COMP 2611, DATA STRUCTURES
LECTURE 21

SORTING

- **Performance Analysis of Selection Sort, Bubble Sort, and Insertion Sort**
- **Mergesort**
- **Quicksort**

Mergesort Algorithm

- Uses a "divide-and-conquer" approach.

```
Mergesort (A, lengthA) {

  sort the first half of A
  sort the second half of A
  merge the sorted halves

}
```

Mergesort

- Suppose we need to sort the following array:

0	1	2	3	4	5	6	7	8	9
56	34	38	71	28	10	91	18	22	75

First half

Second half

- We sort the first half and the second half separately:

0	1	2	3	4	5	6	7	8	9
28	34	38	56	71	10	18	22	75	91

First half

Second half

Merge

```
void mergeArray (int A[], int p, int q, int r):
```

- This version of *mergeArray* accepts a single array, *A*, as a parameter. The values in *A* from location *p* to location *q* are in sorted order. The values from location *q+1* to location *r* are also in sorted order. However, the values from location *p* to location *r* are not sorted.

To be done in Lab #10

0	1	2	3	4	5	6	7	8	9
28	34	38	56	71	10	18	22	75	91

p First half *q* *q+1* Second half *r*

Mergesort

- Suppose we need to sort the following array:

0	1	2	3	4	5	6	7	8	9
56	34	38	71	28	10	91	18	22	75

First half

Second half

- How to sort the first half and the second half?

