DTS203TC
Design and Analysis of Algorithms

Lecture 4: Lab introduction and Quicksort

Dr. Qi Chen, Dr. Siqi Huang, and Dr. Pascal Lefevre
School of AI and Advanced Computing

Pascal.Lefevre@xjtlu.edu.cn

Office hours:

D5012

Wednesdays/Thursdays 14h – 16h

Why does sorting matter?

My answer: sorting can help you get what you want much faster and cheaper

Videos

- https://www.bilibili.com/video/BV1Ws411f7aJ/?spm_id_from=333.337.search-card.all.click
- https://www.bilibili.com/video/BV1ex411e7eb/?spm_id_from=333.337.search-card.all.click

Learning outcomes

- Jupyter Notebook
- Insertion Sort
- Merge Sort
- Quicksort

Jupyter notebook

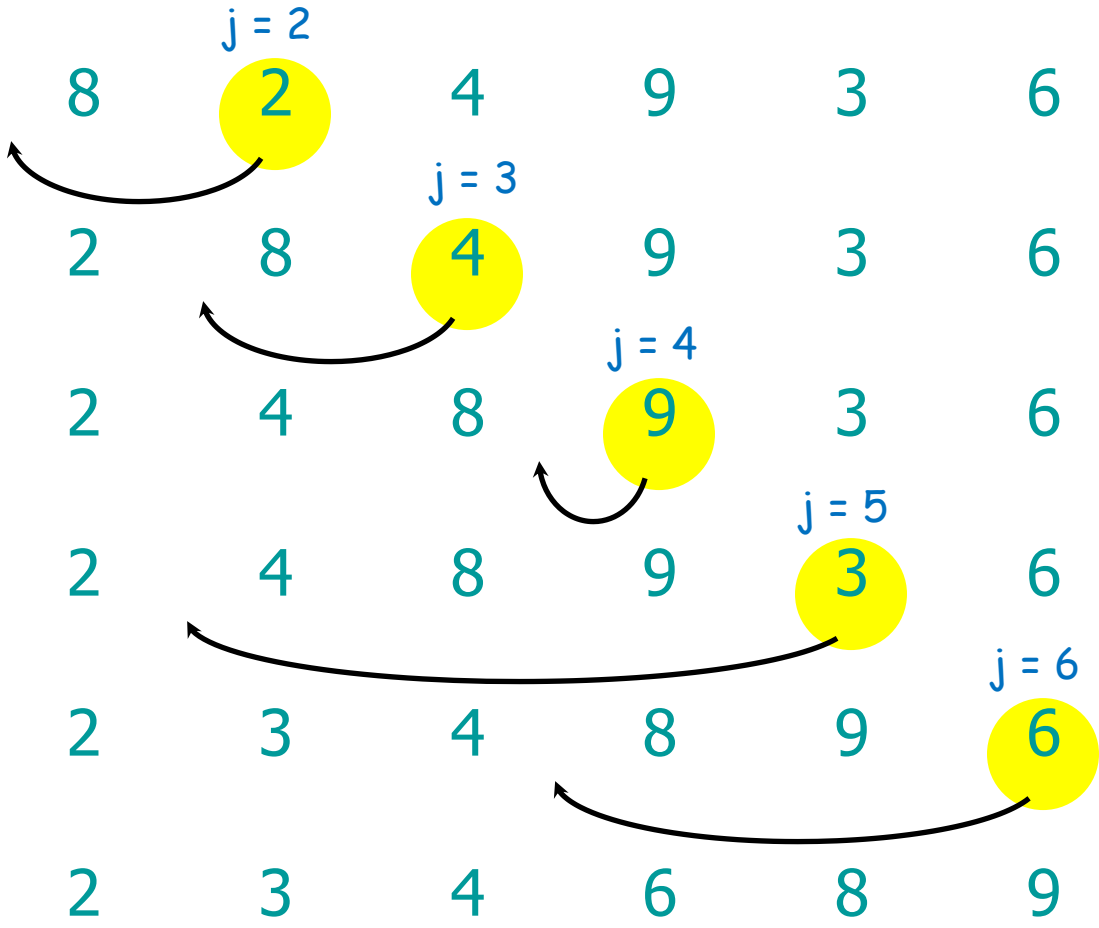
- We assume you can do Python programming
- Jupyter notebook is very convenient for demonstrating and managing your coursework and projects
- Installation and practice

