Q1

171044098
Akif KARTAL

Shell Sort step by step

* A is an ordered integer array with 10 elements from small to large

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
A =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

<u>Shell sort on this array</u>

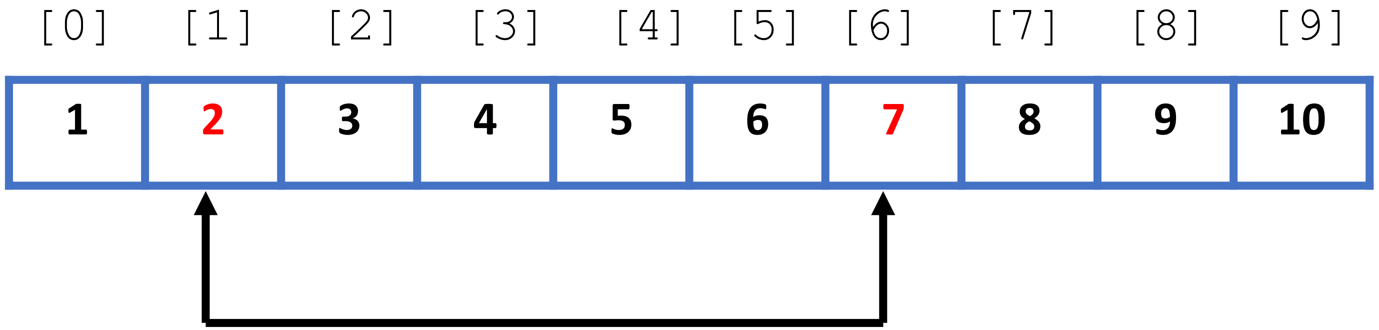
gap: 10/2 = 5

```
// Gap between adjacent elements.
int gap = table.length / 2;
```

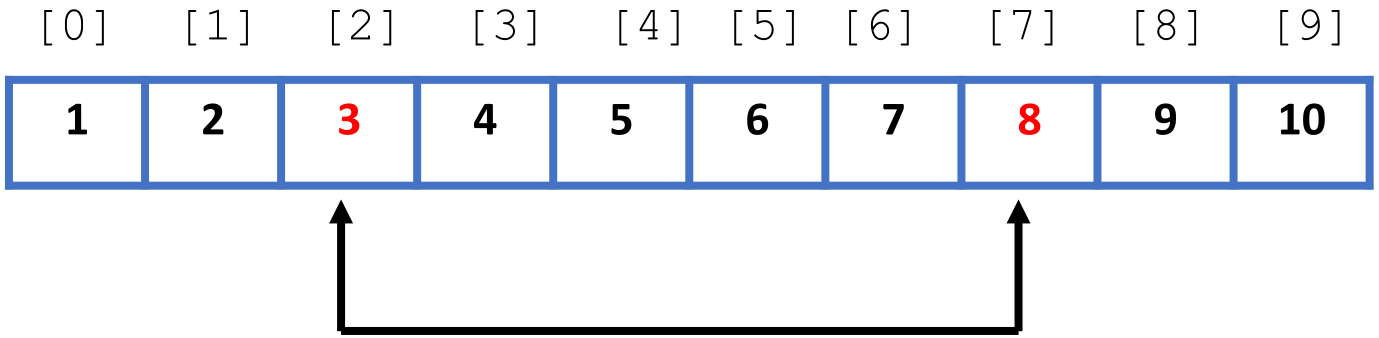
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Compare values between two gap index(5-0 = 5 is gap value) and swap them if second index value less than first index until last index.

gap : 5



gap : 5



Since all values are sorted there is no swap in this gap value when we reach last element we divide gap value by 2.2 and contiune with this gap value when gap equals zero we stop algorithm

gap 5 contiune comparing and swapping like this until the last element...

gap: 5/(2.2) = 2

```
// Reset gap for next pass.
if (gap == 2) {
    gap = 1;
} else {
    gap = (int) (gap / 2.2);
}
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

gap : 2

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

gap 2 contiune comparing and swapping like this until the last element...

gap : 1

```
// Reset gap for next pass.
if (gap == 2) {
    gap = 1;
} else {
    gap = (int) (gap / 2.2);
}
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

gap 1 contiune comparing and swapping like this until the last element...

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
A =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

<u>Analysis :</u>

Number of comparisons	Number of displacements
22	0

* B is an ordered integer array with 10 elements from large to small

gap = 5

```
// Gap between adjacent elements.
int gap = table.length / 2;
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
B =	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

gap = 5

B =

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
5	**9**	**8**	**7**	**6**	**10**	**4**	**3**	**2**	**1**

gap = 5

B =

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
5	**9**	**8**	**7**	**6**	**10**	**4**	**3**	**2**	**1**

gap = 5

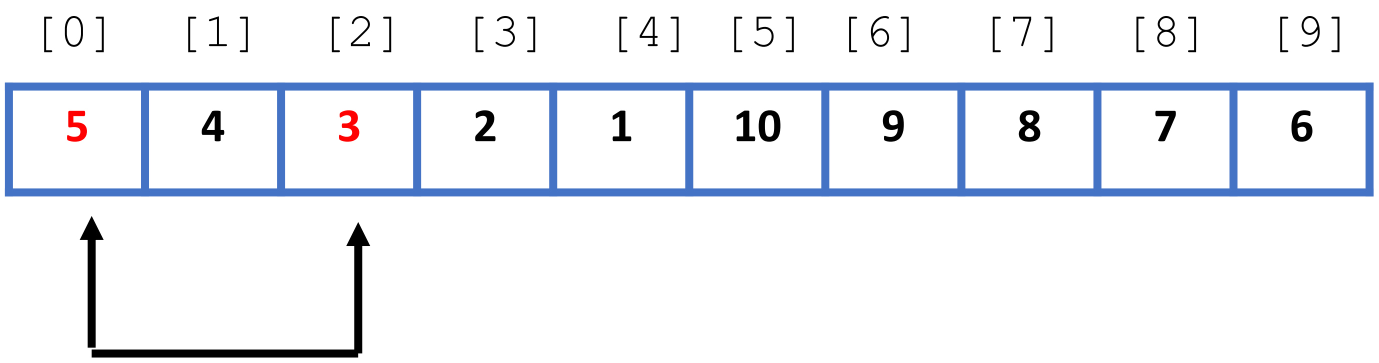
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
B =	**5**	**4**	**8**	**7**	**6**	**10**	**9**	**3**	**2**	**1**

gap 5 contiune comparing and swapping like this until the last element...

So we contiune like this until when we reach last element and we divide gap value by 2.2 and contiune with this gap value when gap equals zero we stop algorithm.

gap = 2

```
// Reset gap for next pass.
if (gap == 2) {
      gap = 1;
} else {
      gap = (int) (gap / 2.2);
}
```



gap 2 contiune comparing and swapping like this until the last element...

gap = 1

```
// Reset gap for next pass.
if (gap == 2) {
      gap = 1;
} else {
      gap = (int) (gap / 2.2);
}
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
3	2	1	4	9	5	7	6	10	8

gap 1 contiune comparing and swapping like this until the last element...

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
B =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

<u>Analysis :</u>

Number of comparisons	Number of displacements
22	13

* C = {5, 2, 13, 9, 1, 7, 6, 8, 1, 15, 4, 11}

gap: 12/2 = 6

C =

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	**2**	**13**	**9**	**1**	**7**	**6**	**8**	**1**	**15**	**4**	**11**

gap: 6

C =

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	13	9	1	7	6	8	1	15	4	11

gap 6 contiune comparing and swapping like this until the last element...

gap: 6/(2.2) = 2

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	5	2	1	9	1	7	6	8	13	15	4	11

gap 2 contiune comparing and swapping like this until the last element...

gap = 1

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	**1**	**2**	**1**	**7**	**5**	**8**	**4**	**9**	**6**	**11**	**13**	**15**

gap 1 contiune comparing and swapping like this until the last element...

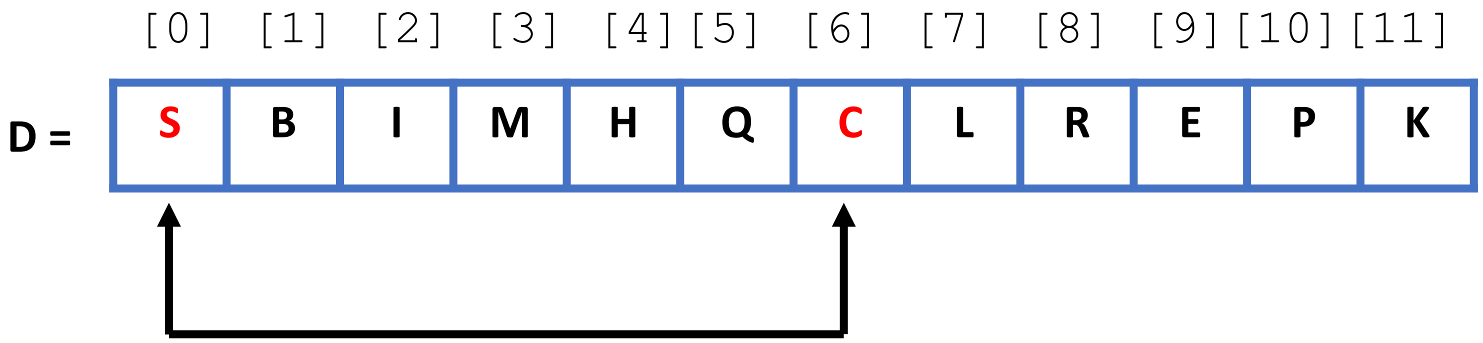
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	**1**	**1**	**2**	**4**	**5**	**6**	**7**	**8**	**9**	**11**	**13**	**15**

Analysis :

Number of comparisons	Number of displacements
27	16

*** D = {‘S’, ‘B’, ‘I’, ‘M’, ‘H’, ‘Q’, ‘C’, ‘L’, ‘R’, ‘E’, ‘P’, ‘K’}**

gap = 6



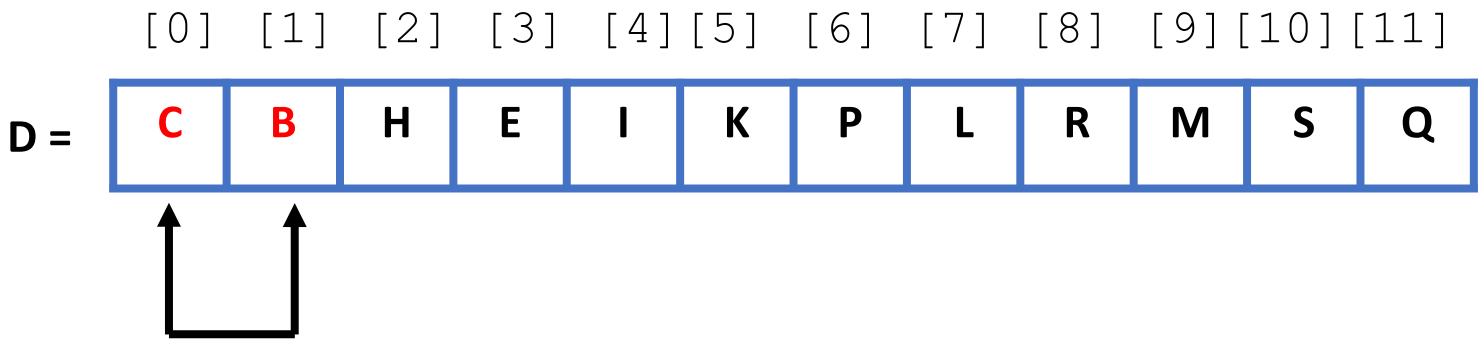
gap 6 contiune comparing and swapping like this until the last element...

gap = 2

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
D =	**C**	**B**	**I**	**E**	**H**	**K**	**S**	**L**	**R**	**M**	**P**	**Q**

gap 2 contiune comparing and swapping like this until the last element...

gap = 1



gap 1 contiune comparing and swapping like this until the last element...