0	1	2	3	4
56	34	38	71	28

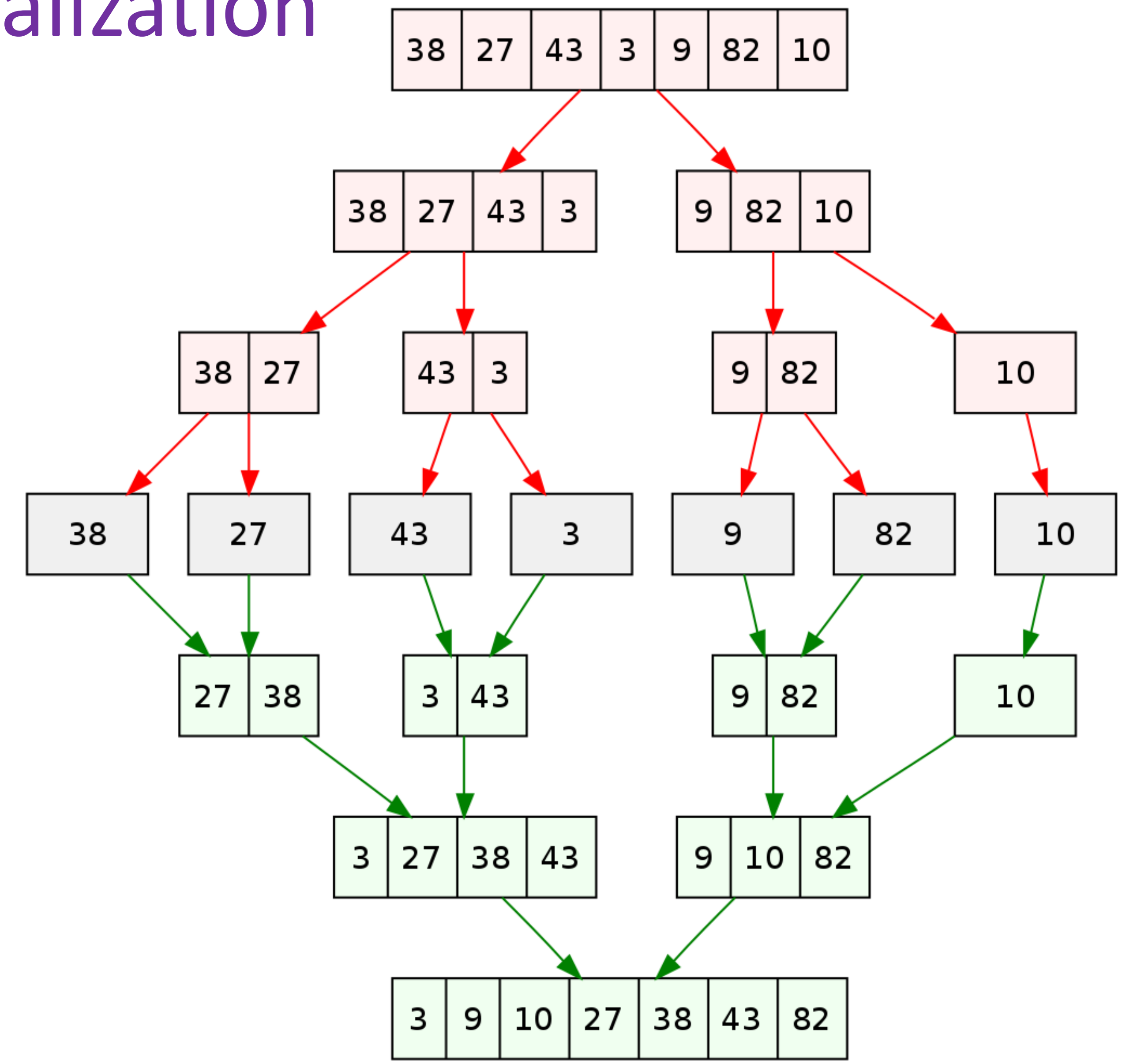
First half

Second half

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