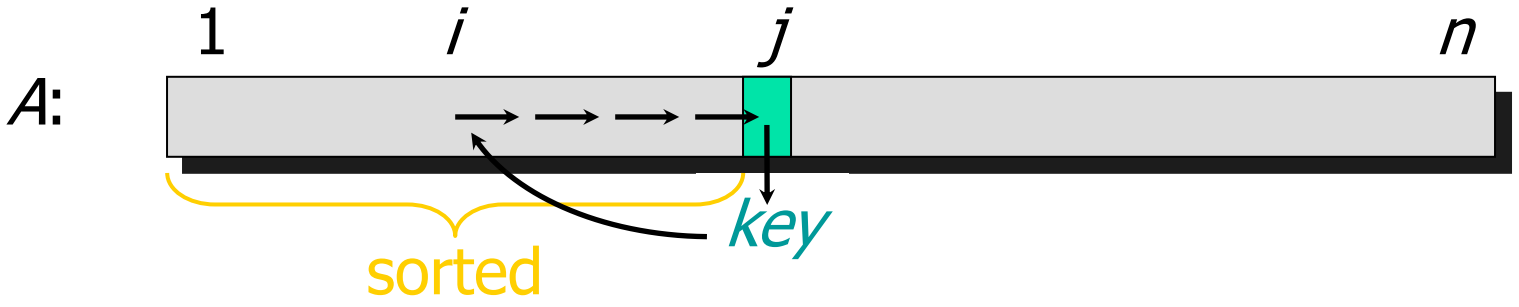
Insertion Sort

Insertion sort

- Pseudocode

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

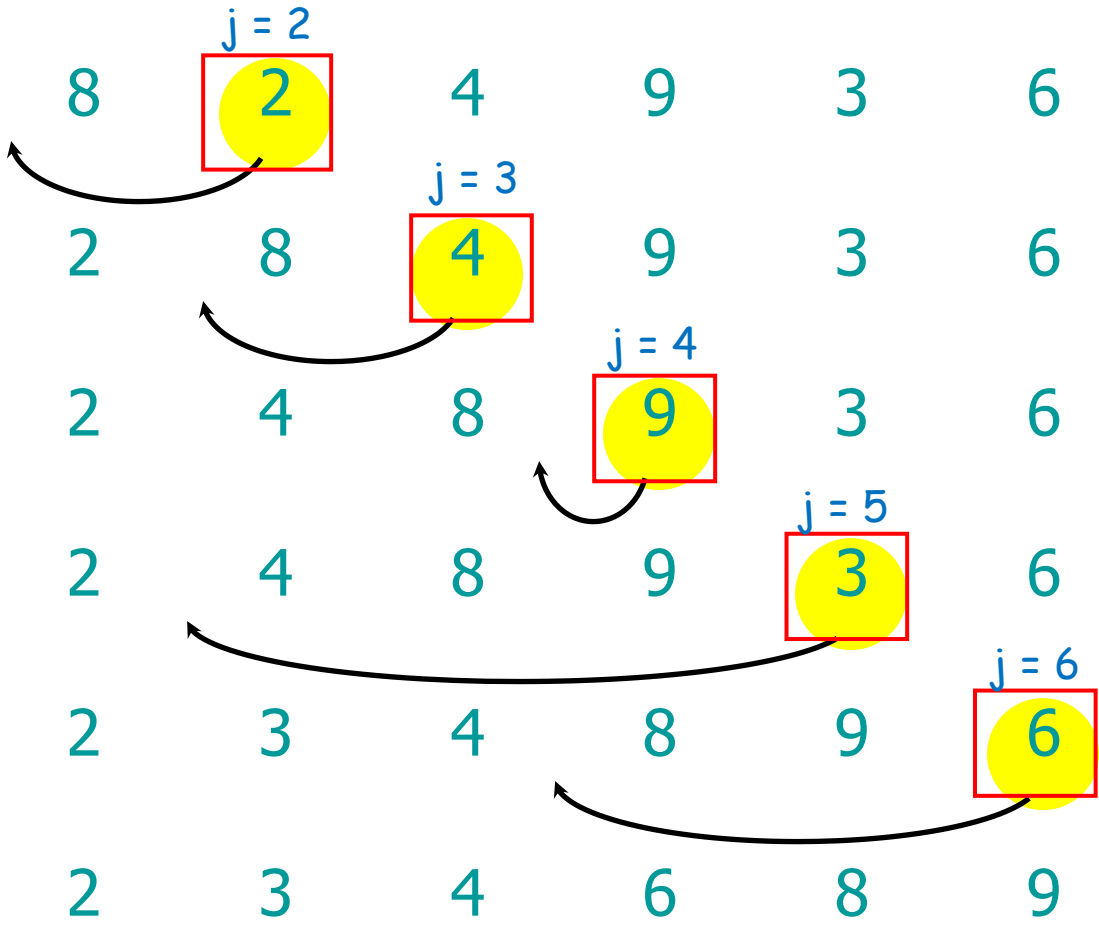


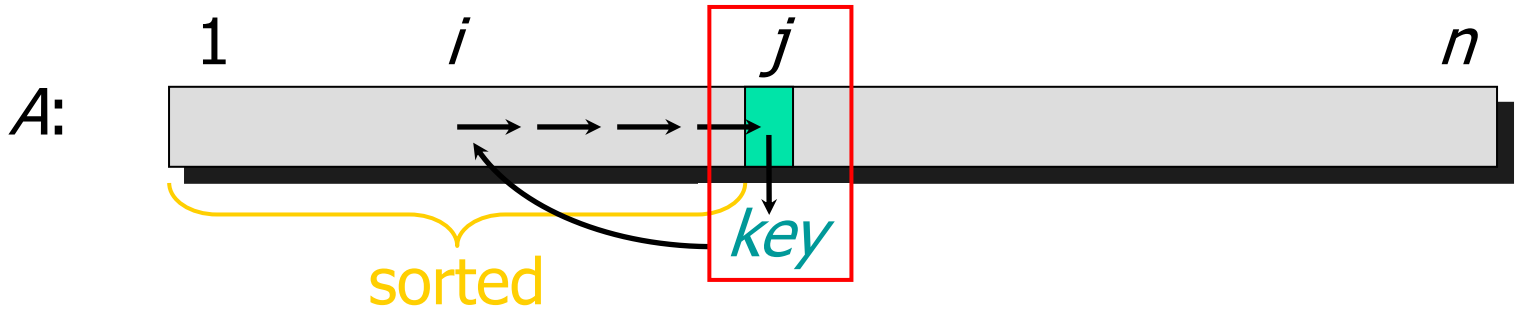


Insertion sort

- Pseudocode

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

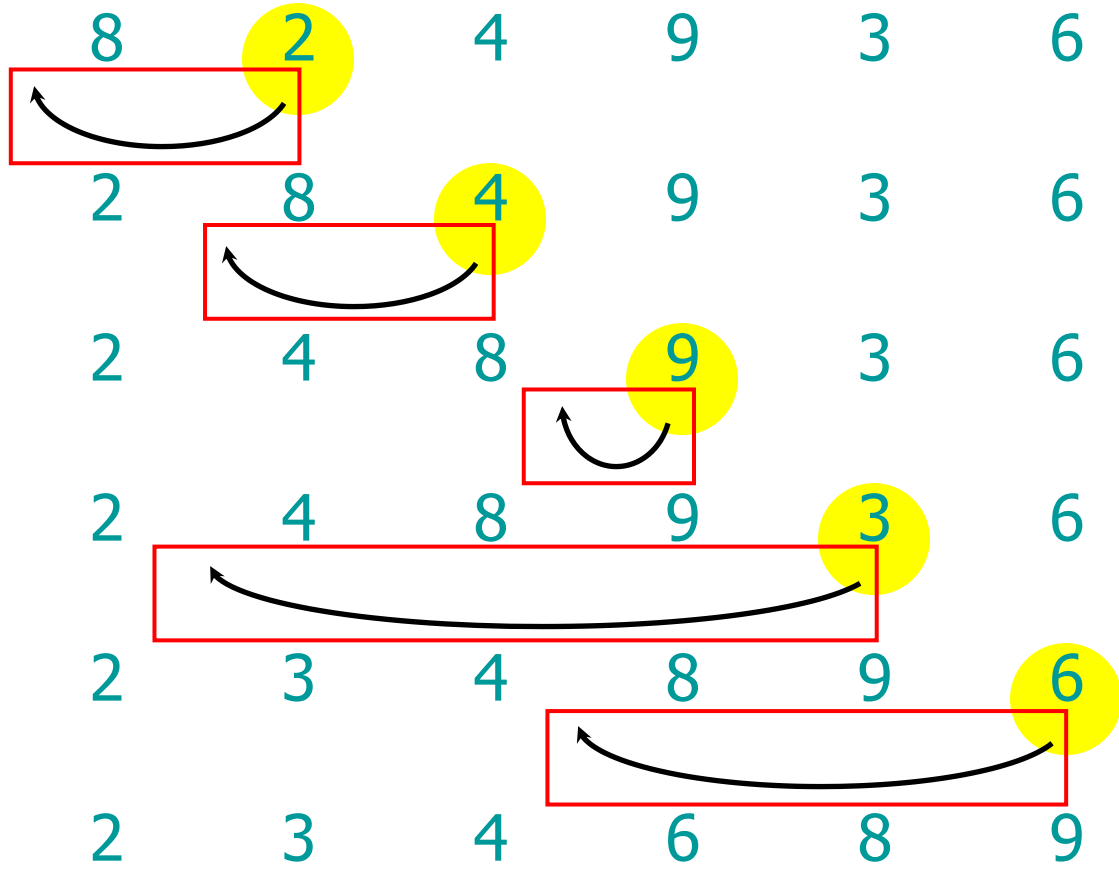


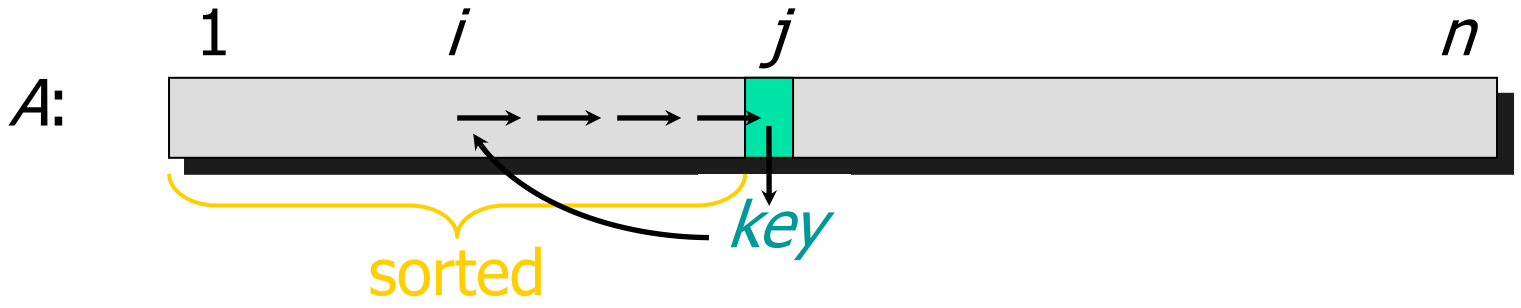


Insertion sort

- Pseudocode

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```





INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

```python
def insertion_sort(InputList):
    for j in range(1, len(InputList)):
        key = InputList[j]
        i = j-1
        # Compare the current element with next one
        while (InputList[i] > key) and (i >= 0):
            InputList[i+1] = InputList[i]
            i=i-1
        InputList[i+1] = key

arr = [19,2,31,45,30,11,121,27]

insertion_sort(arr)

print(arr)
```

```
[2, 11, 19, 27, 30, 31, 45, 121]
```

Merge Sort

Two steps

1. **Divide**
   - dividing a sequence of n numbers into **two** smaller sequences

2. **Conquer**
   - merging two sorted sequences of **total length n**

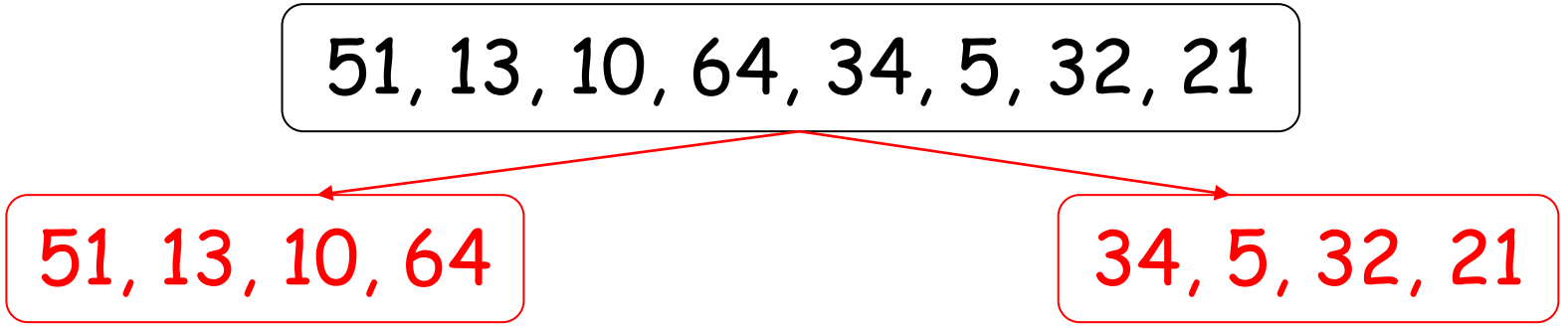
First step

Divide

- dividing a sequence of n numbers into **two** smaller sequences is straightforward