D =

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	**C**	**E**	**H**	**I**	**K**	**L**	**M**	**P**	**Q**	**R**	**S**

<u>Analysis :</u>

Number of comparisons	**Number of displacements**
27	14

Merge Sort step by step

* A is an ordered integer array with 10 elements from small to large

A =	1	2	3	4	5	6	7	8	9	10

First we divide the array into 2 pieces until we get a just one element array since 1 element is already sorted.

Then, when we reach the just one element we start to merge the sub array while merging we Sort them and then do merge.

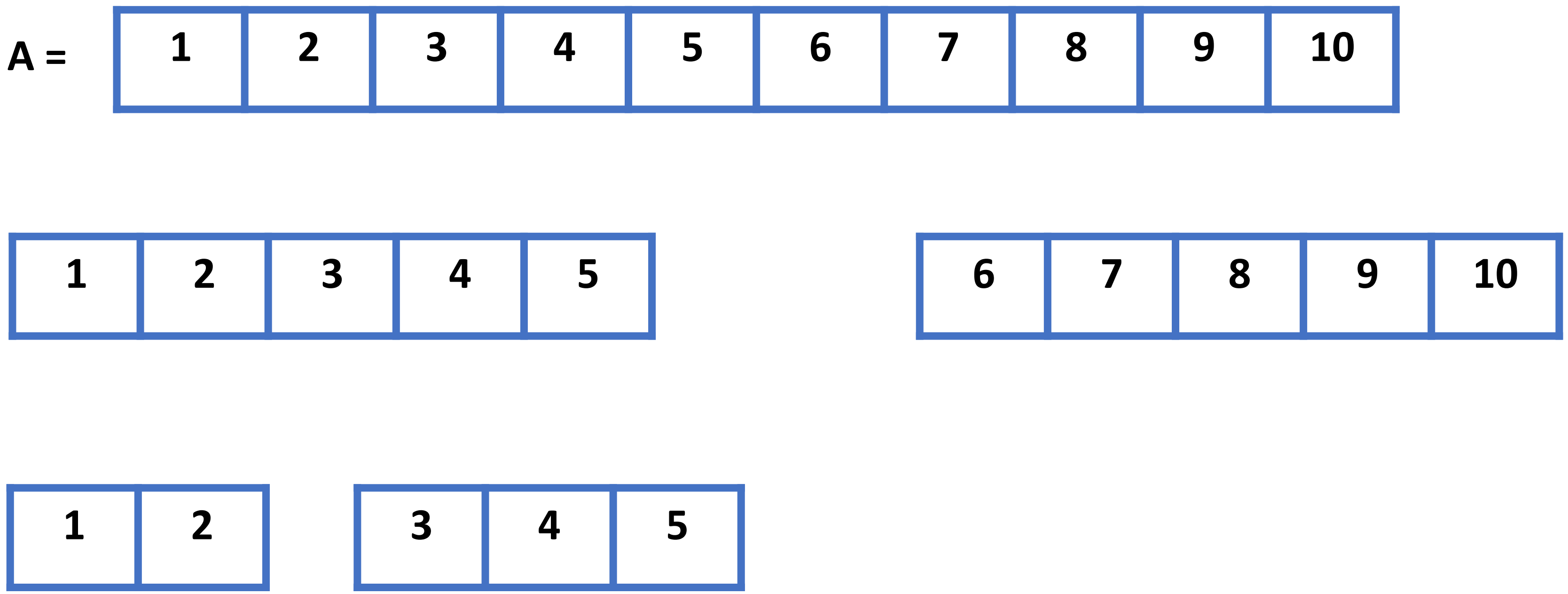
If we divide the array we will get;