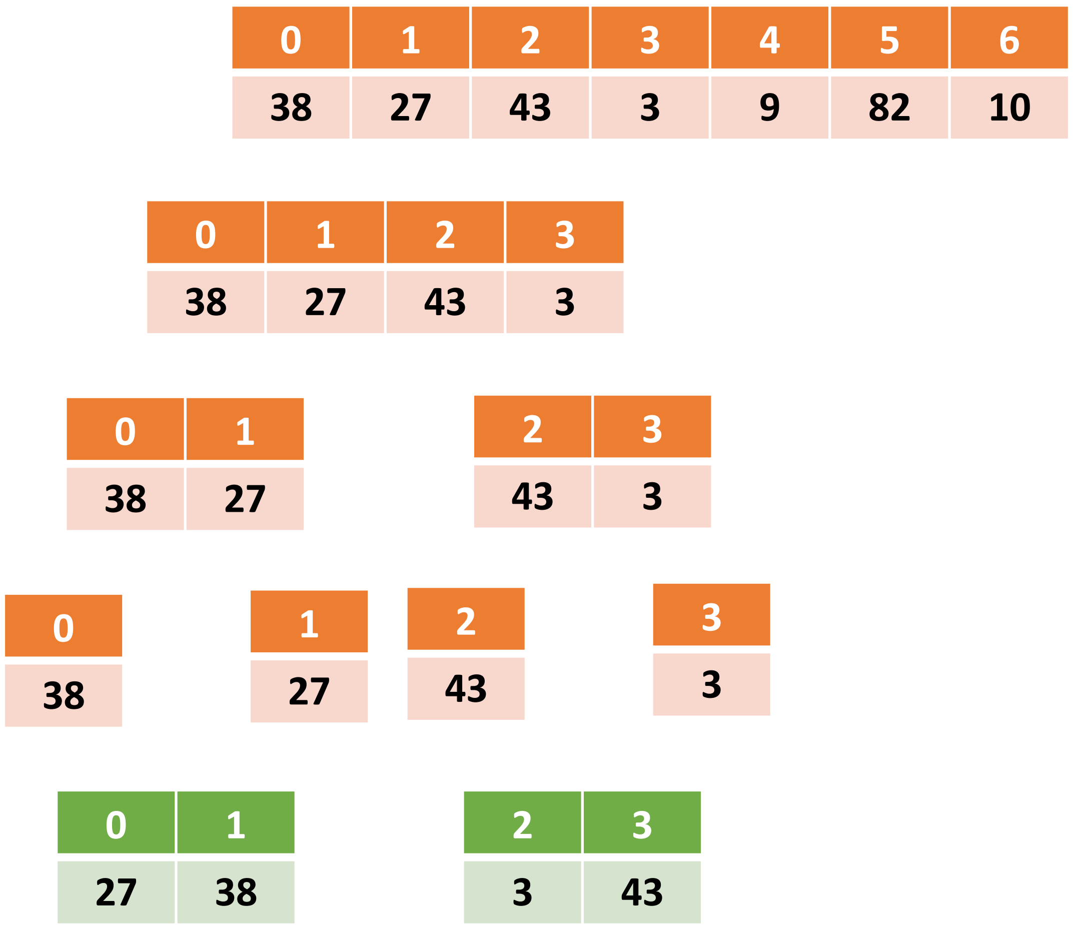
0	1
38	27

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)
```

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

