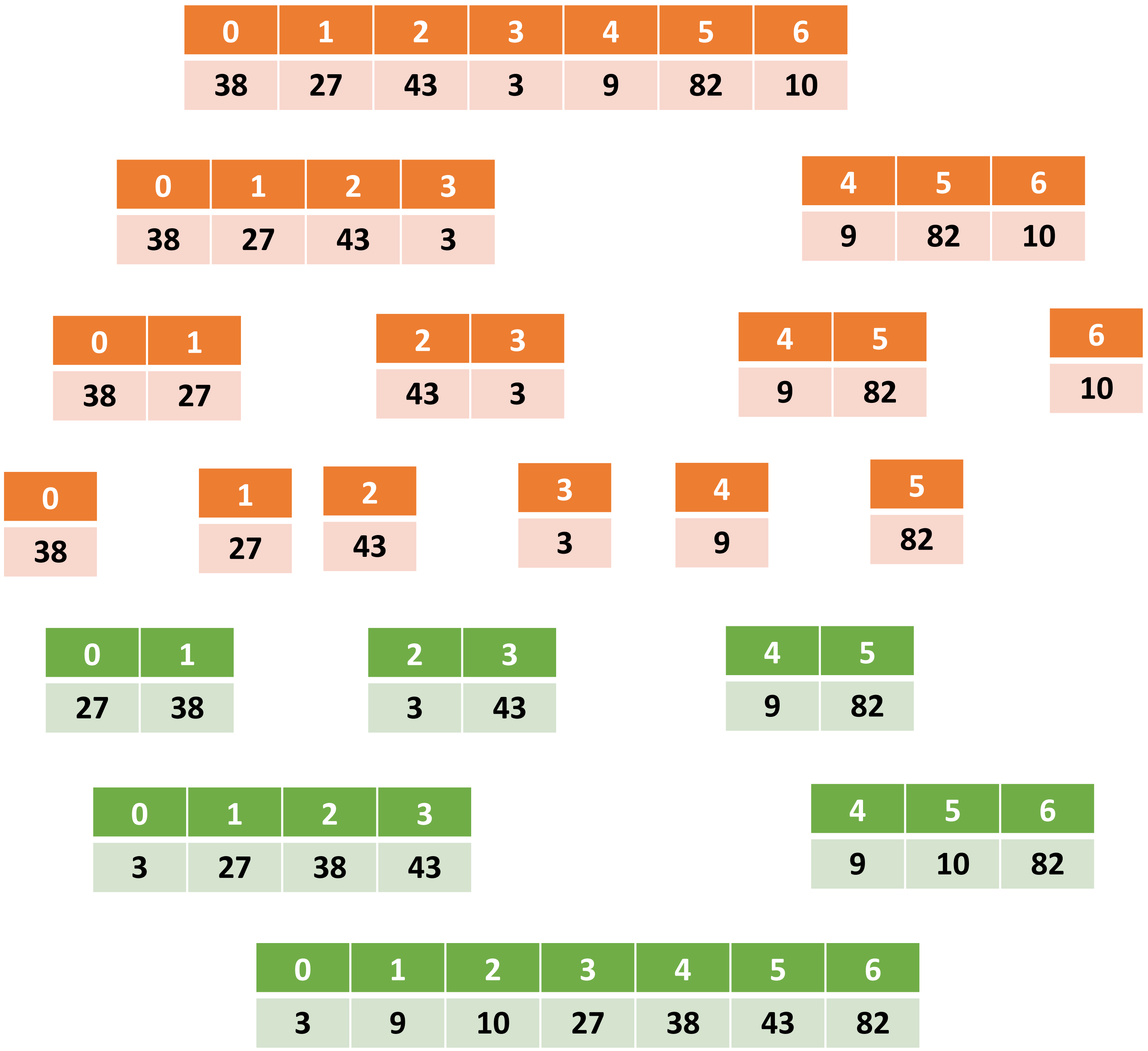



```
mergeSort (A, 0, 6):

mid = (0 + 6) / 2 = 3
mergeSort (A, 0, 3)
mergeSort (A, 4, 6):

mid = (4 + 6) / 2 = 5
mergeSort (A, 4, 5)
mergeSort (A, 6, 6):
   terminates!
merge (A, 4, 5, 6)
```




```
mergeSort (A, 0, 6):

    mid = (0 + 6) / 2 = 3
    mergeSort (A, 0, 3)
    mergeSort (A, 4, 6)
    merge (A, 0, 3, 6)
```

`mergeSort (A, 0, 6)`



Quicksort Algorithm

- Suppose the portion of the array A between p and r needs to be sorted:

p						r
30	65	10	75	55	90	60

- Find an index q and reorganize elements such that:
  - All elements to the left of q are smaller than $A[q]$
  - All elements to the right of q are greater than $A[q]$