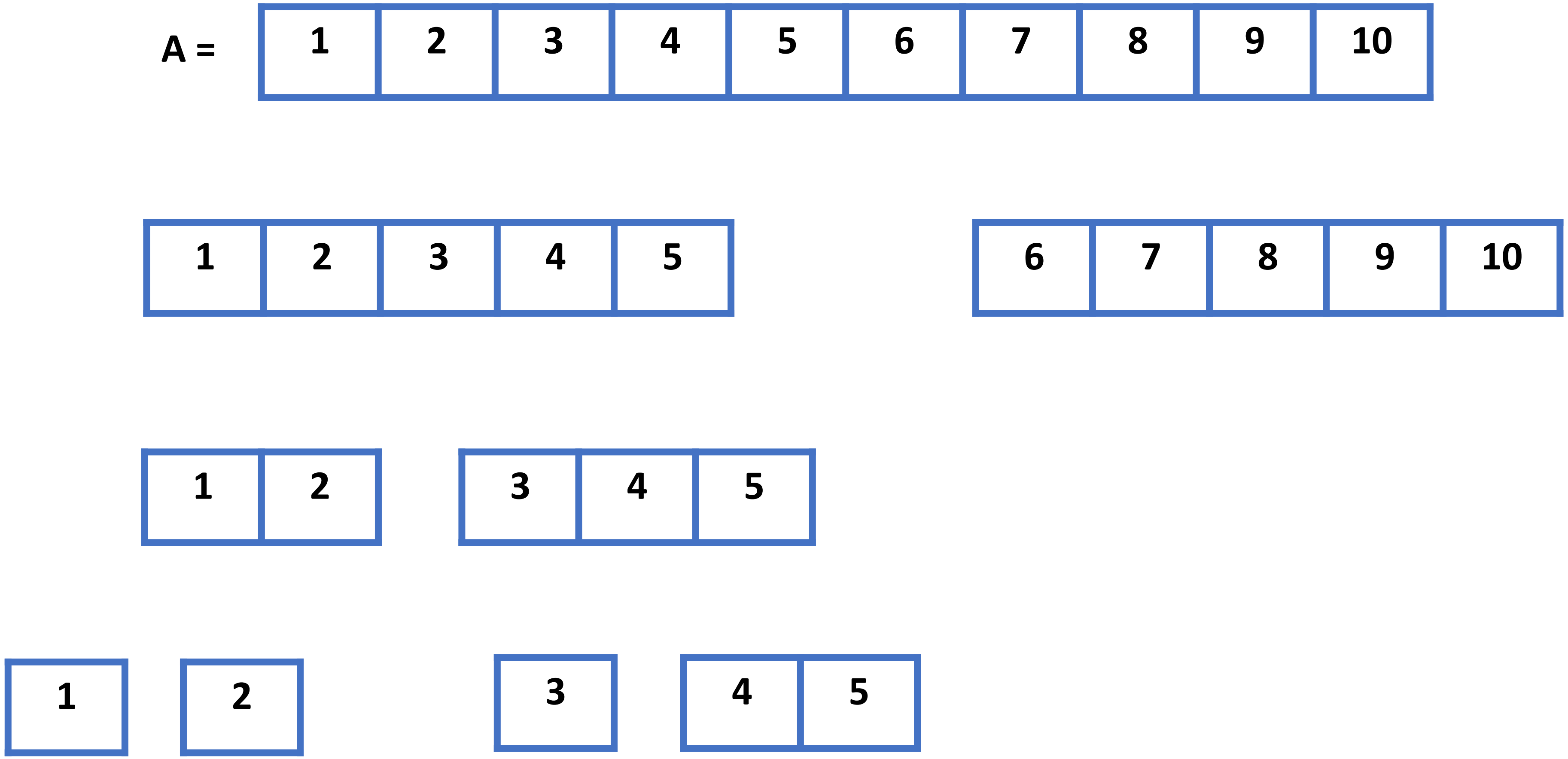
A =

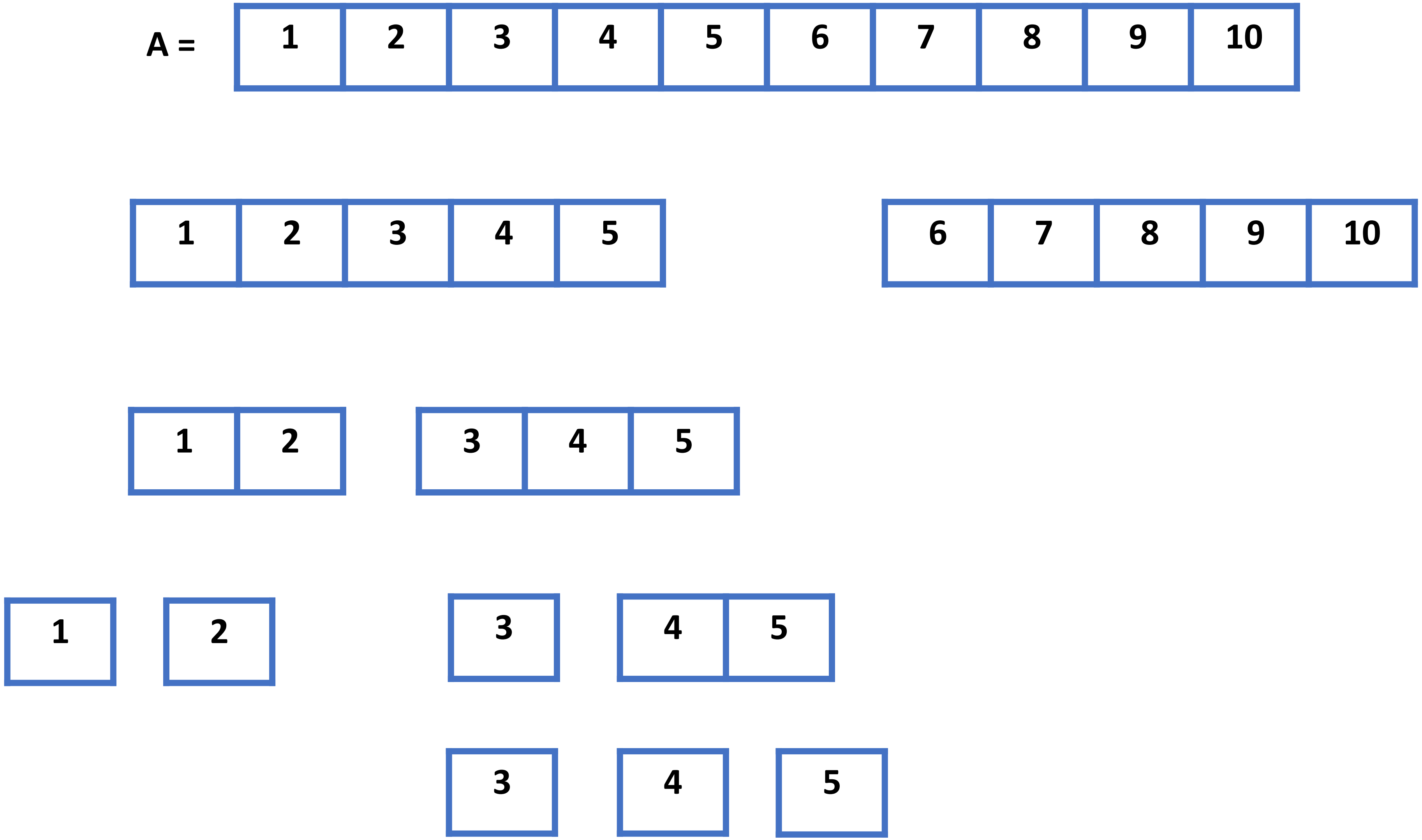
1	2	3	4	5	6	7	8	9	10

1	2	3	4	5

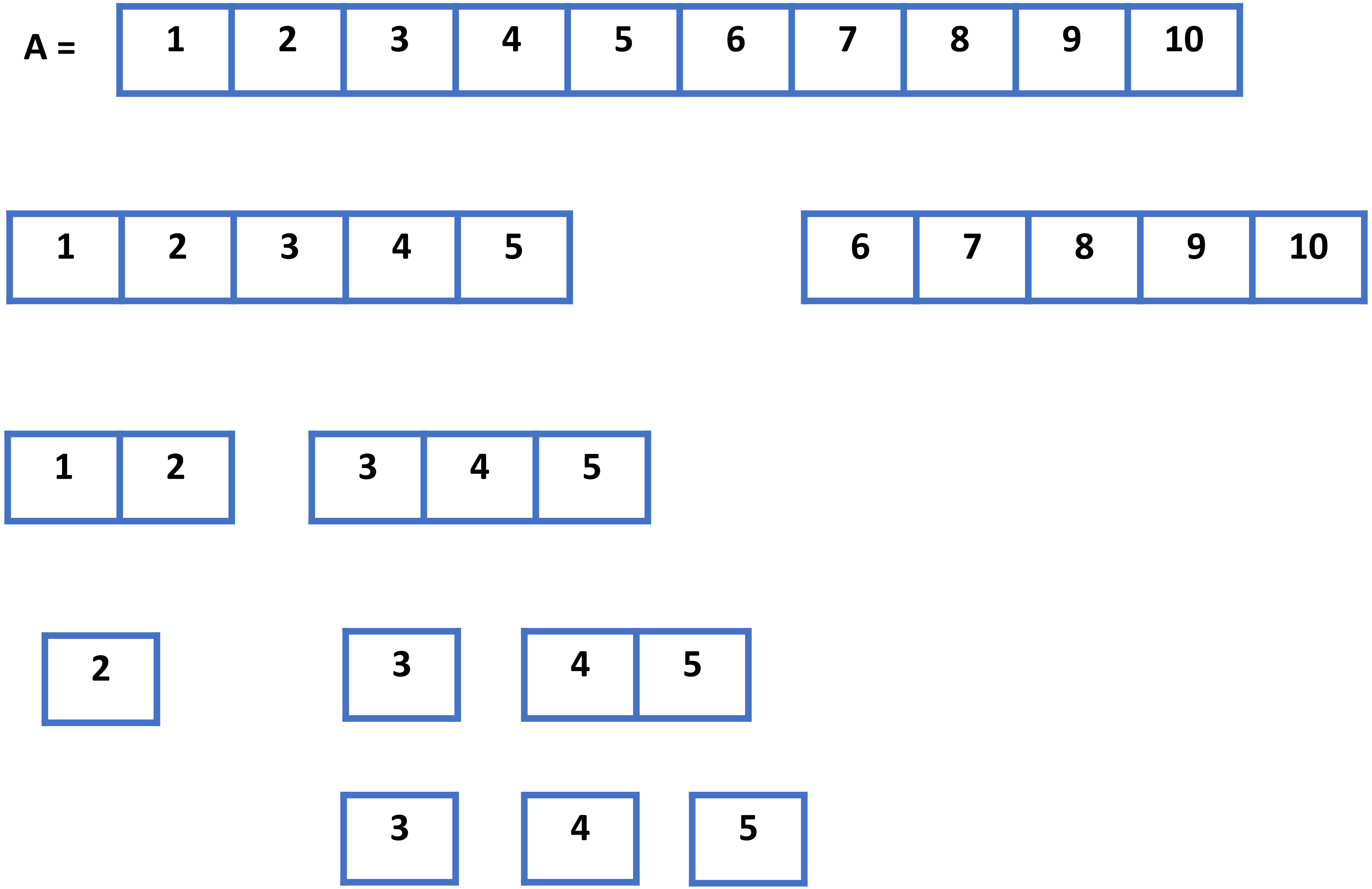
6	7	8	9	10

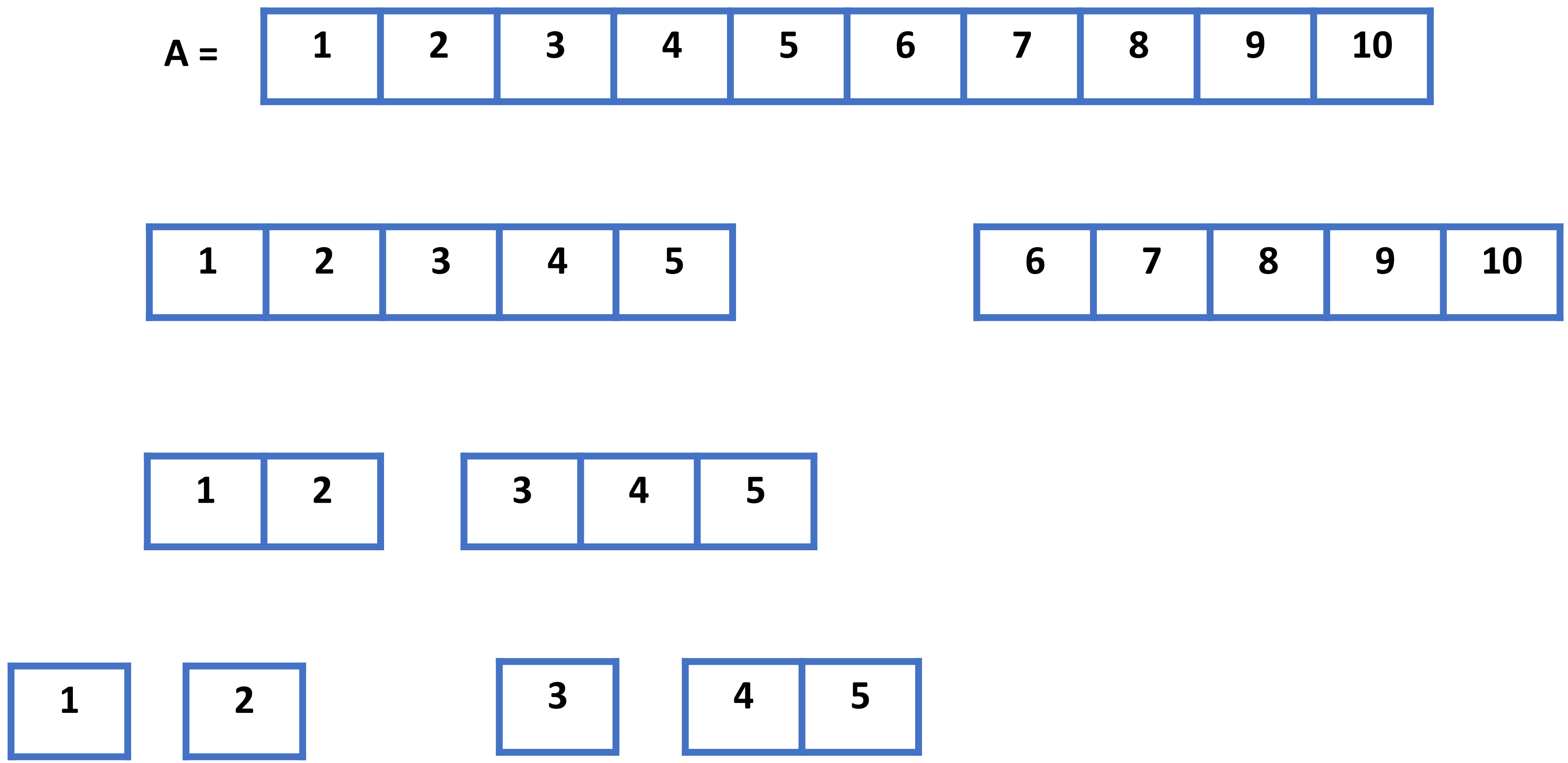

A =
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5
6 7 8 9 10
1 2
3 4 5

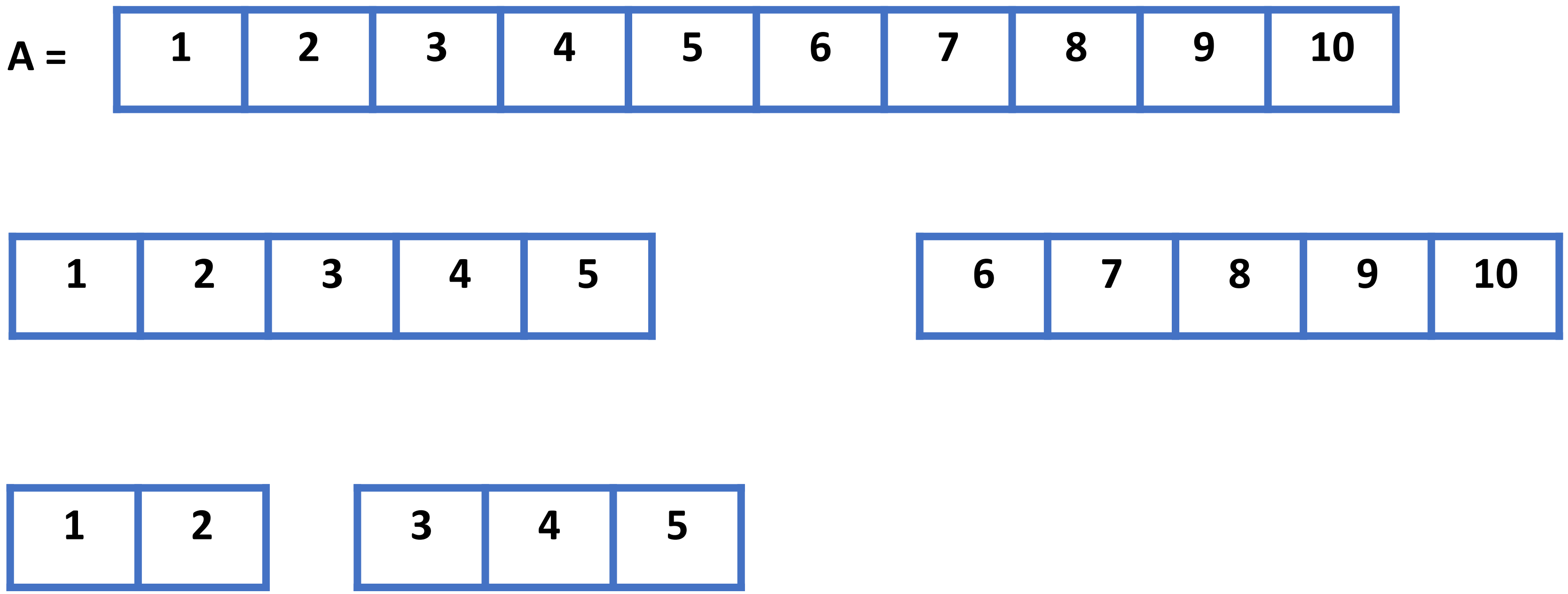

A =
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5
6 7 8 9 10
1 2
3 4 5
1
2
3
4 5


A =
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5
6 7 8 9 10
1 2
3 4 5
1
2
3
4 5
3
4
5

Now we start to merge operation...


A =
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5
6 7 8 9 10
1 2
3 4 5
1
2
3
4 5
3
4
5


A =
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5
6 7 8 9 10
1 2
3 4 5
1
2
3
4 5


A =
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5
6 7 8 9 10
1 2
3 4 5

A =

1	2	3	4	5	6	7	8	9	10

1	2	3	4	5

6	7	8	9	10

We apply same process to the right side of array then we will get;

A =

1	2	3	4	5	6	7	8	9	10

1	2	3	4	5

6	7	8	9	10

A =

1	2	3	4	5	6	7	8	9	10

We get sorted array;

A =									
1	2	3	4	5	6	7	8	9	10

Analysis :

Number of comparisons	Number of displacements
24	0

* B is an ordered integer array with 10 elements from large to small

B =

10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

First we divide the array into 2 pieces until we get a just one element array since 1 element is already sorted.