2	3
43	3

0
38

1
27

2
43

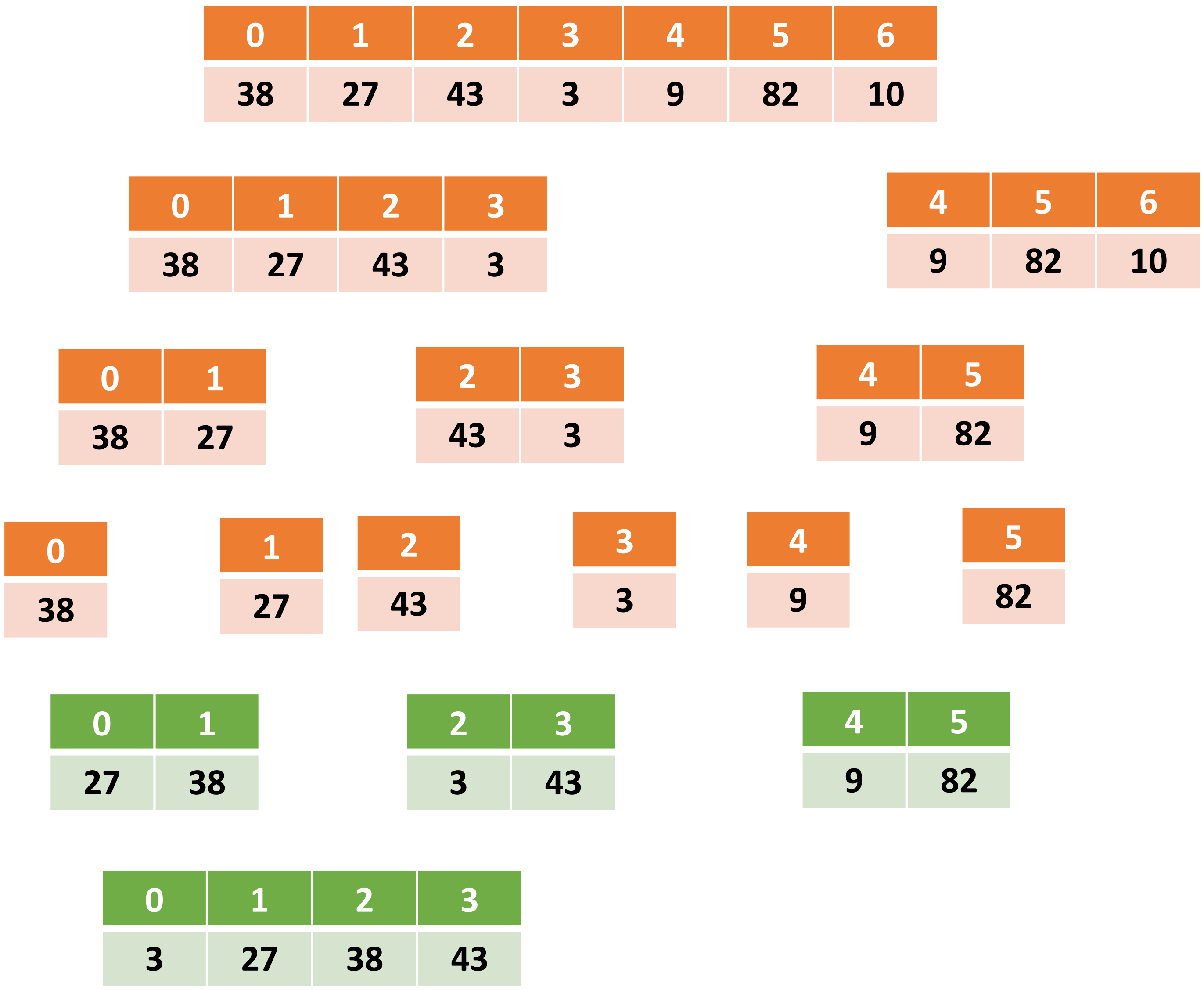
3
3

0	1
27	38

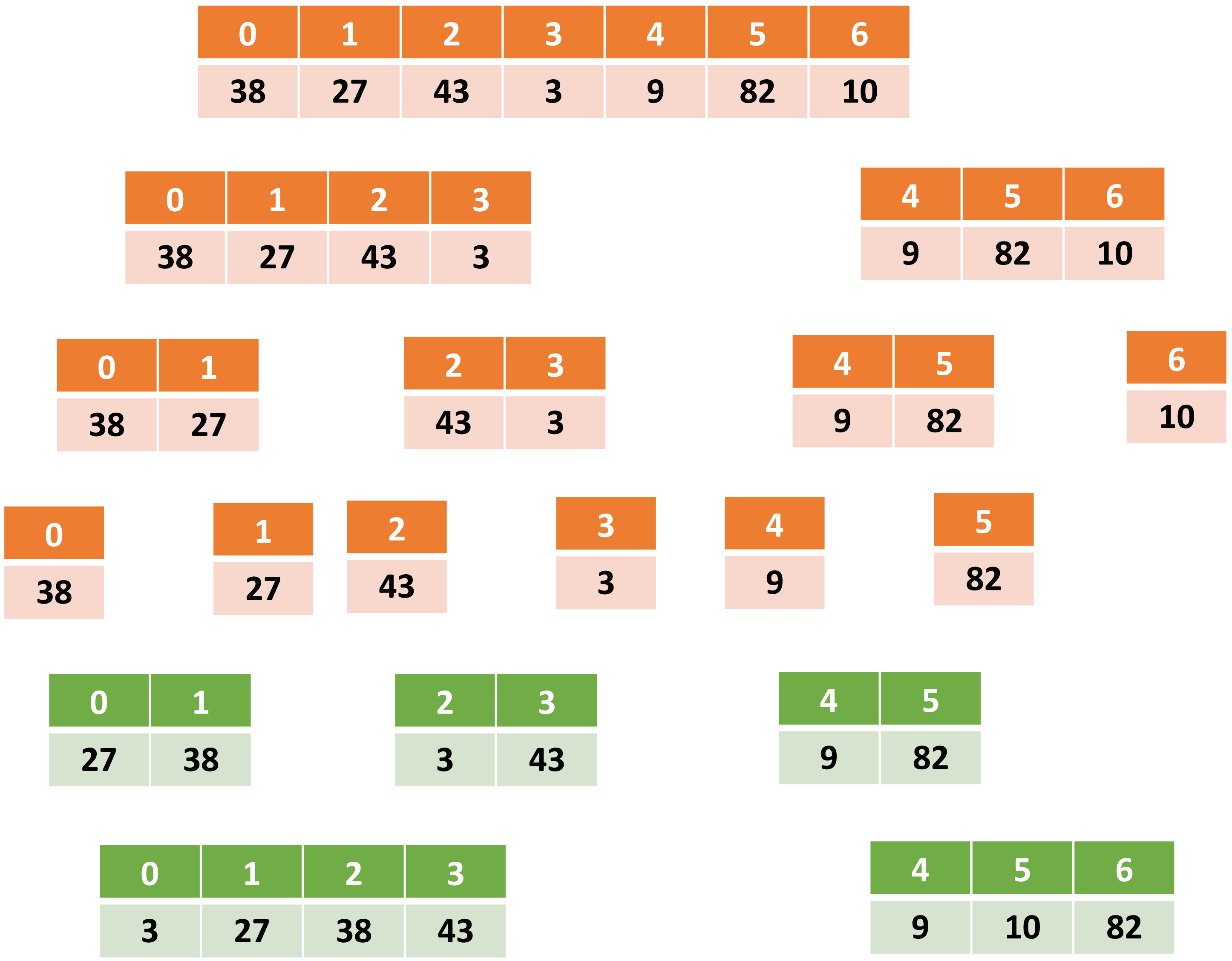
2	3
3	43

0	1	2	3
3	27	38	43