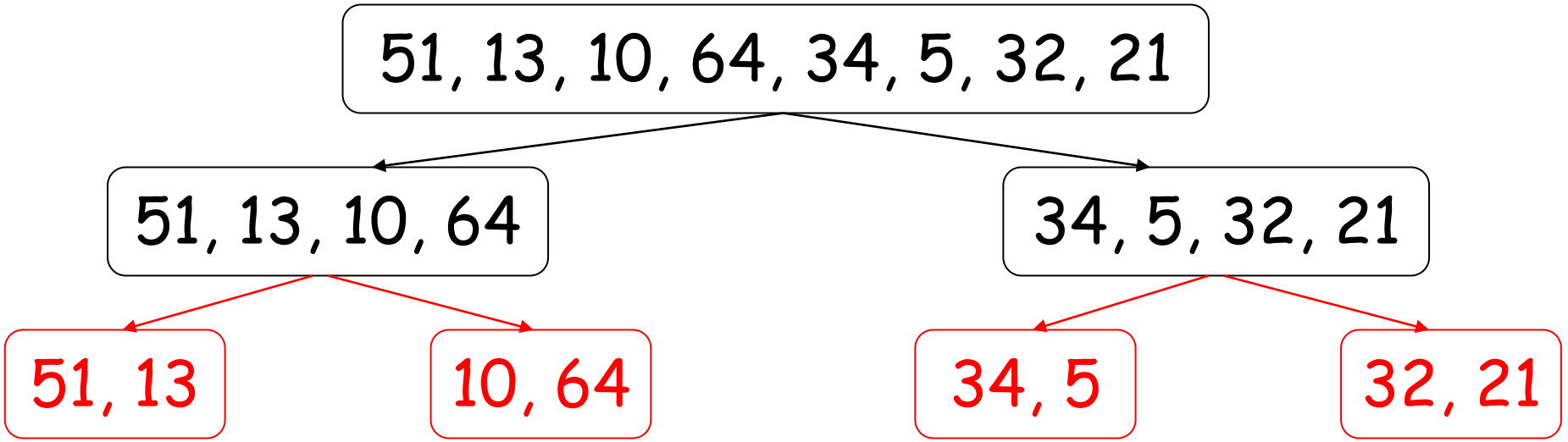
51, 13, 10, 64, 34, 5, 32, 21

we want to sort these 8 numbers,
divide them into two halves

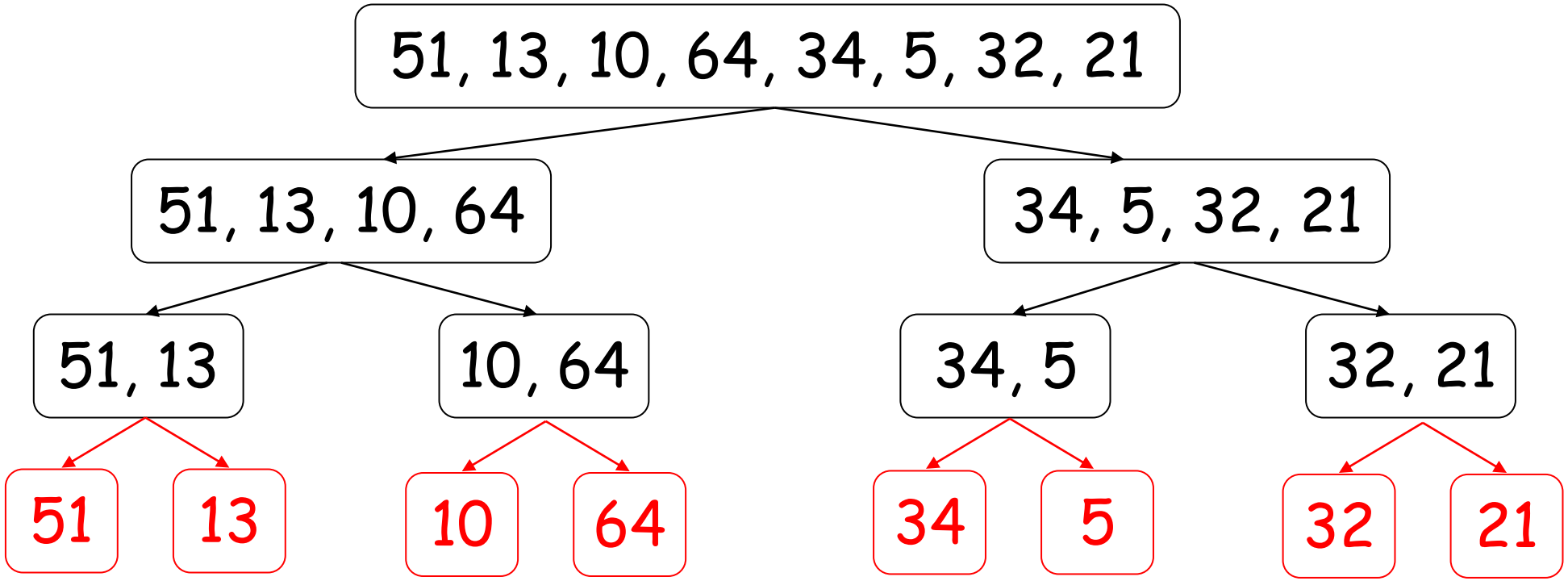


divide these 4 numbers into halves

similarly for these 4



further halve each sequence …
until we get sequence with only 1 number

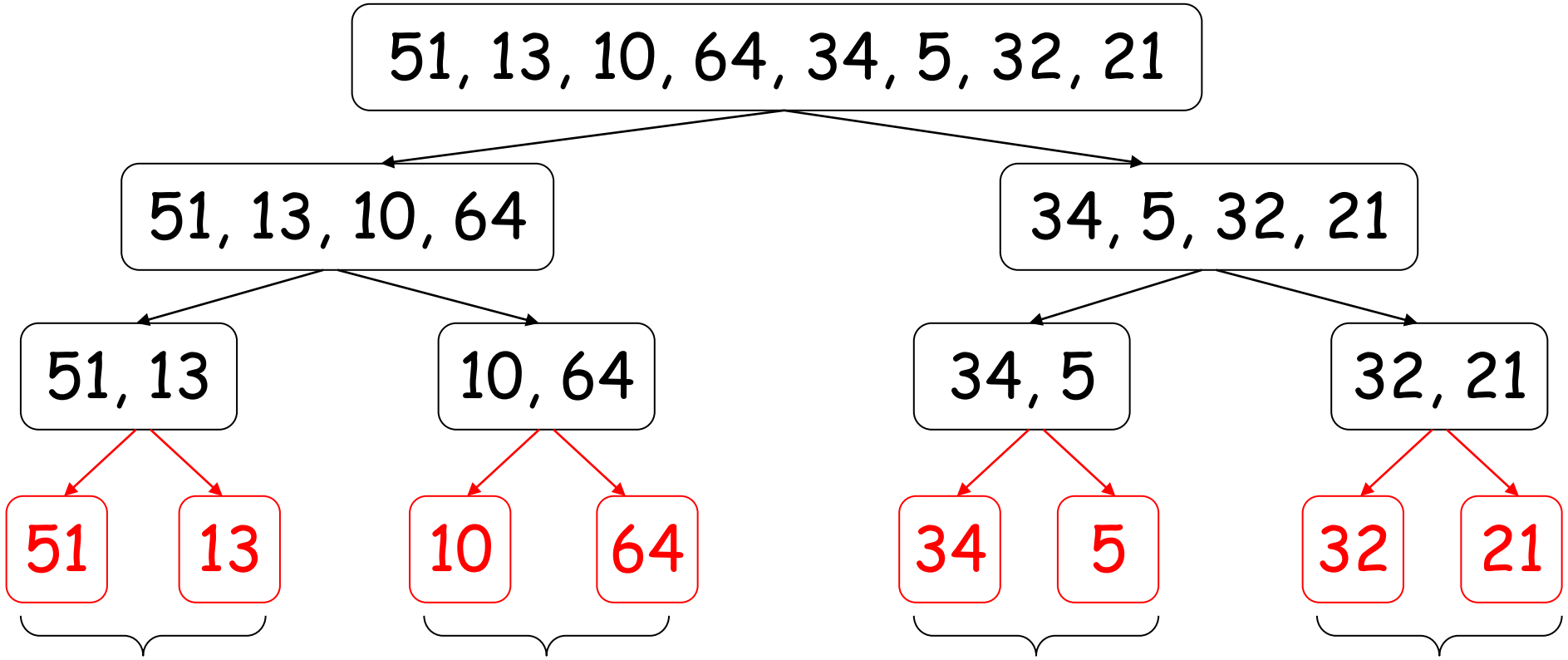


further divide each shorter sequence …
until we get sequence with only 1 number

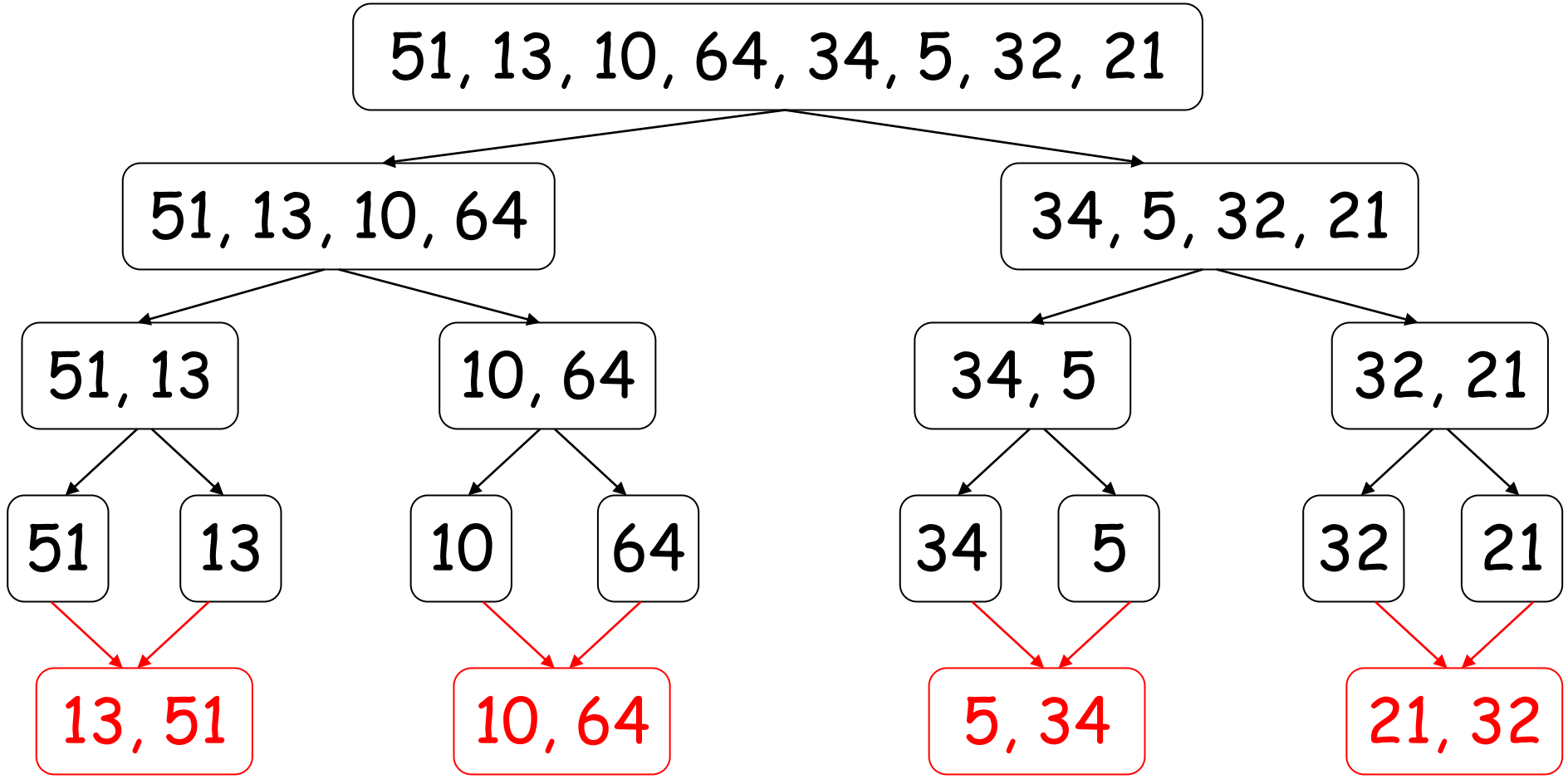
Second step

Conquer

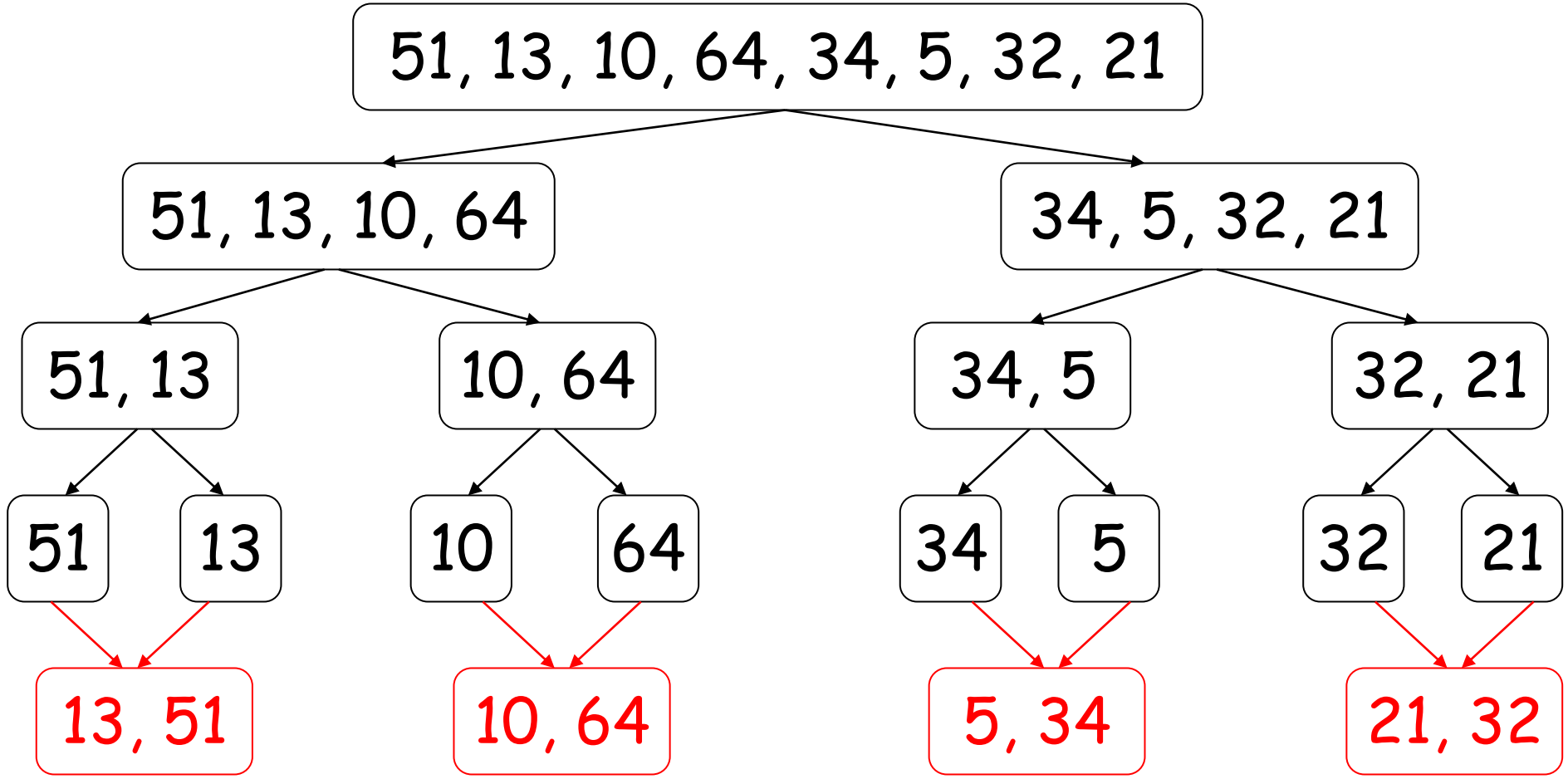
- merge two sorted sequences of **total length n** can also be done easily, at most **n-1** comparisons