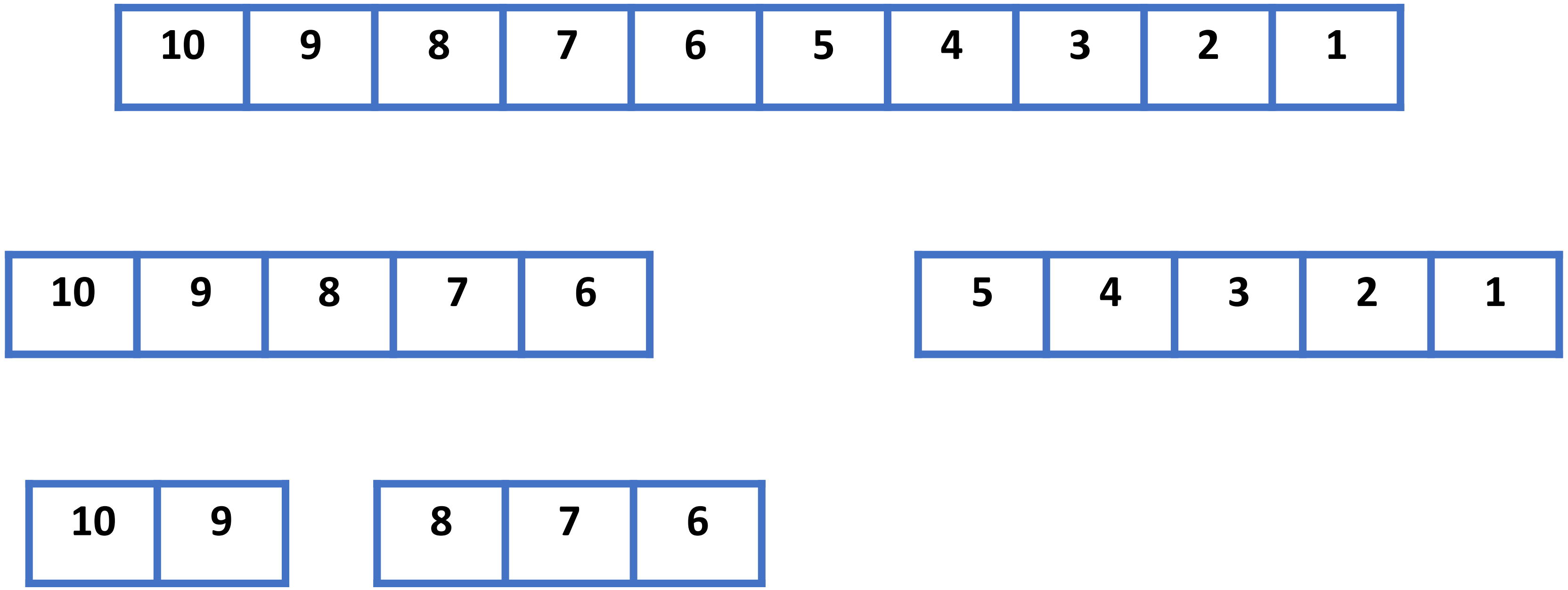
Then, when we reach the just one element we start to merge the sub array while merging we Sort them and then do merge.

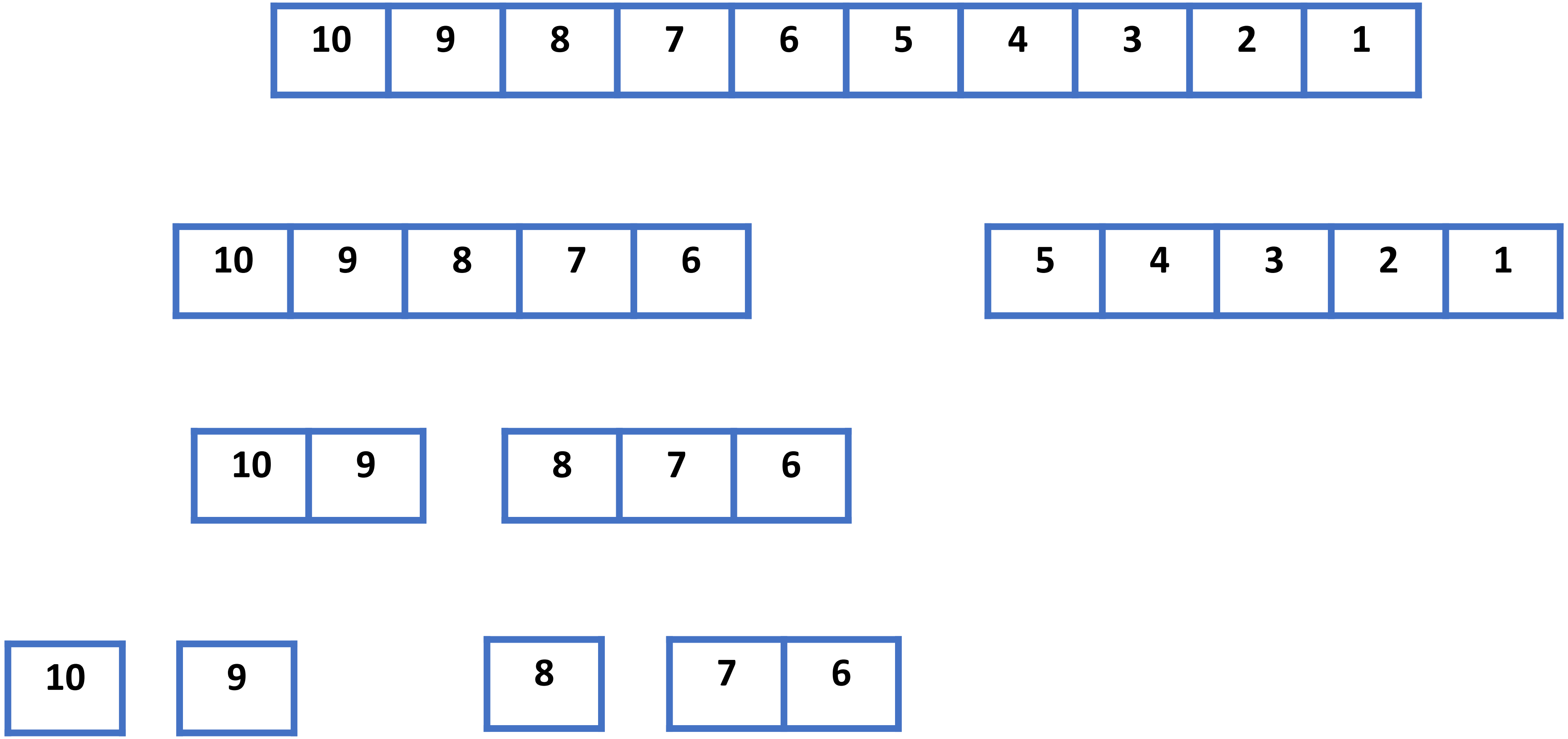
If we divide the array we will get;