```
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)
```

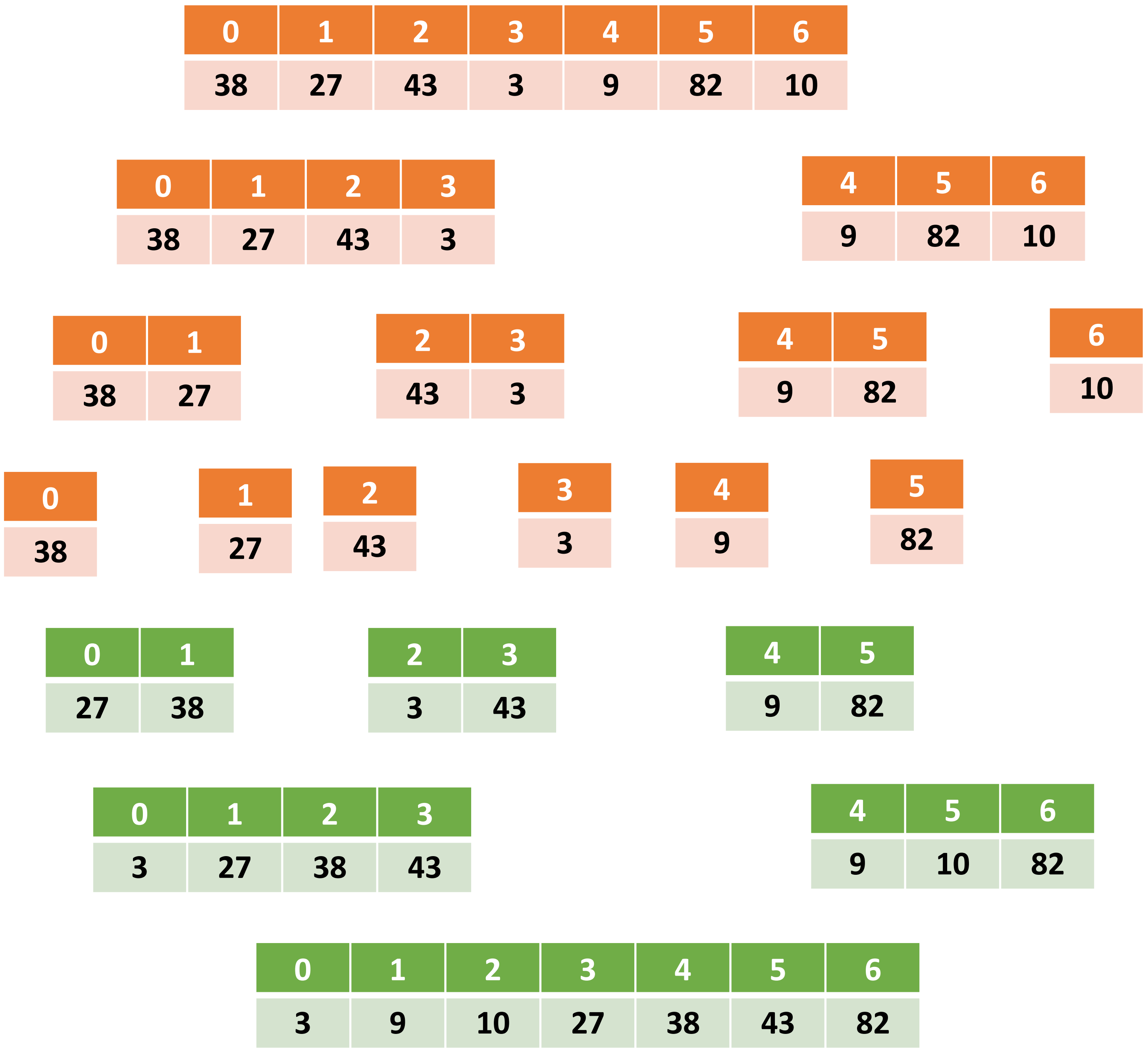