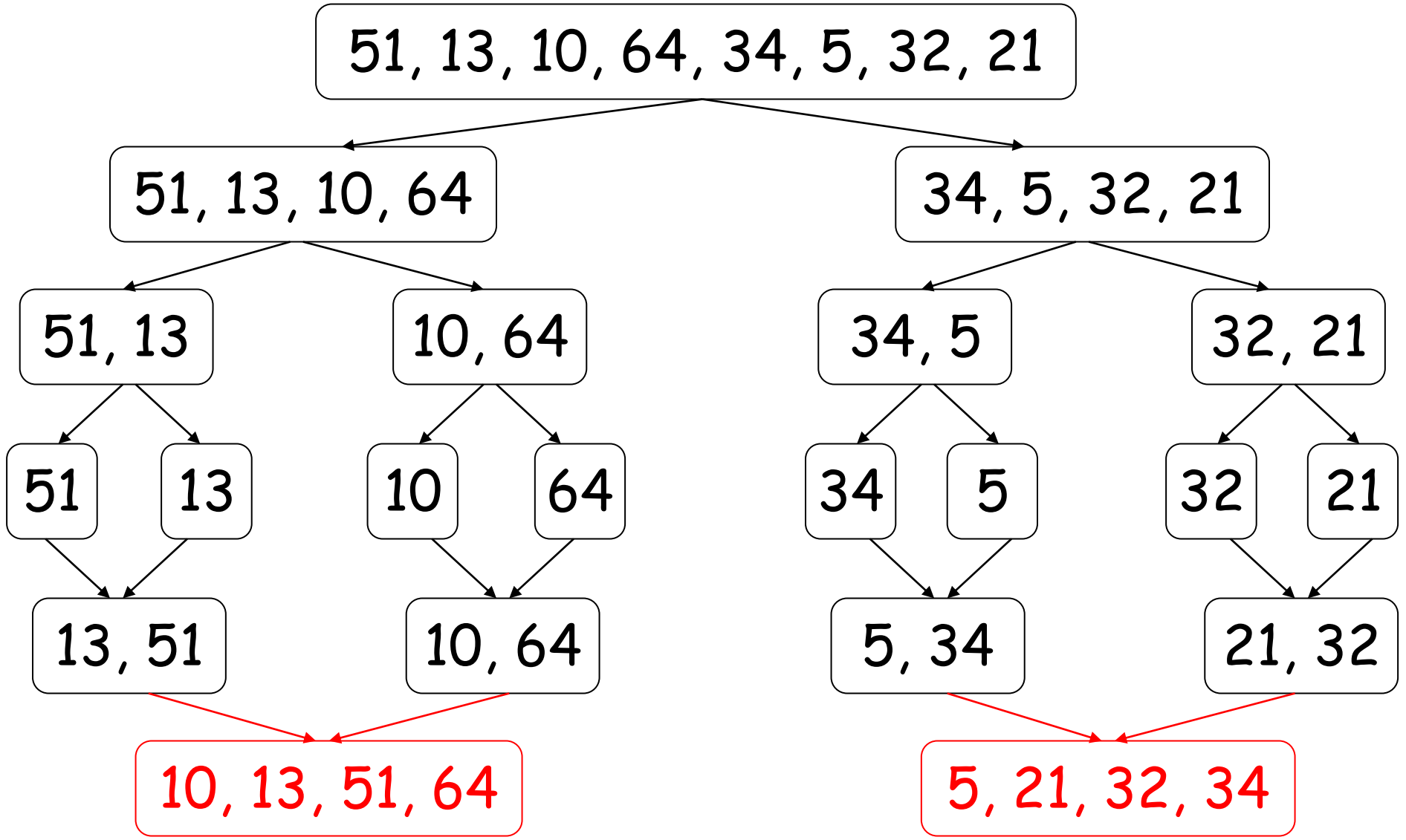
merge pairs of single number into a sequence of 2 sorted numbers



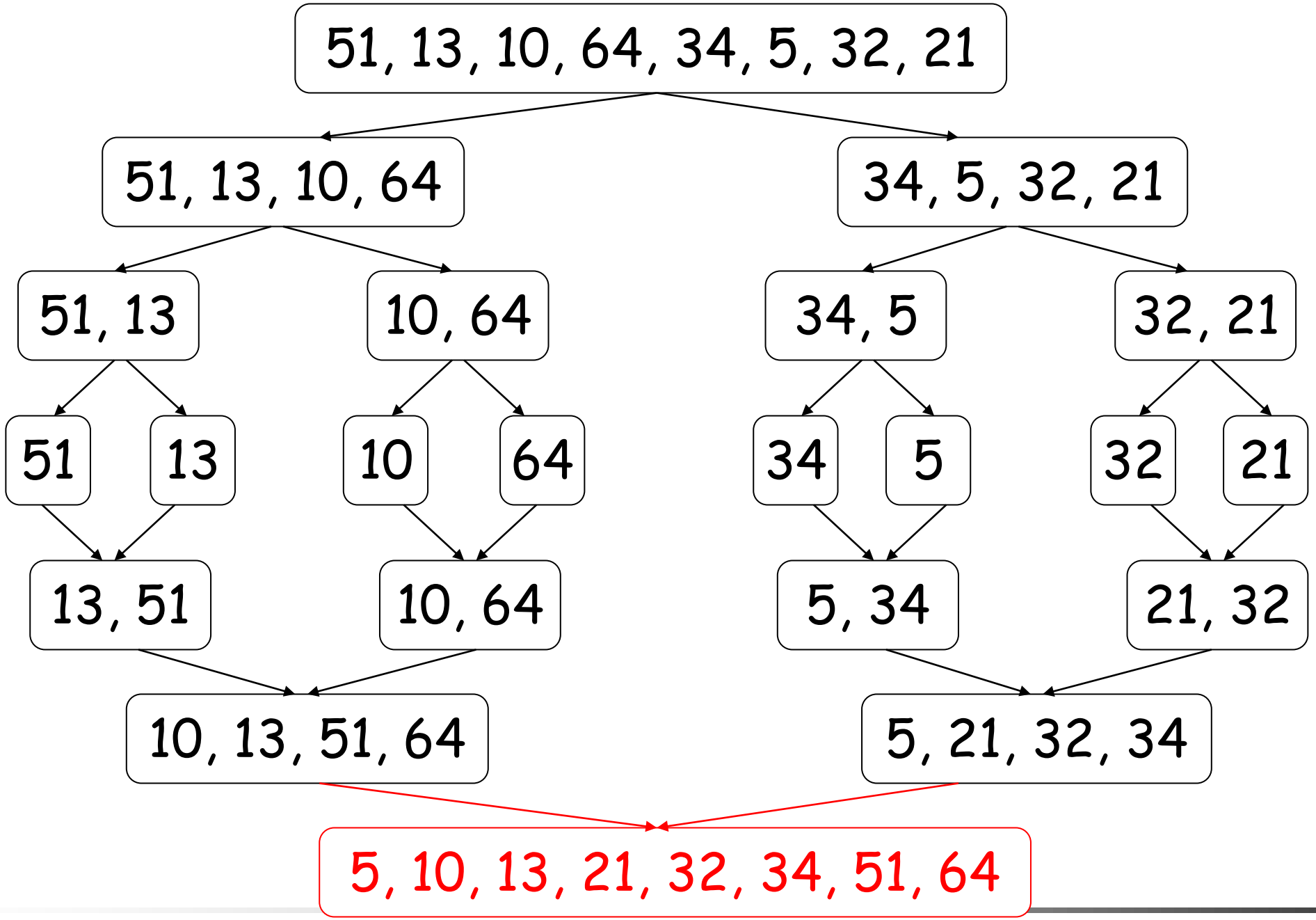
merge pairs of single number into a sequence of 2 sorted numbers