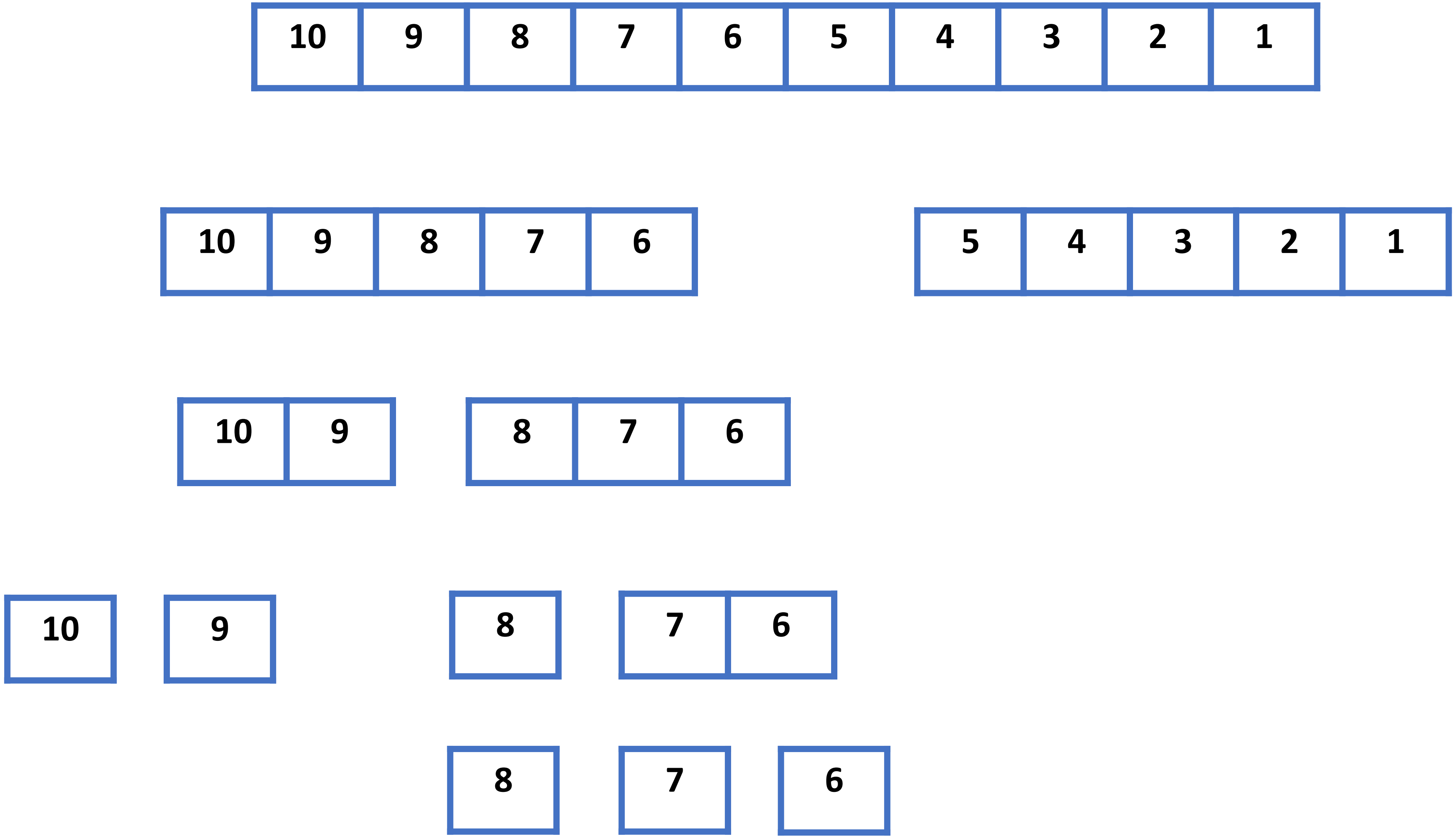
10	9	8	7	6	5	4	3	2	1

10	9	8	7	6

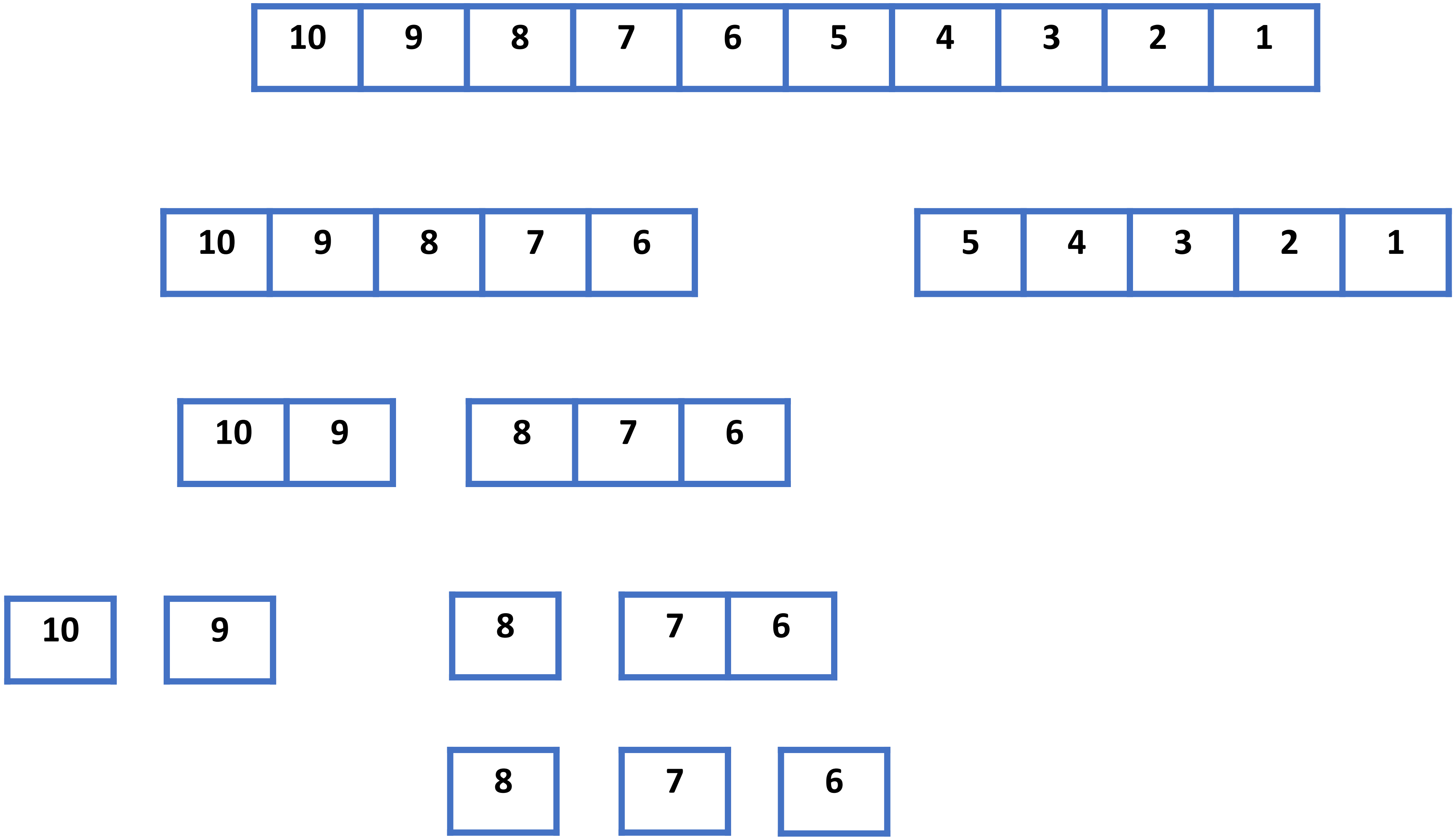
5	4	3	2	1

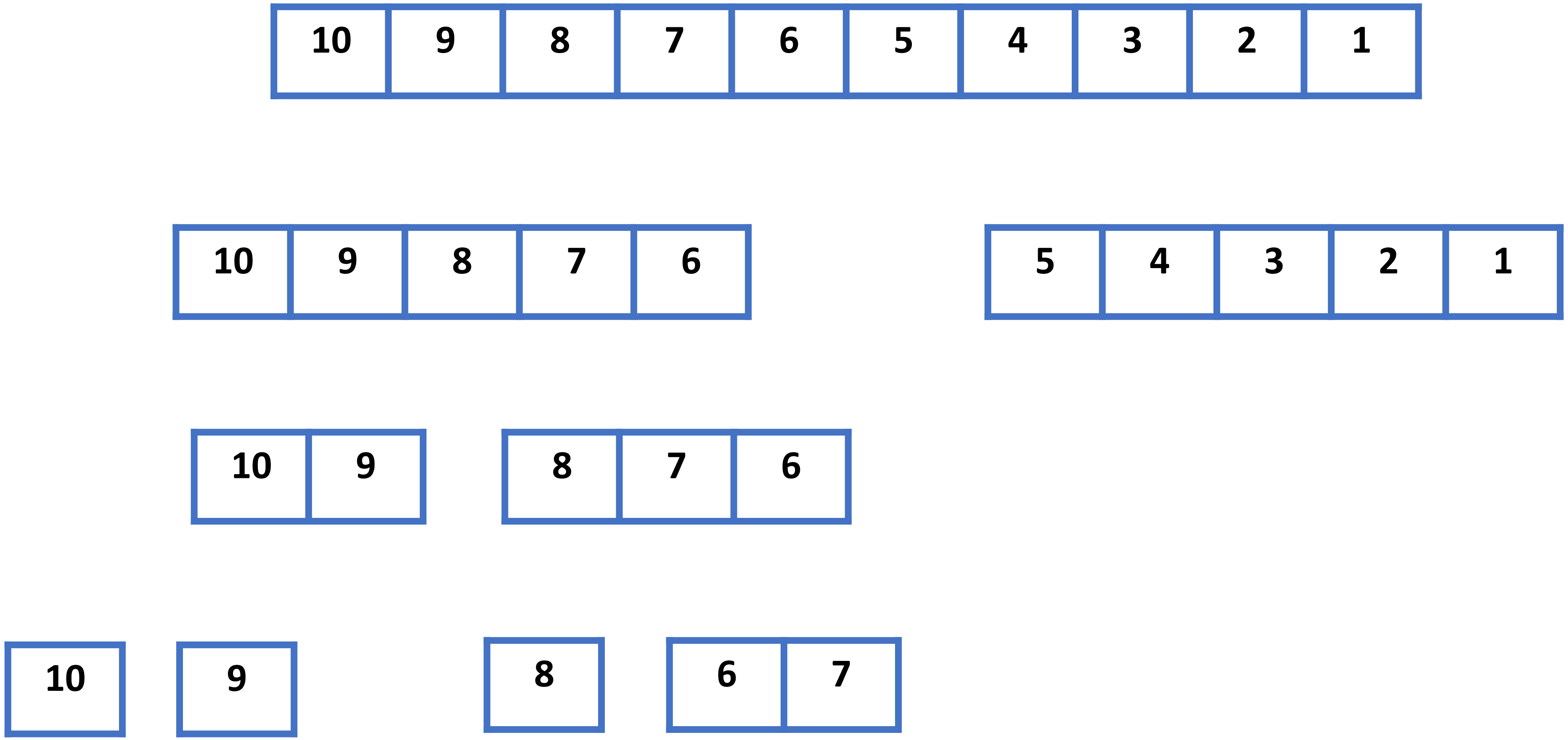

10 9 8 7 6 5 4 3 2 1
10 9 8 7 6
5 4 3 2 1
10 9
8 7 6


10 9 8 7 6 5 4 3 2 1
10 9 8 7 6
5 4 3 2 1
10 9
8 7 6
10
9
8
7 6


10 9 8 7 6 5 4 3 2 1
10 9 8 7 6
5 4 3 2 1
10 9
8 7 6
10
9
8
7 6
8
7
6

Now we start to merge operation...


10 9 8 7 6 5 4 3 2 1
10 9 8 7 6
5 4 3 2 1
10 9
8 7 6
10
9
8
7 6
8
7
6


10 9 8 7 6 5 4 3 2 1
10 9 8 7 6
5 4 3 2 1
10 9
8 7 6
10
9
8
6 7

10	9	8	7	6	5	4	3	2	1

10	9	8	7	6

5	4	3	2	1

9	10

6	7	8

10	9	8	7	6	5	4	3	2	1

6	7	8	9	10

5	4	3	2	1

We apply same process to the right side of array then we will get;

10	9	8	7	6	5	4	3	2	1

6	7	8	9	10

1	2	3	4	5

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

We get sorted array;

B =

1	2	3	4	5	6	7	8	9	10

Analysis :

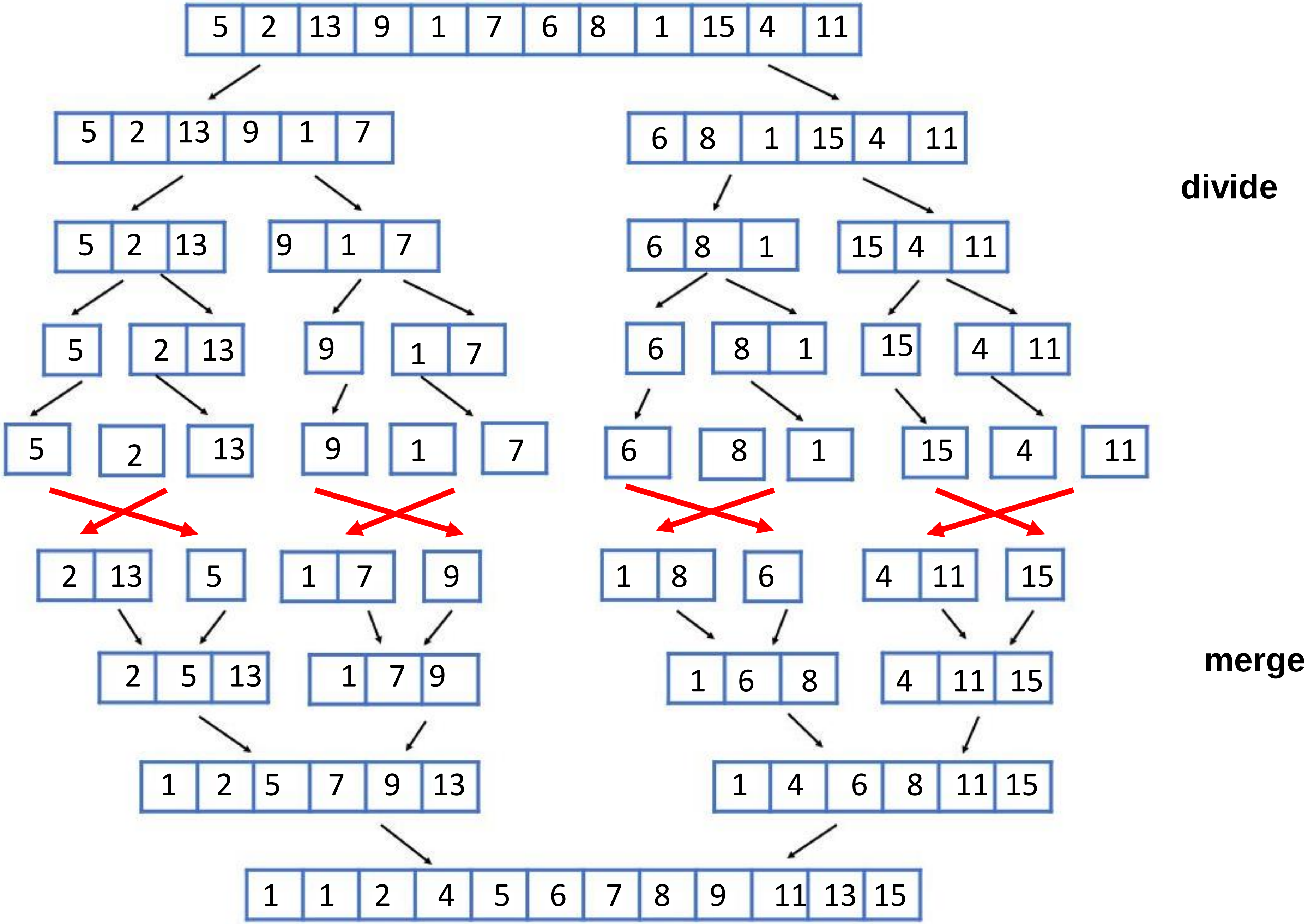
Number of comparisons	Number of displacements
28	15