```
mergeSort (A, 0, 6):

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):
```

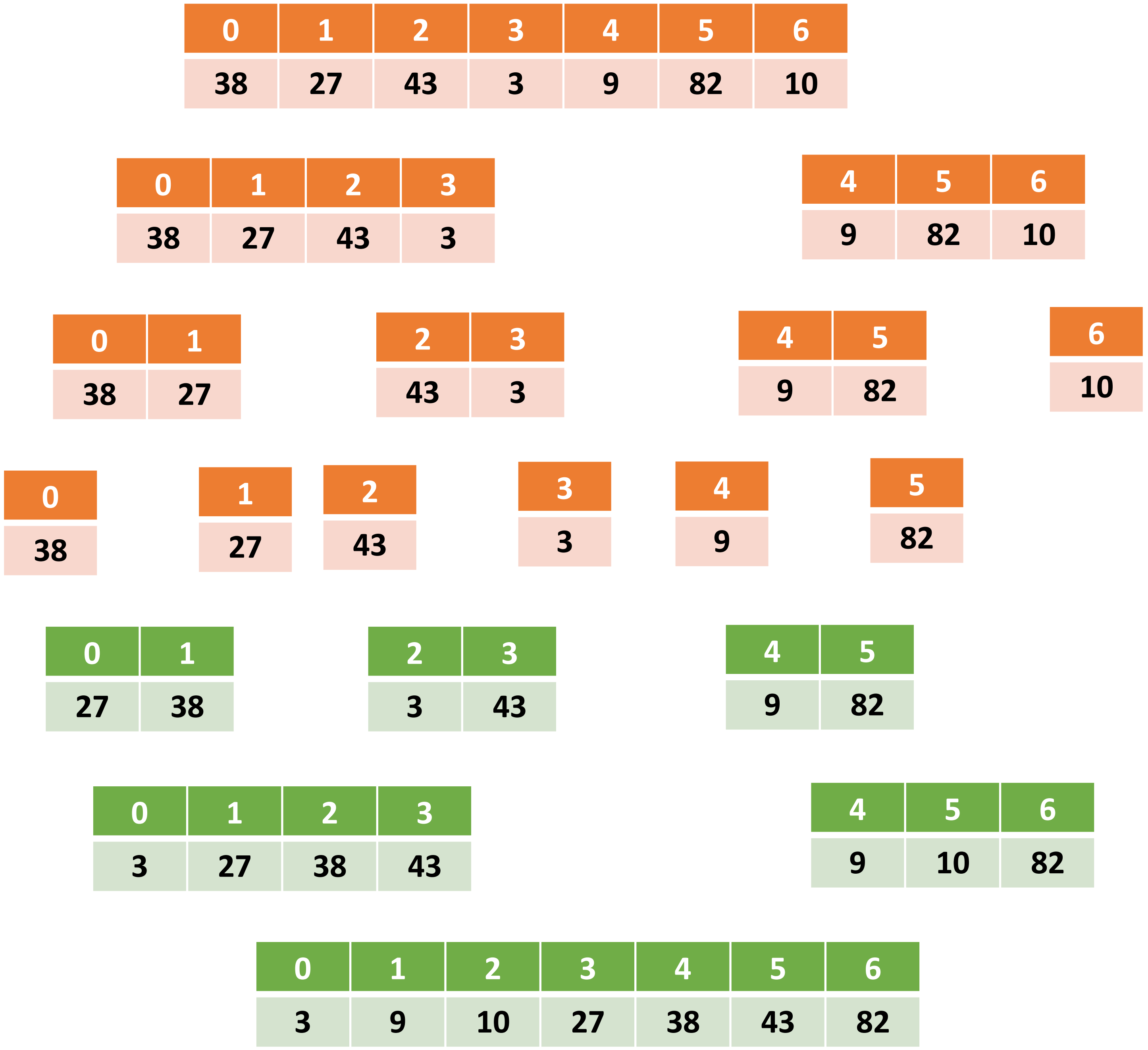
```
mid = (4 + 6) / 2 = 5
mergeSort (A, 4, 5)
mergeSort (A, 6, 6):
   terminates!
merge (A, 4, 5, 6)
```