then merge again into sequences of
4 sorted numbers

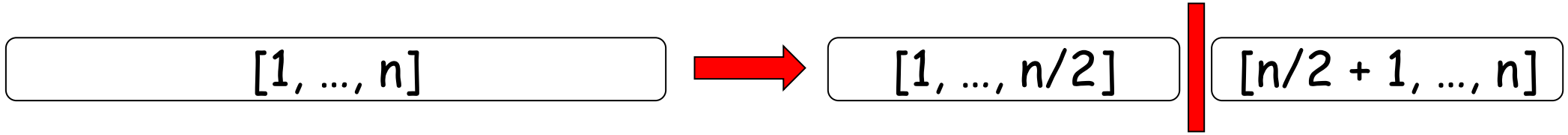


one more merge give the final sorted sequence

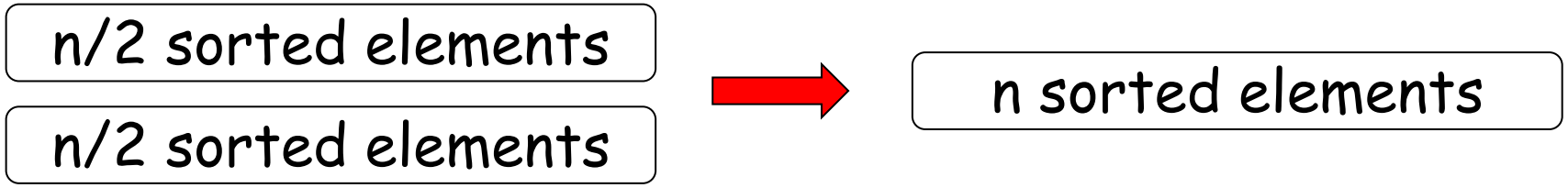


Summary

- **Divide**
  - divide a sequence of n numbers into **two** smaller sequences

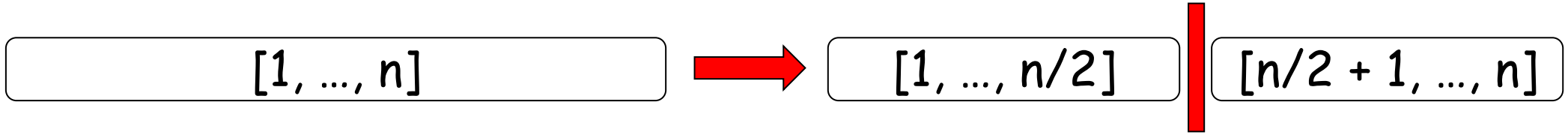


- **Conquer**

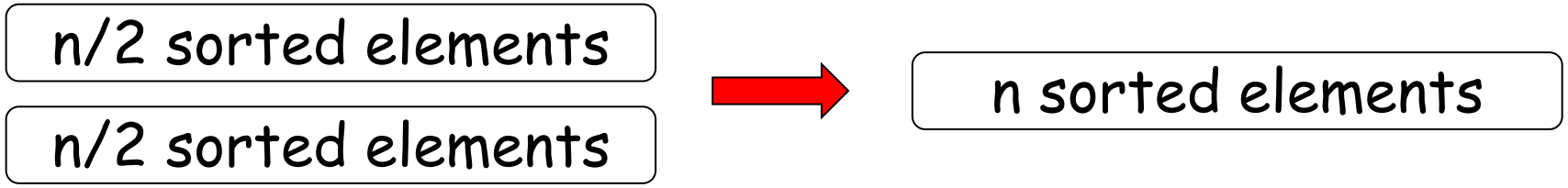


Summary

- **Divide**
  - divide a sequence of n numbers into **two** smaller sequences

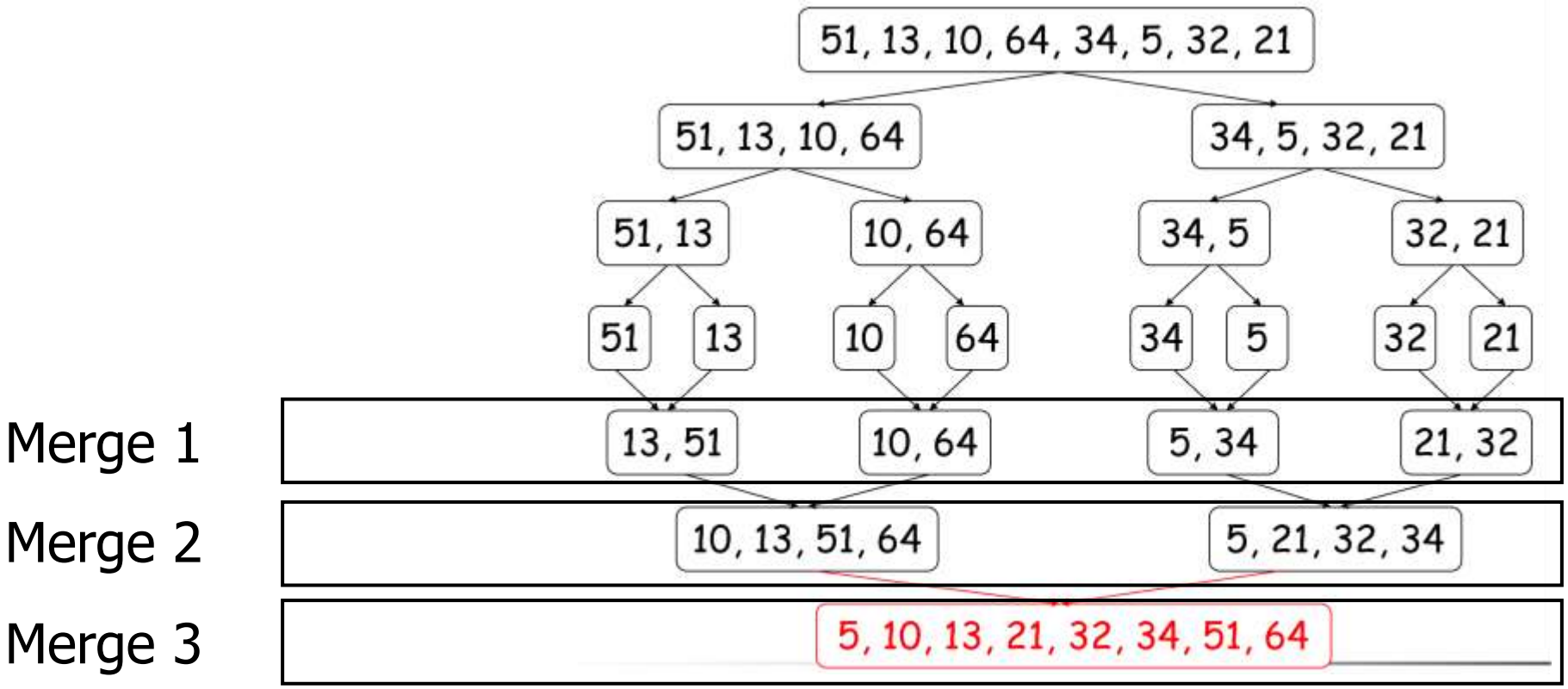


- **Conquer**
  - **merge** two sorted sequences of **total length n**



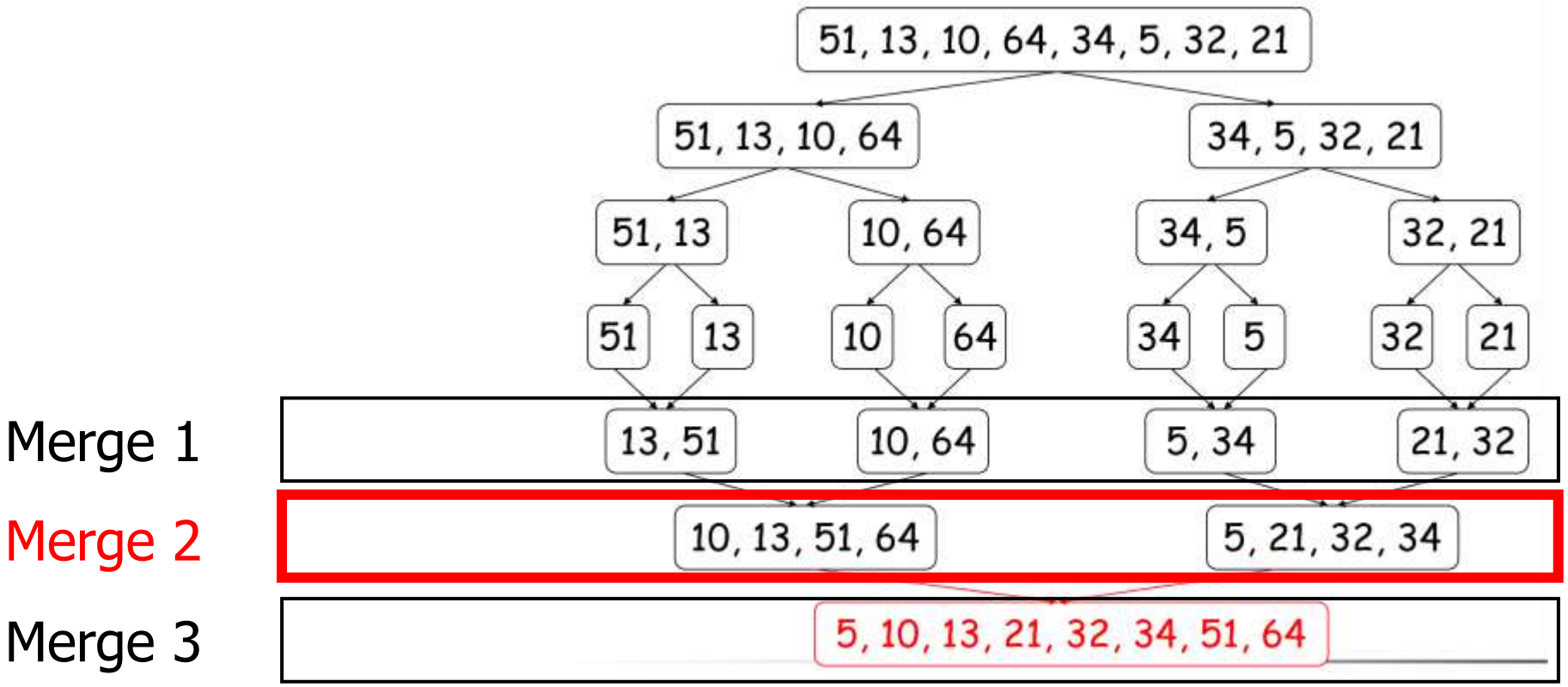
Merge algorithm (intuition)

- **Example**



Merge algorithm (intuition)

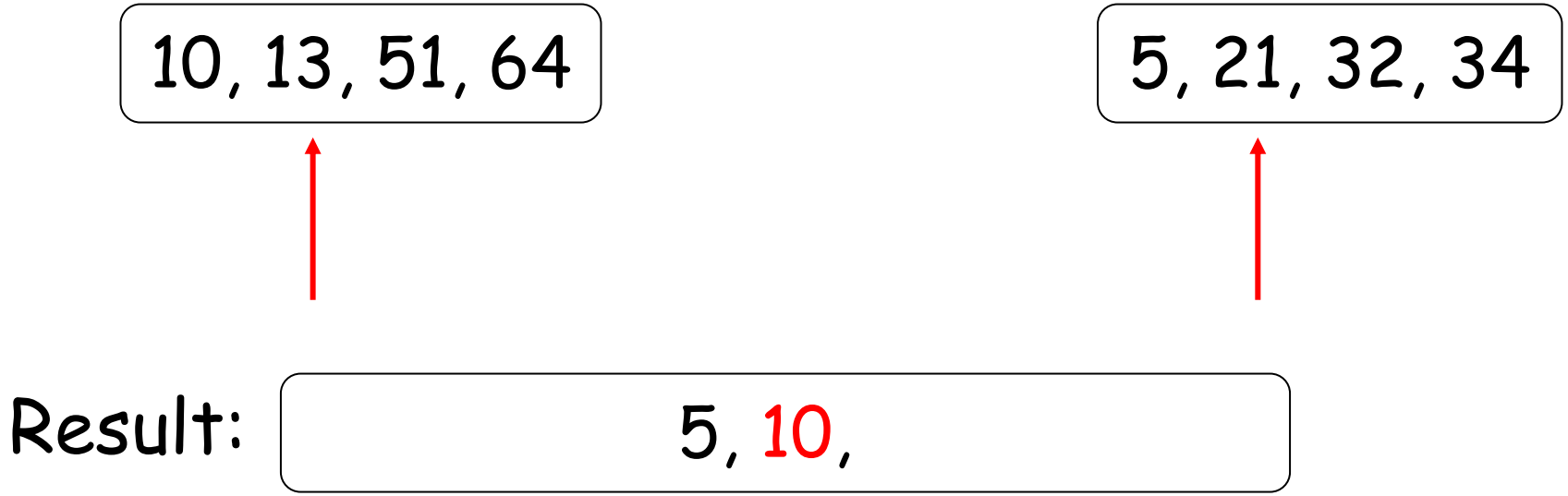
- **Example**



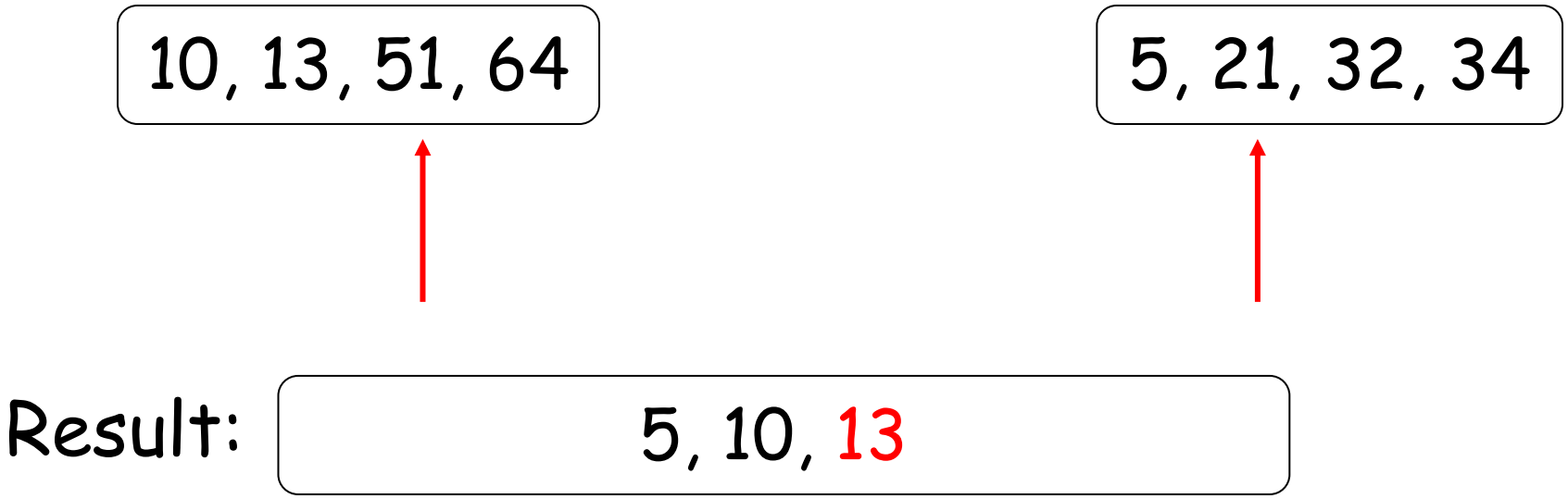
To merge two sorted sequences,
we keep two pointers, one to each sequence

Compare the two numbers pointed,
copy the smaller one to the result
and advance the corresponding pointer