* $C = \{5, 2, 13, 9, 1, 7, 6, 8, 1, 15, 4, 11\}$

C =

5	2	13	9	1	7	6	8	1	15	4	11

If we apply merge sort algorithm



5 2 13 9 1 7 6 8 1 15 4 11
5 2 13 9 1 7
6 8 1 15 4 11
divide
5 2 13
9 1 7
6 8 1
15 4 11
5
2 13
9
1 7
6
8 1
15
4 11
5
2
13
9
1
7
6
8
1
15
4
11
2 13
5
1 7
9
1 8
6
4 11
15
2 5 13
1 7 9
1 6 8
4 11 15
merge
1 2 5 7 9 13
1 4 6 8 11 15
1 1 2 4 5 6 7 8 9 11 13 15

C =

1	**1**	**2**	**4**	**5**	**6**	**7**	**8**	**9**	**11**	**13**	**15**

Analysis :

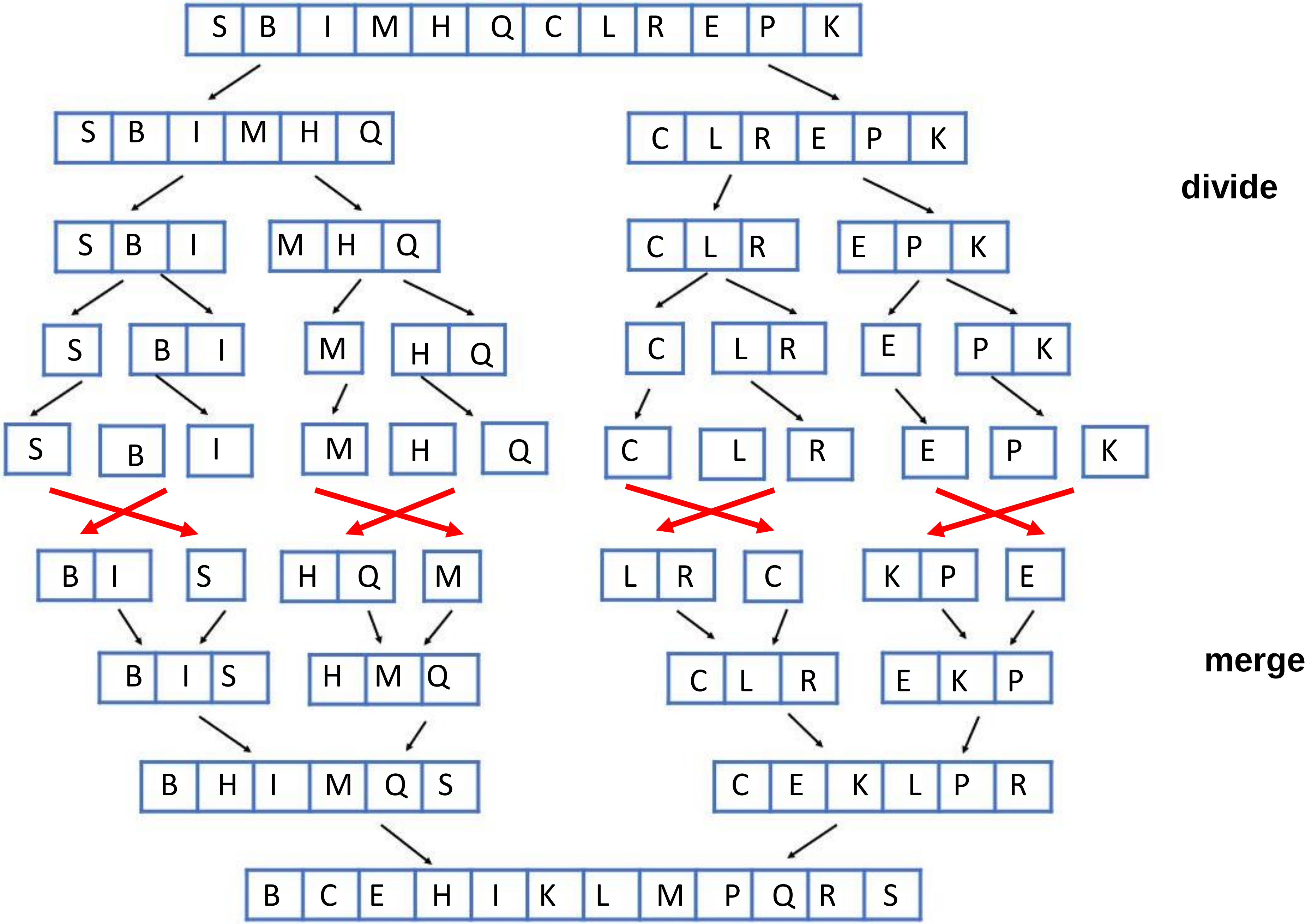
Number of comparisons	Number of displacements
43	31

* D = {‘S’, ‘B’, ‘I’, ‘M’, ‘H’, ‘Q’, ‘C’, ‘L’, ‘R’, ‘E’, ‘P’, ‘K’}

D =

S	**B**	**I**	**M**	**H**	**Q**	**C**	**L**	**R**	**E**	**P**	**K**

If we apply merge sort algorithm


S B I M H Q C L R E P K
S B I M H Q
C L R E P K
divide
S B I
M H Q
C L R
E P K
merge
B I S
H M Q
C L R
E K P
B H I M Q S
C E K L P R
B C E H I K L M P Q R S

D =

B	C	E	H	I	K	L	M	P	Q	R	S

Analysis :

Number of comparisons	Number of displacements
42	32

Heap Sort step by step

*** A is an ordered integer array with 10 elements from small to large**

A =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

In Heap sort first we put all element of array in an array based max heap, after creating max heap we remove element by one one and put them end of array in heap since that place empty after remove operation.

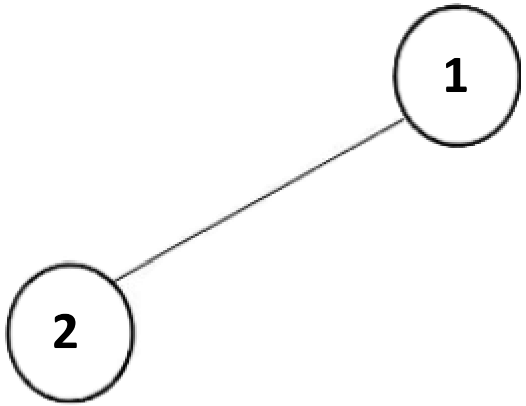
After all elements were removed we get sorted array.

Let's simulate this...