```
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

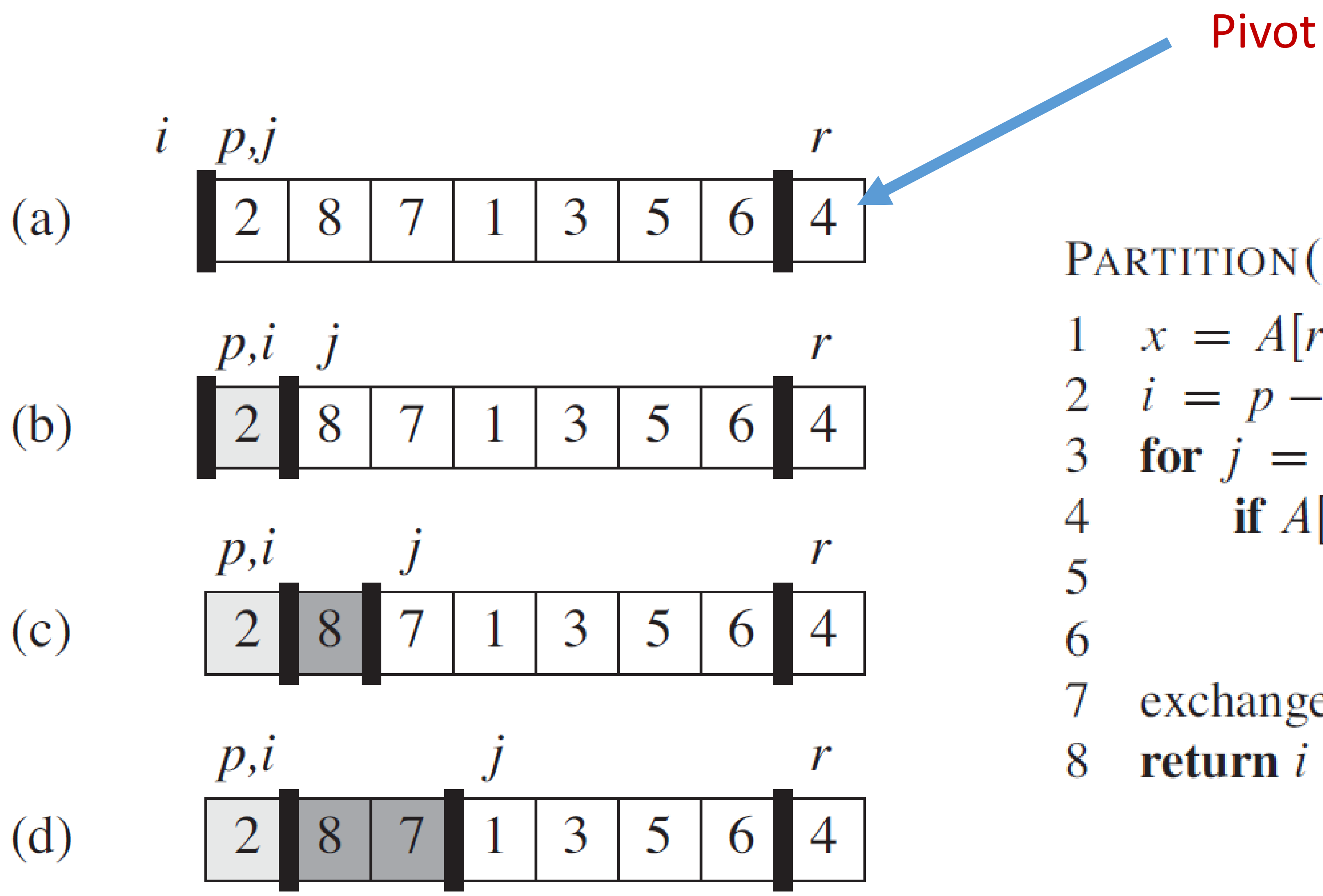
p			q			r
30	10	55	60	65	90	75

Quicksort Function

- Like mergesort, it uses a “divide-and-conquer” approach.