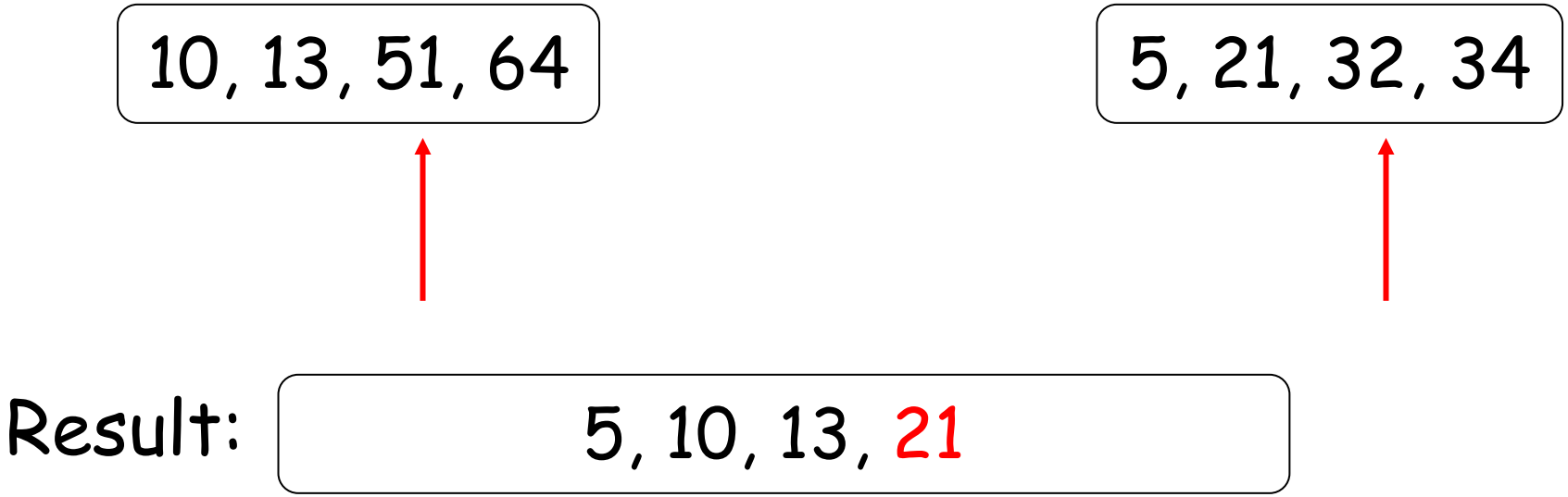
Then compare again the two numbers
pointed to by the pointer;
copy the smaller one to the result
and advance that pointer



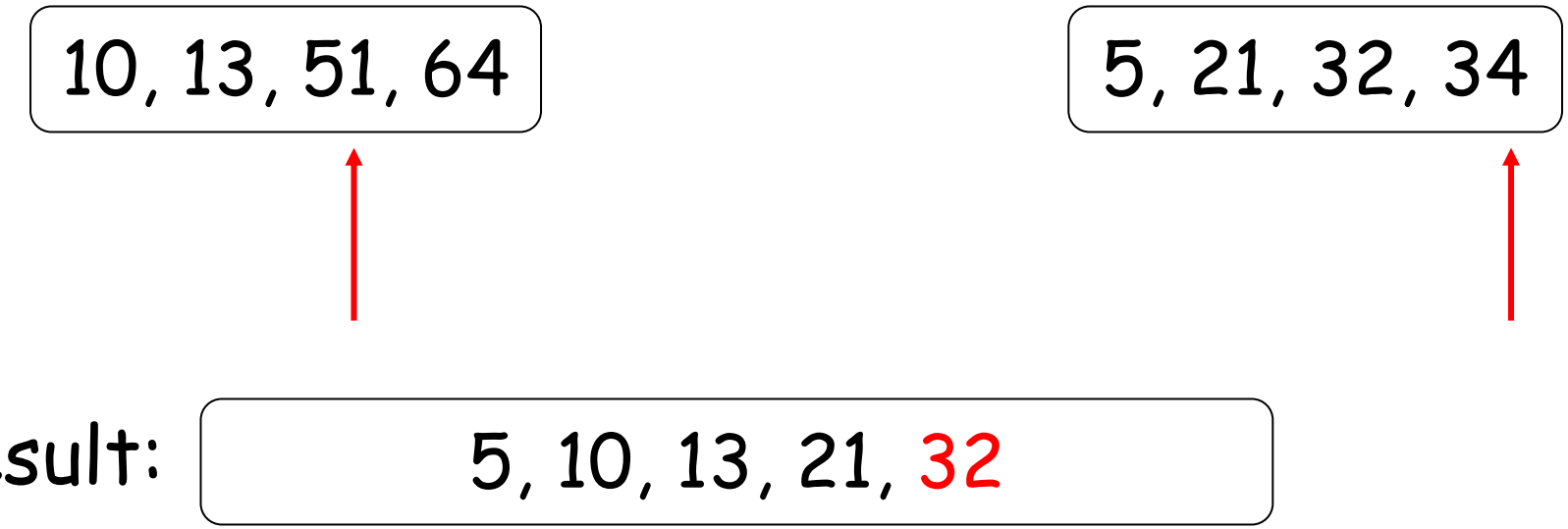
Repeat the same process …

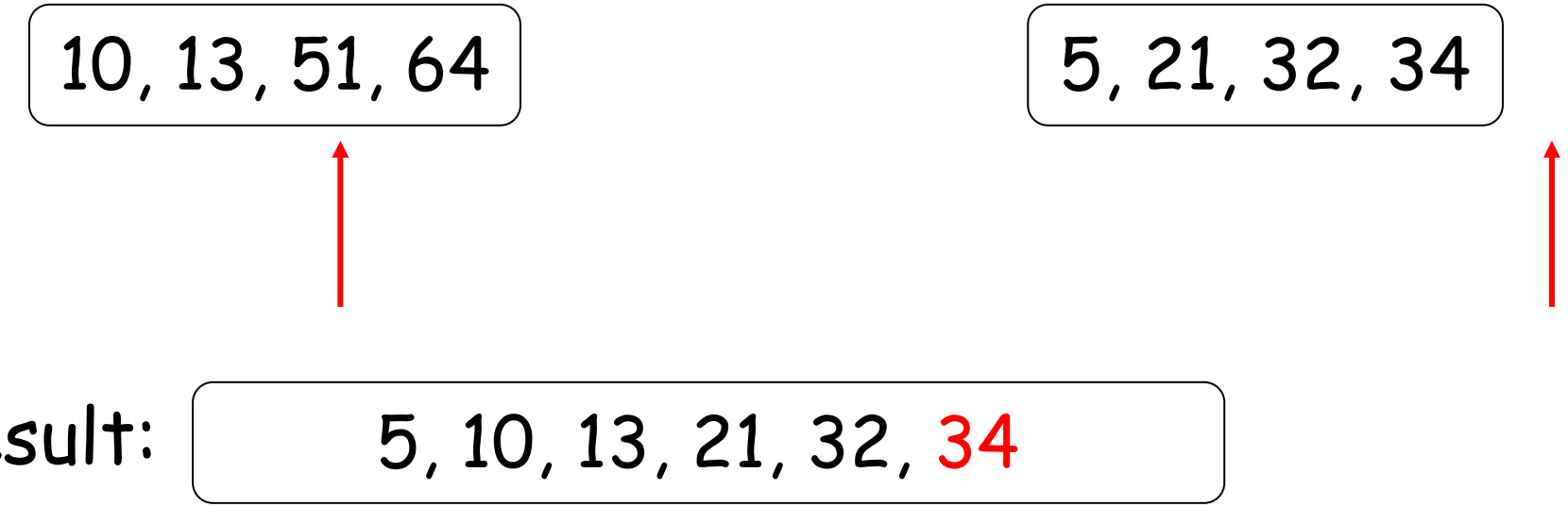


Again ...



and again ...





When we reach the end of one sequence, simply copy the remaining numbers in the other sequence to the result

Result: 5, 10, 13, 21, 32, 34, 51, 64

Then we obtain the final sorted sequence

Merge algorithm (implementation)

- **Pseudo-code**
- **Python**

Algorithm Mergesort(A[1..n])

if n > 1 then begin

copy A[1..⌊n/2⌋] to B[1..⌊n/2⌋]

copy A[⌊n/2⌋+1..n] to C[1..⌈n/2⌉]

Mergesort(B[1..⌊n/2⌋])

Mergesort(C[1..⌈n/2⌉])

Merge(B, C, A)

end

```
def merge_sort(unsorted_list):
    if len(unsorted_list) <= 1:
        return unsorted_list

    # Find the middle point and divide it
    middle = len(unsorted_list) // 2

    #divide the array into two halves
    left_list = unsorted_list[:middle]
    right_list = unsorted_list[middle:]

    #call merge_sort for the first half
    left_list = merge_sort(left_list)
    #call merge_sort for the second half
    right_list = merge_sort(right_list)

    return list(merge(left_list, right_list))
```

```
Algorithm Mergesort(A[1..n])
    if n > 1 then begin
        copy A[1..⌊n/2⌋] to B[1..⌊n/2⌋]
        copy A[⌊n/2⌋+1..n] to C[1..⌈n/2⌉]
        Mergesort(B[1..⌊n/2⌋])
        Mergesort(C[1..⌈n/2⌉])
        Merge(B, C, A)
    end
```

```python
def merge_sort(unsorted_list):
    if len(unsorted_list) <= 1:
        return unsorted_list

    # Find the middle point and divide it
    middle = len(unsorted_list) // 2

    #divide the array into two halves
    left_list = unsorted_list[:middle]
    right_list = unsorted_list[middle:]

    #call merge_sort for the first half
    left_list = merge_sort(left_list)
    #call merge_sort for the second half
    right_list = merge_sort(right_list)

    return list(merge(left_list, right_list))
```

Questions:

What are A, B, C? How about the Python code?

```
Algorithm Mergesort(A[1..n])
    if n > 1 then begin
        copy A[1..⌊n/2⌋] to B[1..⌊n/2⌋]
        copy A[⌊n/2⌋+1..n] to C[1..⌈n/2⌉]
        Mergesort(B[1..⌊n/2⌋])
        Mergesort(C[1..⌈n/2⌉])
        Merge(B, C, A)
    end
```

```python
def merge_sort(unsorted_list):
    if len(unsorted_list) <= 1:
        return unsorted_list

    # Find the middle point and divide it
    middle = len(unsorted_list) // 2

    #divide the array into two halves
    left_list = unsorted_list[:middle]
    right_list = unsorted_list[middle:]

    #call merge_sort for the first half
    left_list = merge_sort(left_list)
    #call merge_sort for the second half
    right_list = merge_sort(right_list)

    return list(merge(left_list, right_list))
```

Questions:
How many functions are there?

```
Algorithm Merge(B[1..p], C[1..q], A[1..p+q])
        set i=1, j=1, k=1
        while i<=p and j<=q do
        begin
                if B[i]<=C[j] then
                        set A[k] = B[i] and i = i+1
                else set A[k] = C[j] and j = j+1
                k = k+1
        end
        if i==p+1 then copy C[j..q] to A[k..(p+q)]
        else copy B[i..p] to A[k..(p+q)]
```

```python
def merge(left_half,right_half):
    res = left_half +right_half
    i=j=k=0
    while(i<=len(left_half)-1) and (j<=len(right_half)-1):

        if left_half[i] <= right_half[j]:
            res[k] = left_half[i]
            i = i + 1
        else:
            res[k] = right_half[j]
            j= j + 1
        k=k+1
    if(i==len(left_half)):
        res[k:len(res)]= right_half[j:len(right_half)]
    if(j==len(right_half)):
        res[k:len(res)]= left_half[i:len(right_half)]
    return res

arr = [64, 34, 25, 12, 22, 11, 90]
print(merge_sort(arr))
```

```
[11, 12, 22, 25, 34, 64, 90]
```

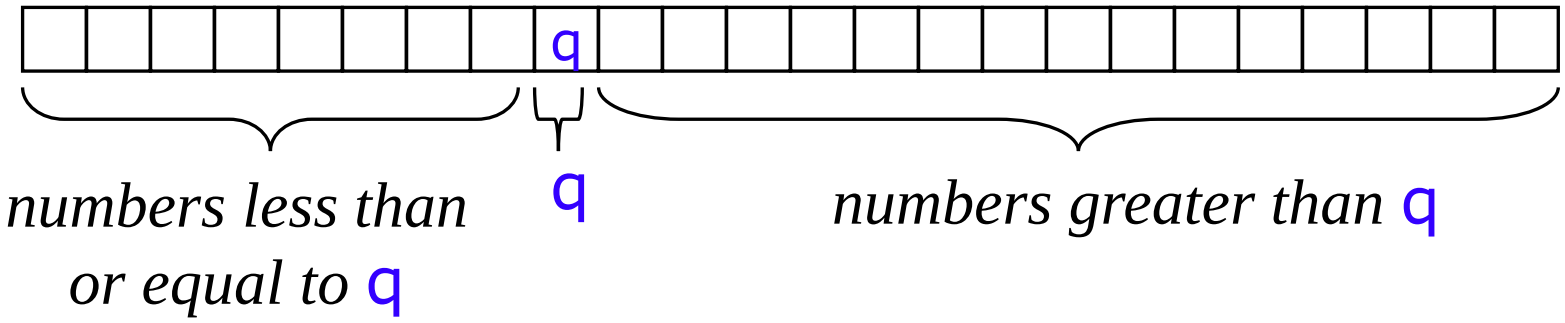
Quicksort

Quicksort

- Divide-and-conquer algorithm
- Sorts "in place" (like insertion sort, but not like merge sort).