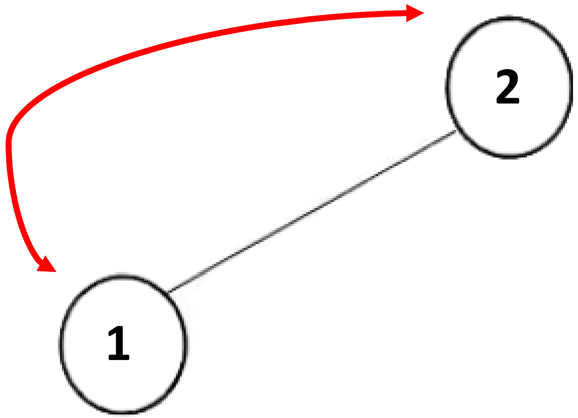
Add 1

1

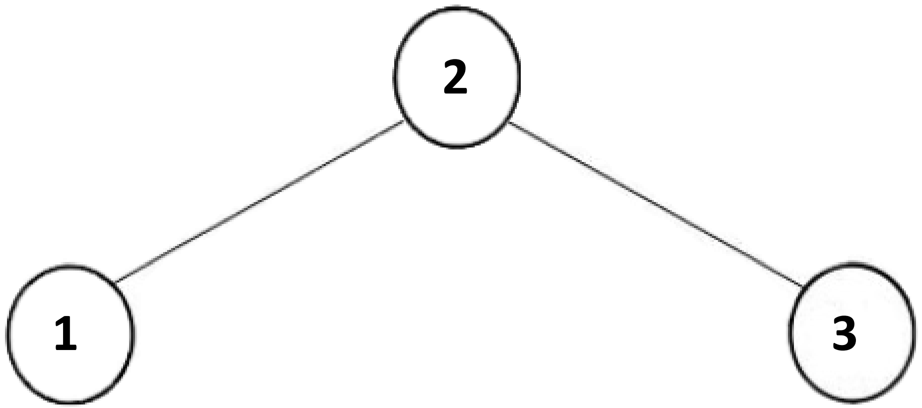
Add 2



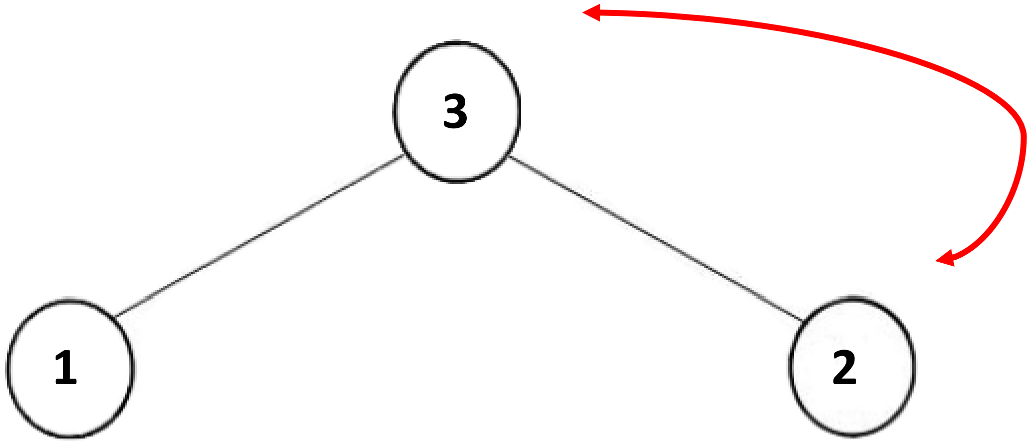
Swap



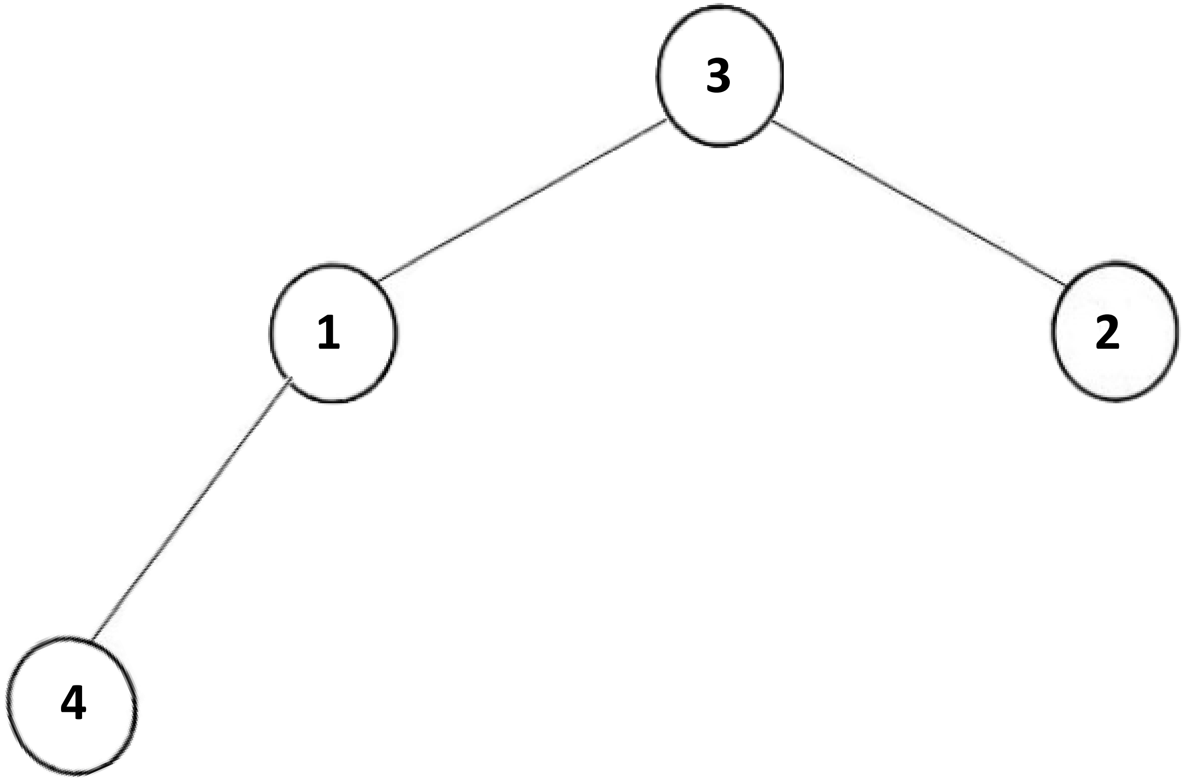
Add 3



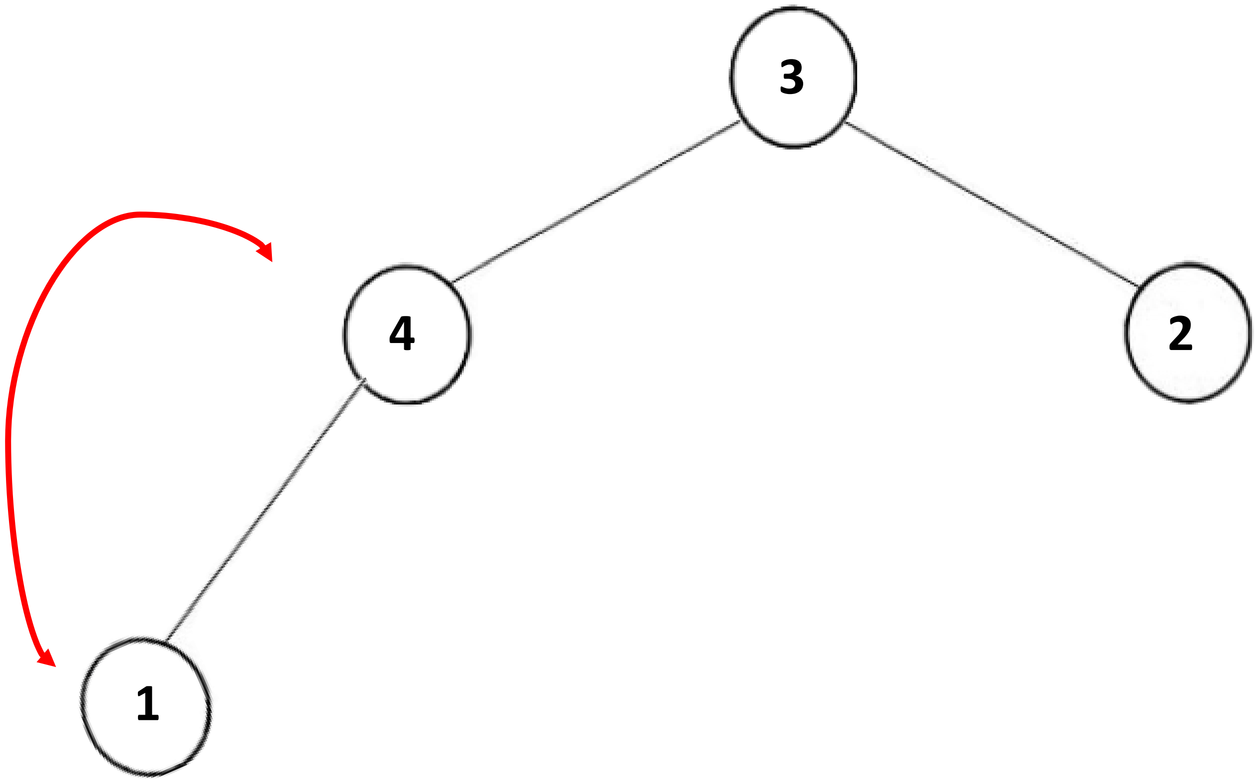
Swap



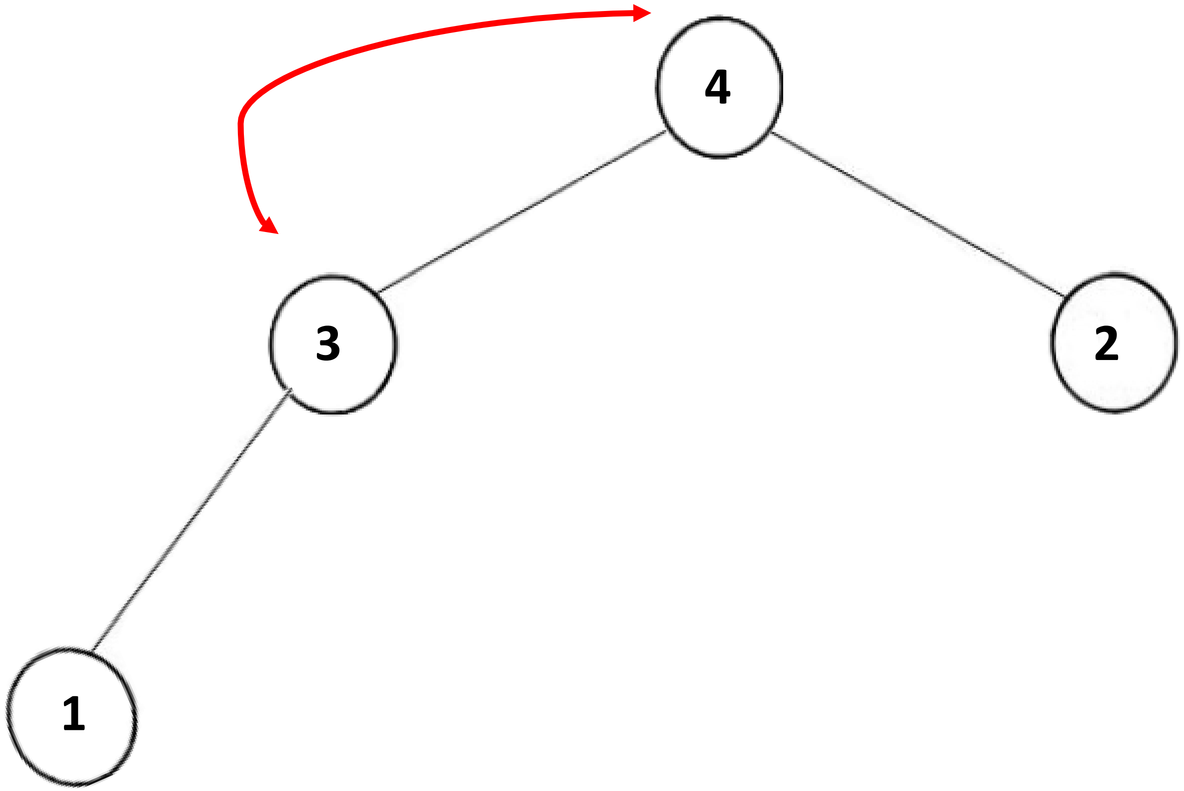
Add 4



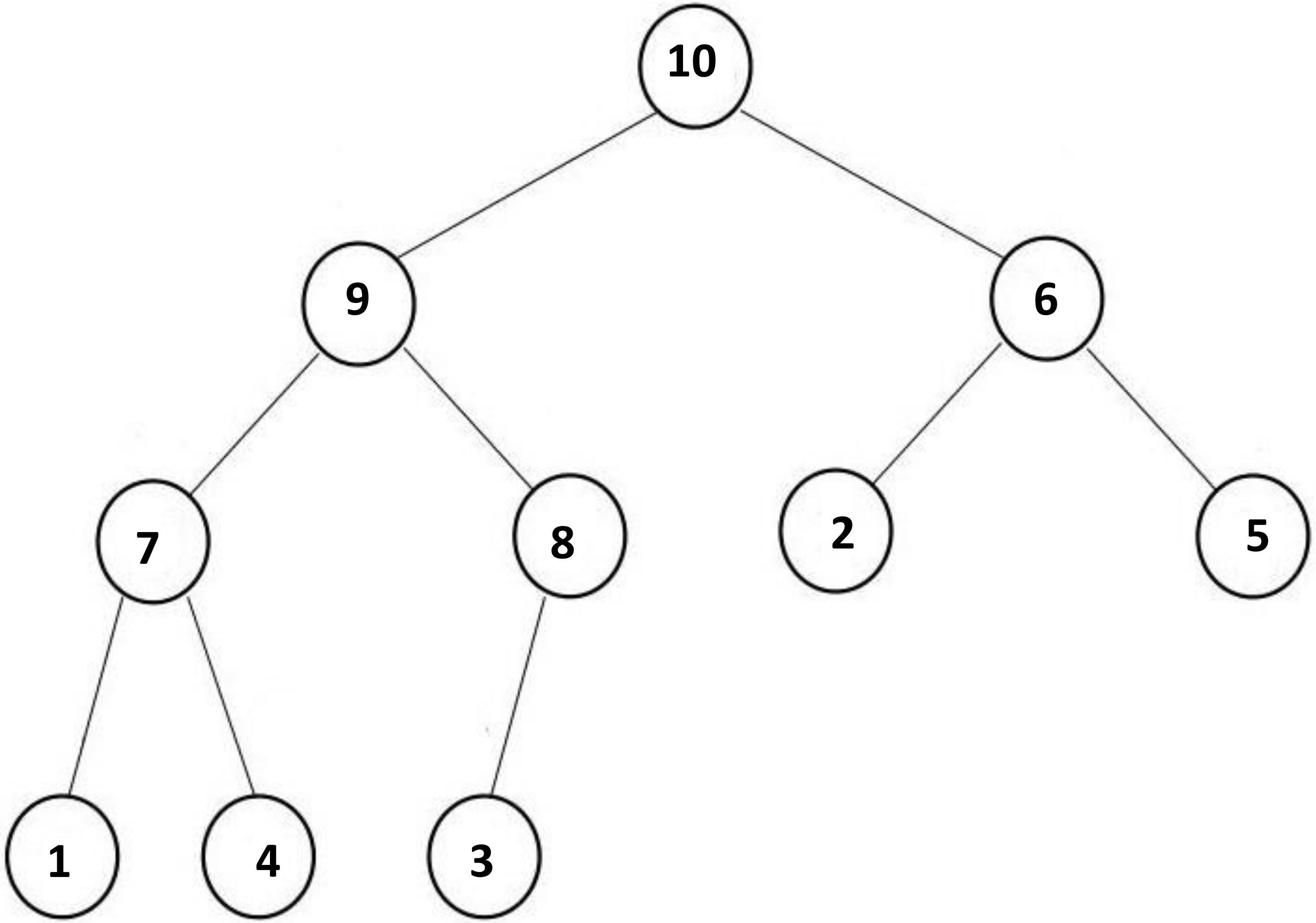
Swap



Swap



This process contiune like this until all array is done...