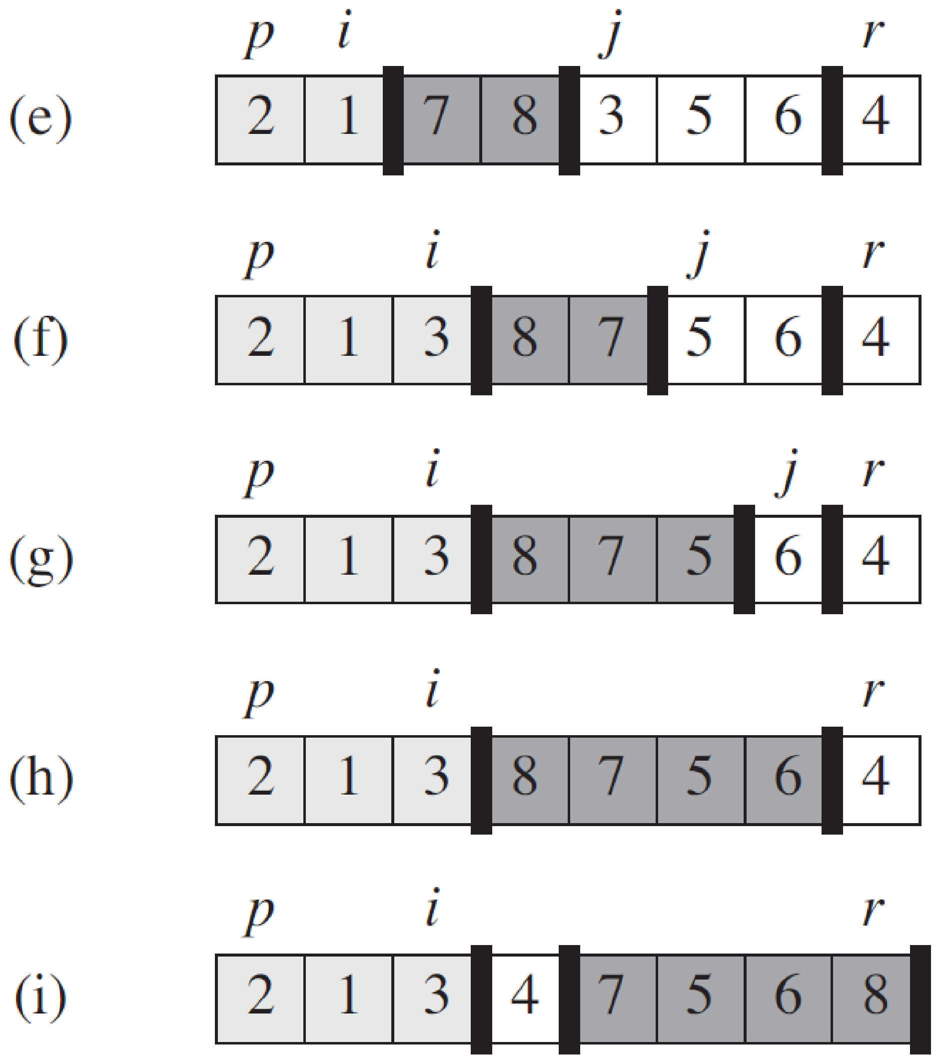
```
void quickSort (int A[], int p, int r) {
   int q;

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

}
```

Quicksort: Partition




```
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 Exercise

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

p						r
30	65	10	75	55	90	60

$\Downarrow$

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
```