Achieved by a *partition* algorithm

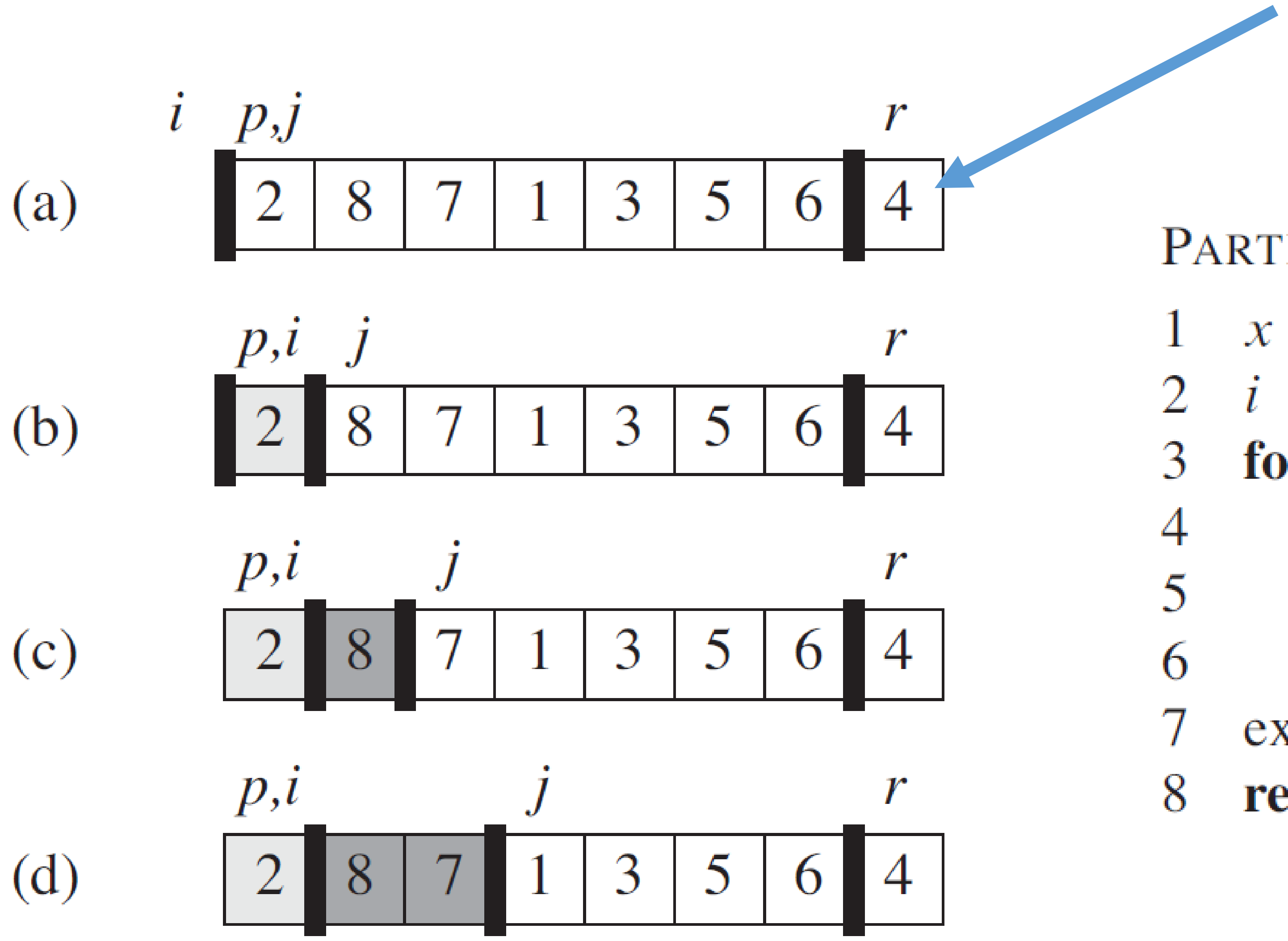
p			q			r
30	10	55	60	65	90	75

Quicksort Function

```
void quickSort (int A[], int p, int r) {
   int q;

   if (p < r) {
       q = partition (A, p, r);
       quickSort (A, p, q-1);
       quickSort (A, q+1, r);
   }

}
```