Quicksort

- Pick some number q from the array
- Move all numbers less than (or equal to) q to the beginning of the array
- Move all number greater than q to the end of the array
- Quicksort the numbers less than or equal to q
- Quicksort the numbers greater than or equal to q



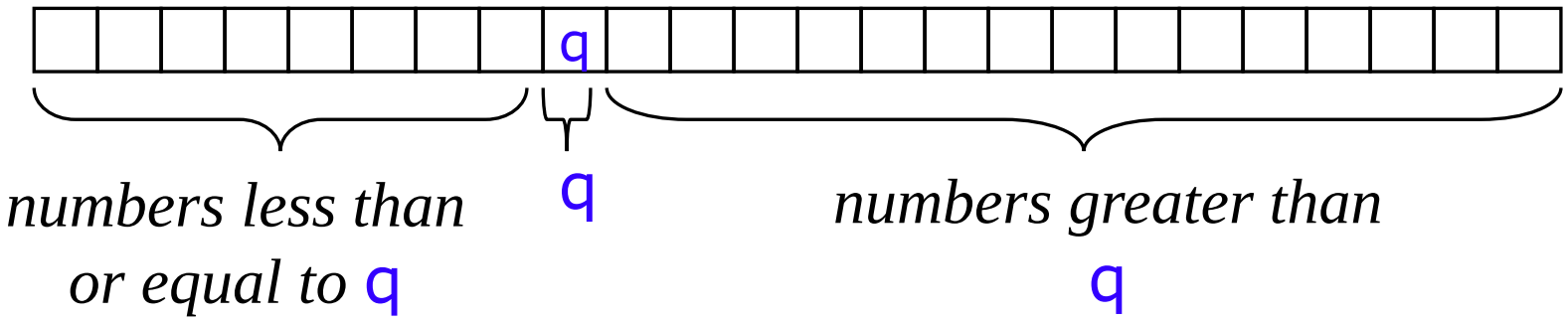
Quicksort pseudo code

```
Quicksort(A,p,r)
if p<r
        // partition an array such that:
        // all A[p,…,q-1] are less than or equal to A[q]
        // all A[q+1,…,r] are larger than A[q]
        q = Partition(A,p,r)
        Quicksort(A,p,q-1)
        Quicksort(A,q+1,r)
```

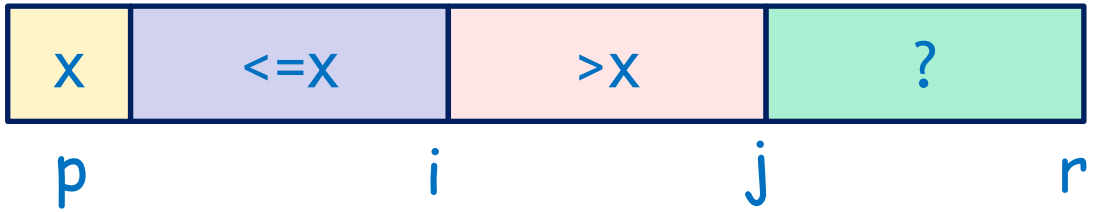
Initial call: Quicksort(A,1,n)

Partitioning

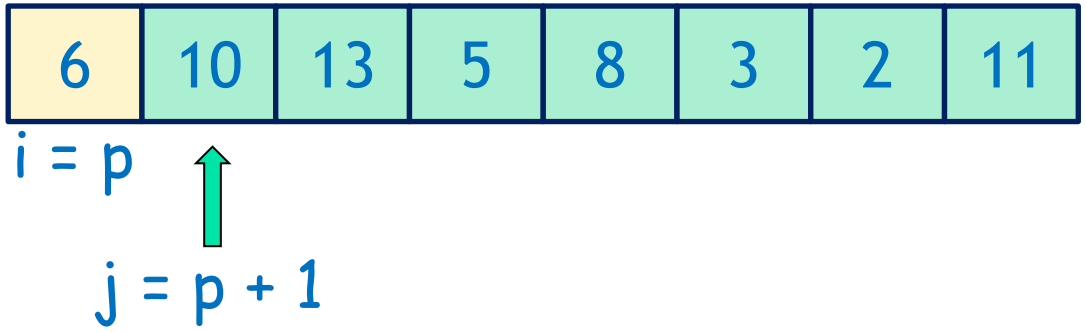
- A key step in the Quicksort algorithm is partitioning the array
  - We choose some (any) number q in the array to use as a pivot
  - We partition the array into three parts:



Partitioning

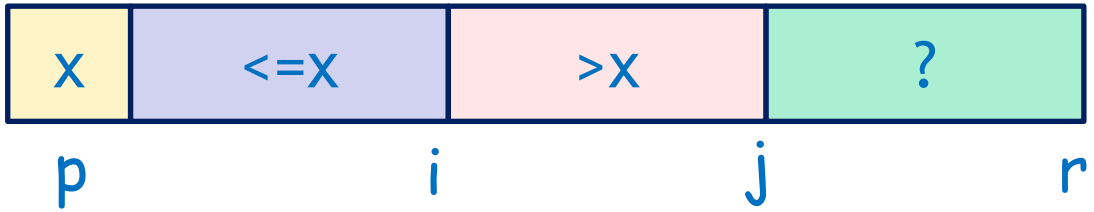


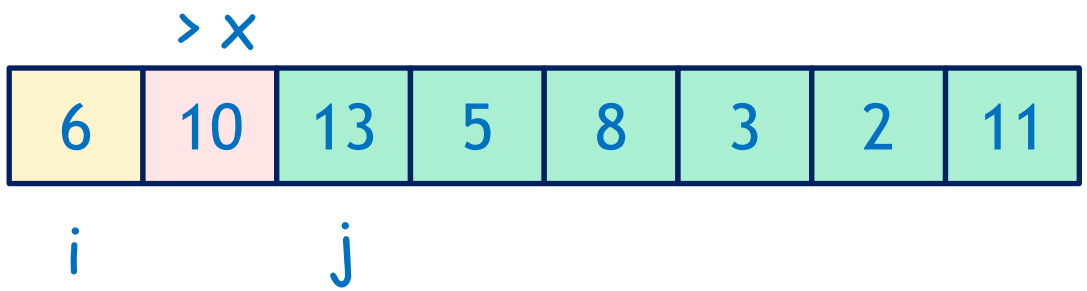

```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```



Partitioning

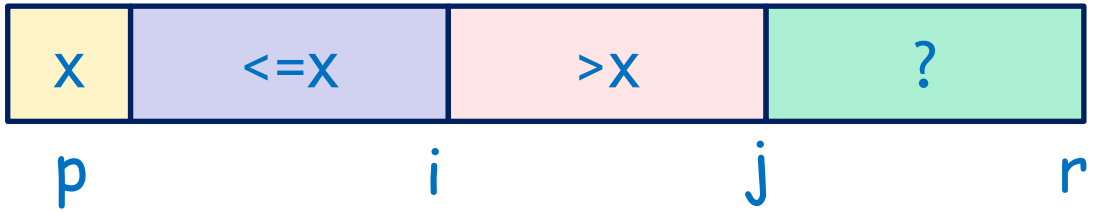
```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```

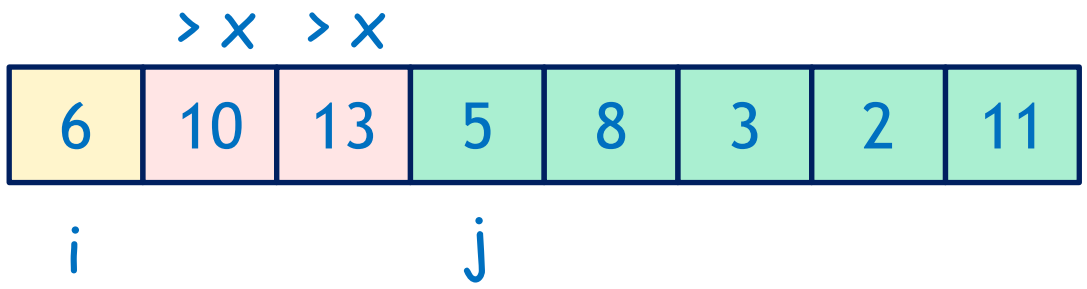




Partitioning

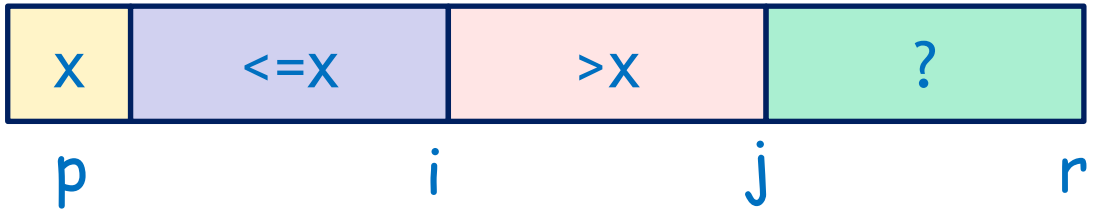
```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```

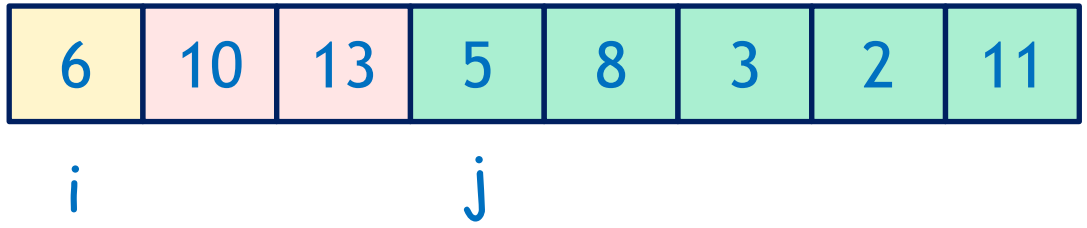


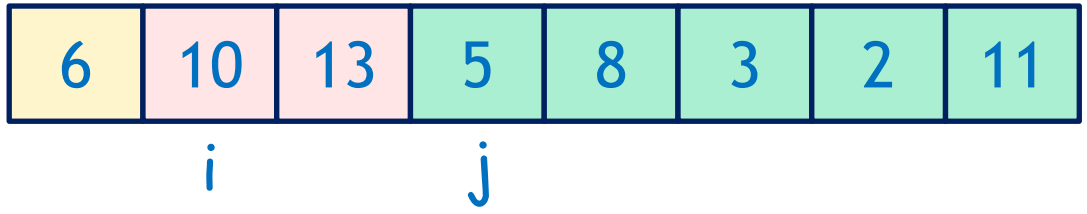


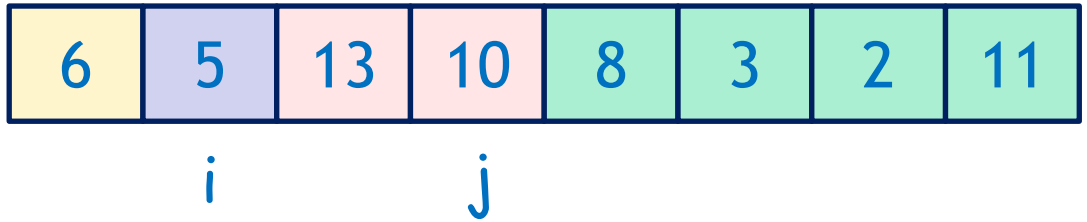
Partitioning

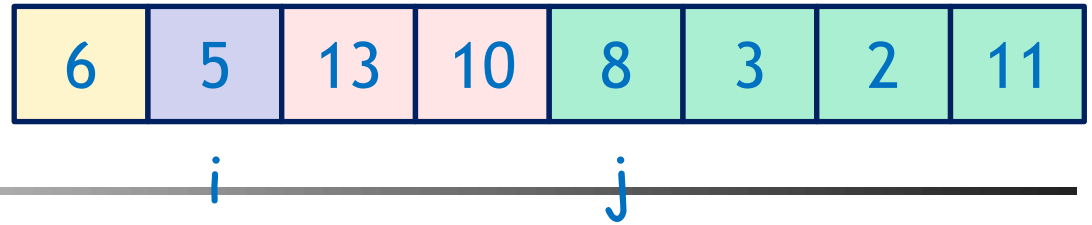
```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```