Quicksort: Partition

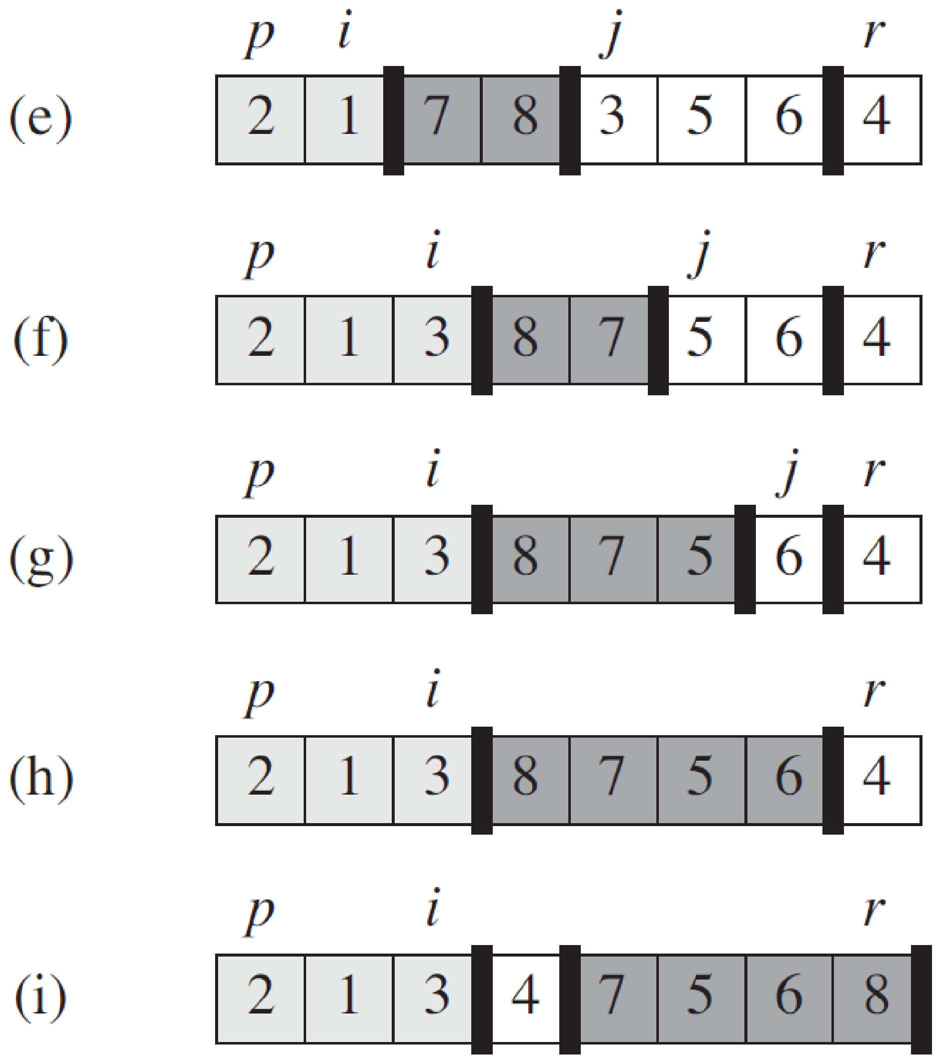
Pivot (x = A[r])



PARTITION(A, p, r)

```
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort: Partition




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

Quicksort: Partition Example

➢ What is the effect of partition (A, p, r) on the following array, *A*?

i	*p j*						*r*
	30	65	10	75	55	90	60

Pass #1

- Increment *i* since 30 ≤ 60:

p i j						*r*
30	65	10	75	55	90	60

- Exchange A[*i*] with A[*j*]:

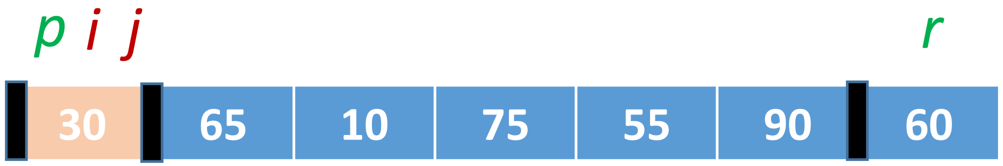
p i j						*r*
30	65	10	75	55	90	60

PARTITION(A, p, r)

```
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

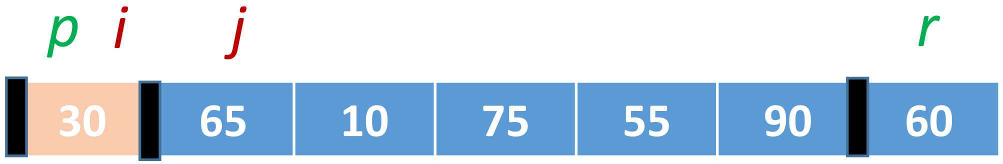
Quicksort: Partition Example

- From Pass #1:



Pass #2

- No change to *i*, since 65 > 60

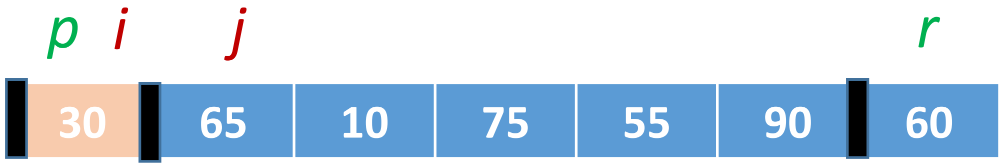


PARTITION(A, p, r)

$x = A[r]$
$i = p - 1$
for $j = p$ **to** $r - 1$
  if $A[j] \le x$
    $i = i + 1$
    exchange $A[i]$ with $A[j]$
exchange $A[i+1]$ with $A[r]$
return $i + 1$

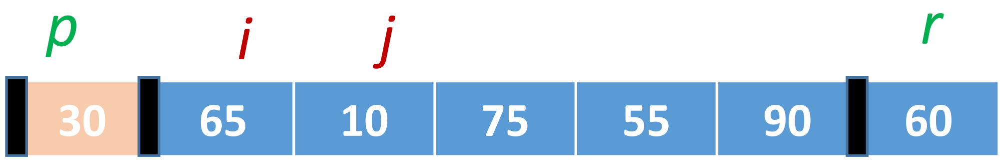
Quicksort: Partition Example

- From Pass #2:

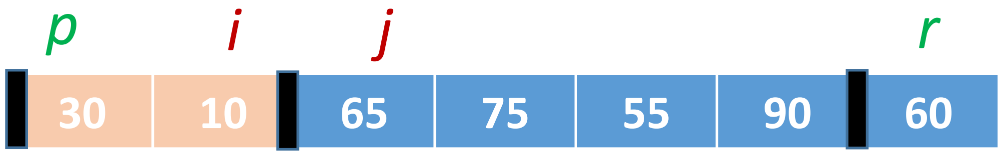


Pass #3

- Increment *i* since 10 ≤ 60:



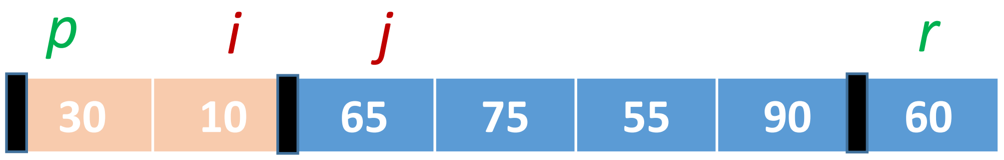
- Exchange A[*i*] with A[*j*]:




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

Quicksort: Partition Example

- From Pass #3:



Pass #4

- No change to *i* since 75 > 60:


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

Quicksort: Partition Example

➢ From Pass #4:

p	*i*		*j*			*r*
30	10	65	75	55	90	60

Pass #5

- Increment *i* since 55 ≤ 60:

p		*i*		*j*		*r*
30	10	65	75	55	90	60

- Exchange A[*i*] with A[*j*]:

p		*i*		*j*		*r*
30	10	55	75	65	90	60

PARTITION(A, p, r)

```
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort: Partition Example

➢ From Pass #5:

Pass #6

- No change to *i* since 90 > 60:

***for* loop terminates**

PARTITION(A, p, r)

```
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort: Partition Example

- From Pass #6:

Exchange A[*i*+1] with A[*r*]:

return *i* + 1

PARTITION(A, p, r)

$x = A[r]$
$i = p - 1$
for $j = p$ **to** $r - 1$
    if $A[j] \leq x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$

Quicksort: Partition Example

- What is the effect of partition (A, p, r) on the following array, *A*?

p						*r*
30	65	10	75	55	90	60

p			*q*			*r*
30	10	55	60	65	90	75

PARTITION(A, p, r)

```
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Topological Sorting

- Given n items, numbered 1 to n, and m requirements of the form $j \rightarrow k$, meaning that item j must come before item k, arrange the items in an order such that all the requirements are satisfied or determine that no solution is possible.

- For example, suppose that $n = 9$ and $m = 10$ with the following requirements:

$$3 \rightarrow 7 \quad 4 \rightarrow 2 \quad 8 \rightarrow 6 \quad 9 \rightarrow 5 \quad 1 \rightarrow 2$$
$$6 \rightarrow 5 \quad 2 \rightarrow 5 \quad 7 \rightarrow 8 \quad 8 \rightarrow 1 \quad 1 \rightarrow 9$$

- Two of the many solutions are:

4 3 7 8 6 1 9 2 5
3 7 8 6 1 9 4 2 5

Topological sorting can be implemented using a *graph*.

Topological Sorting: Example

- Five morocoys, *A*, *B*, *C*, *D*, and *E* ran a race.
- *A* finished before *B*, but behind *C*. *D* finished before *E*, but behind *B*. What was the finishing order?
- C A B D E

Sorting Experiments

Observe the difference in performance between the different sorting algorithms