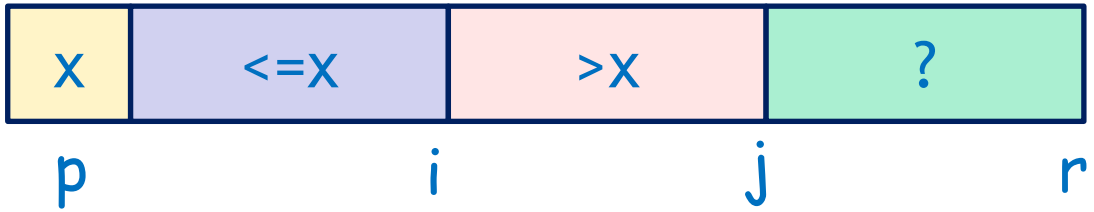




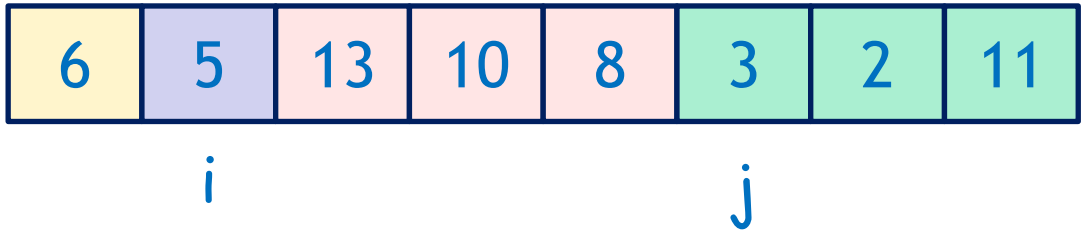




Partitioning

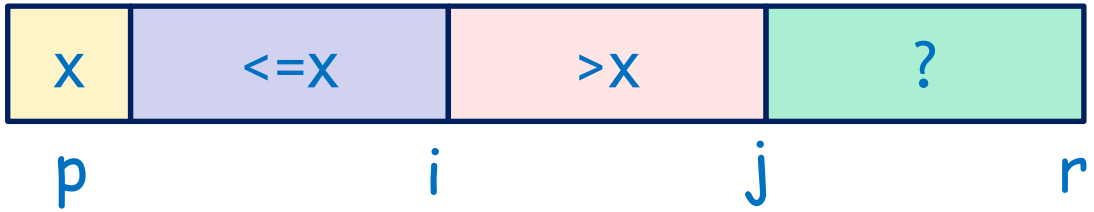


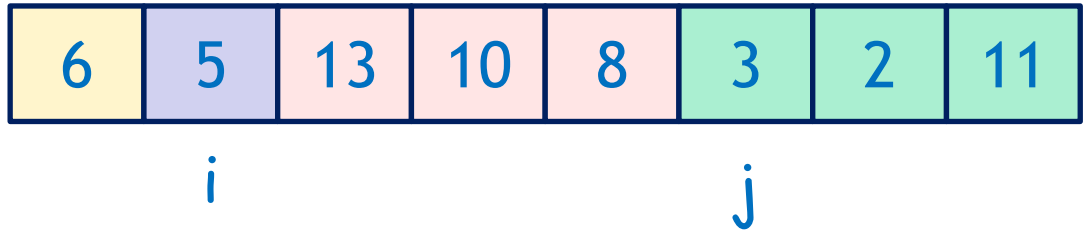

```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```

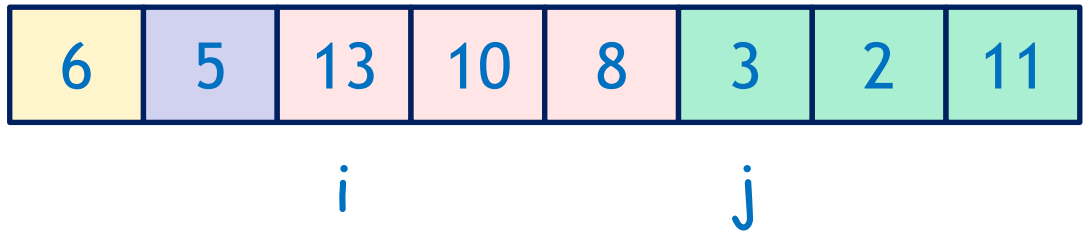


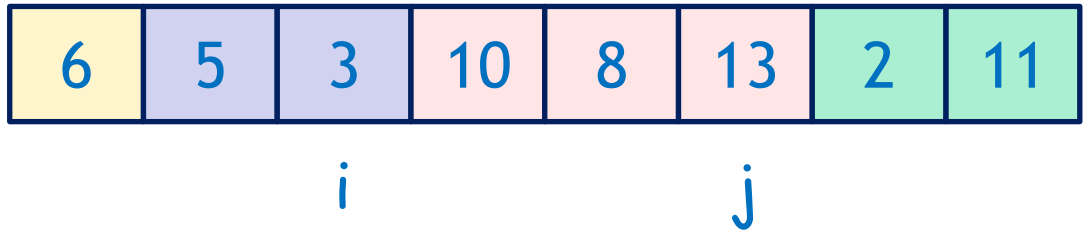
Partitioning

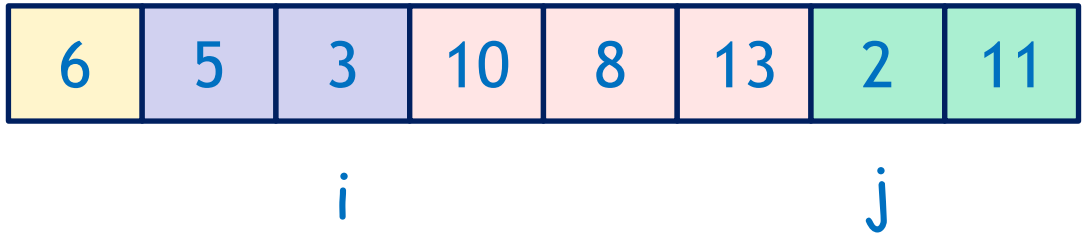
```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```





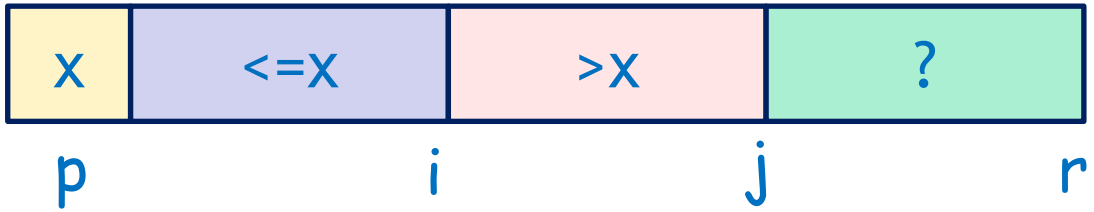


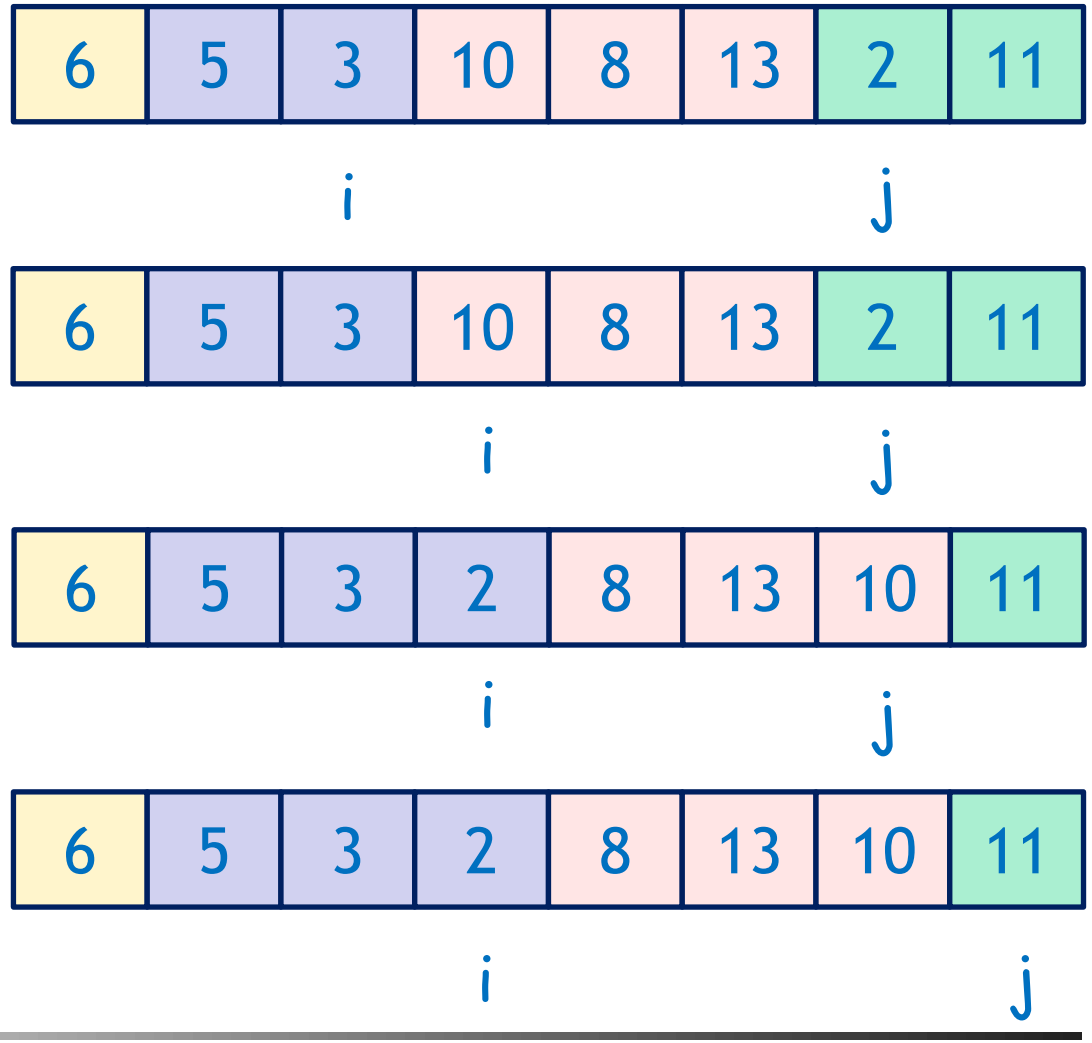




Partitioning

```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```





Partitioning

```
Partition(A,p,r)
x = A[p]
i = p
for j = p+1 to r
    if A[j] <= x
        i=i+1
        exchange A[i] with A[j]
exchange A[p] and A[i]
Return i
```