Heap Array:

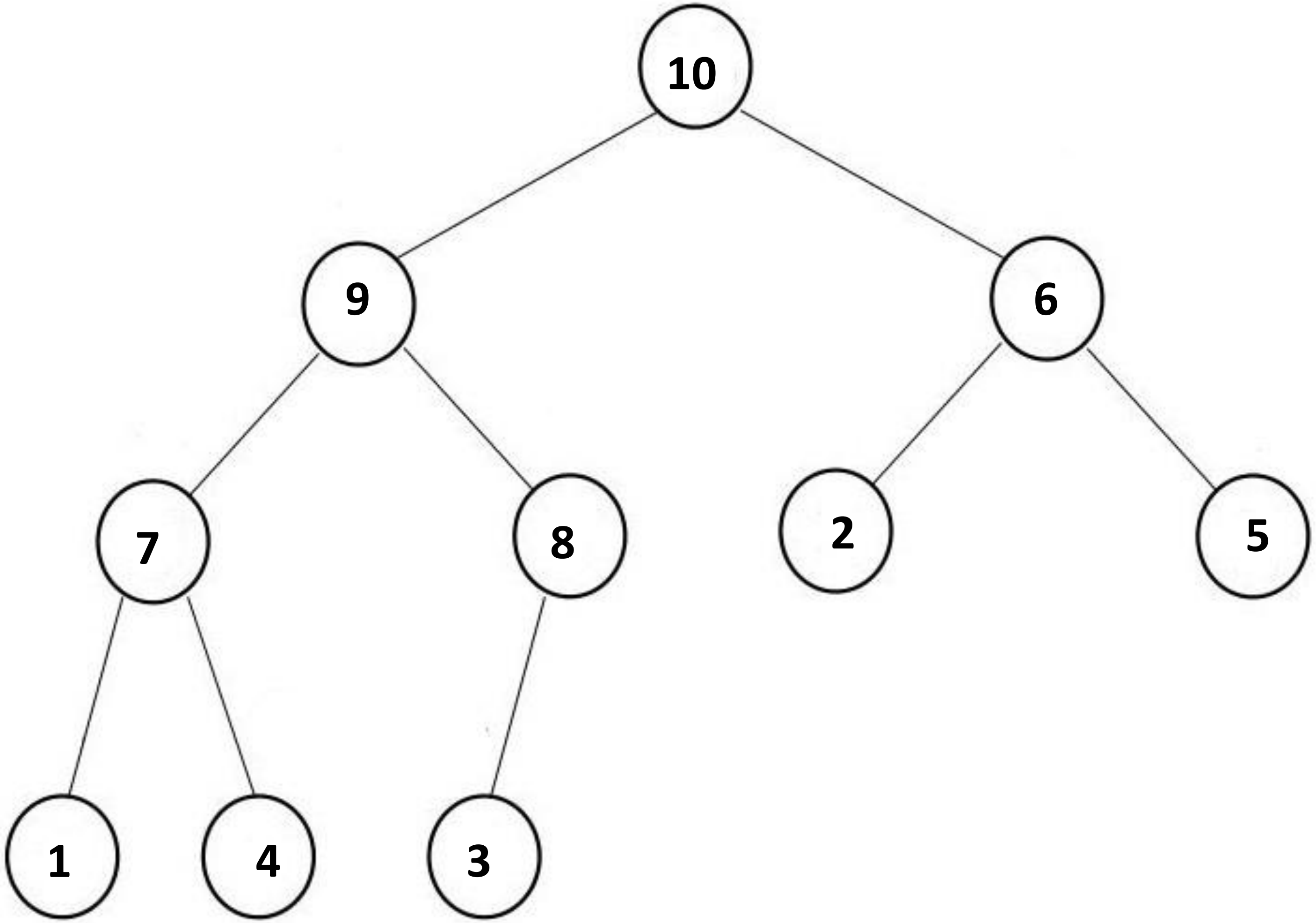
10	9	6	7	8	2	5	1	4	3

Now we remove root element from heap array by one by and put removed element end of the array.

To remove root element we swap it with last element of heap(array) and we continue swapping last element to adjust max heap property then since last place of array(heap) is empty we can put removed root element to that place.

Removed elements denoted red color.

Remove 10

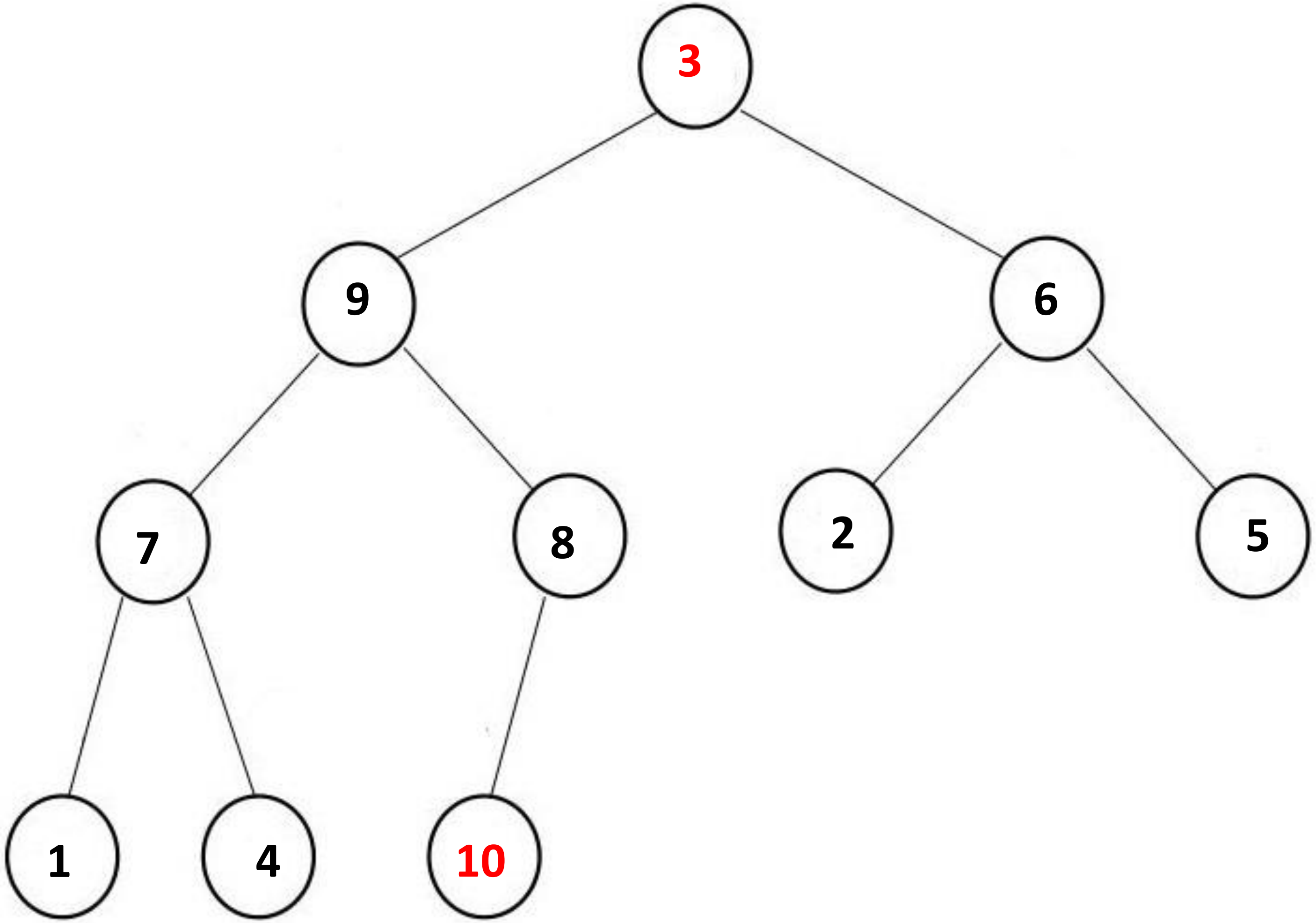


Heap Array:

10	9	6	7	8	2	5	1	4	3

Removed elements denoted red color.

Swap

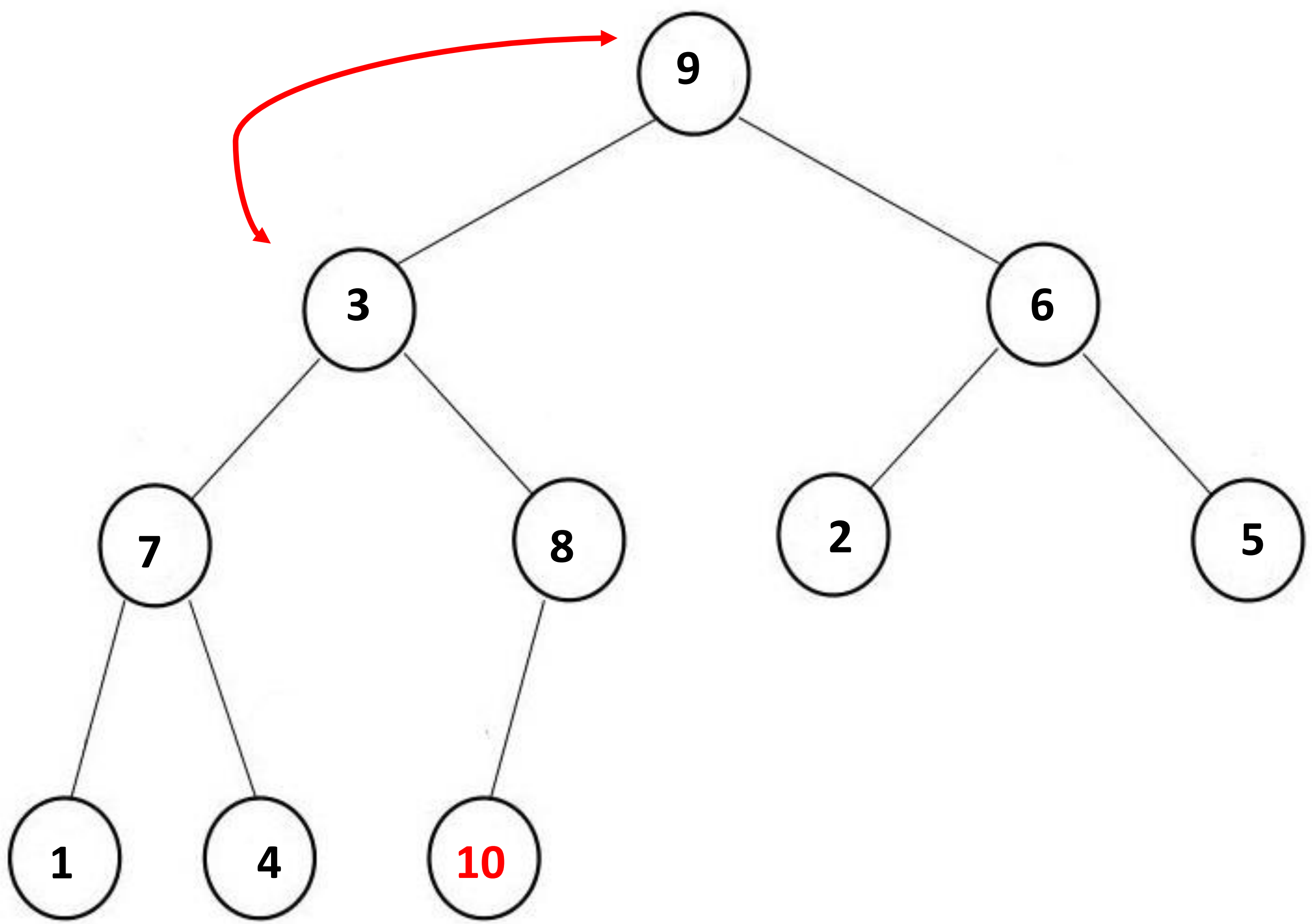


Heap Array:

3	9	6	7	8	2	5	1	4	10

Removed elements
denoted red color.

Swap

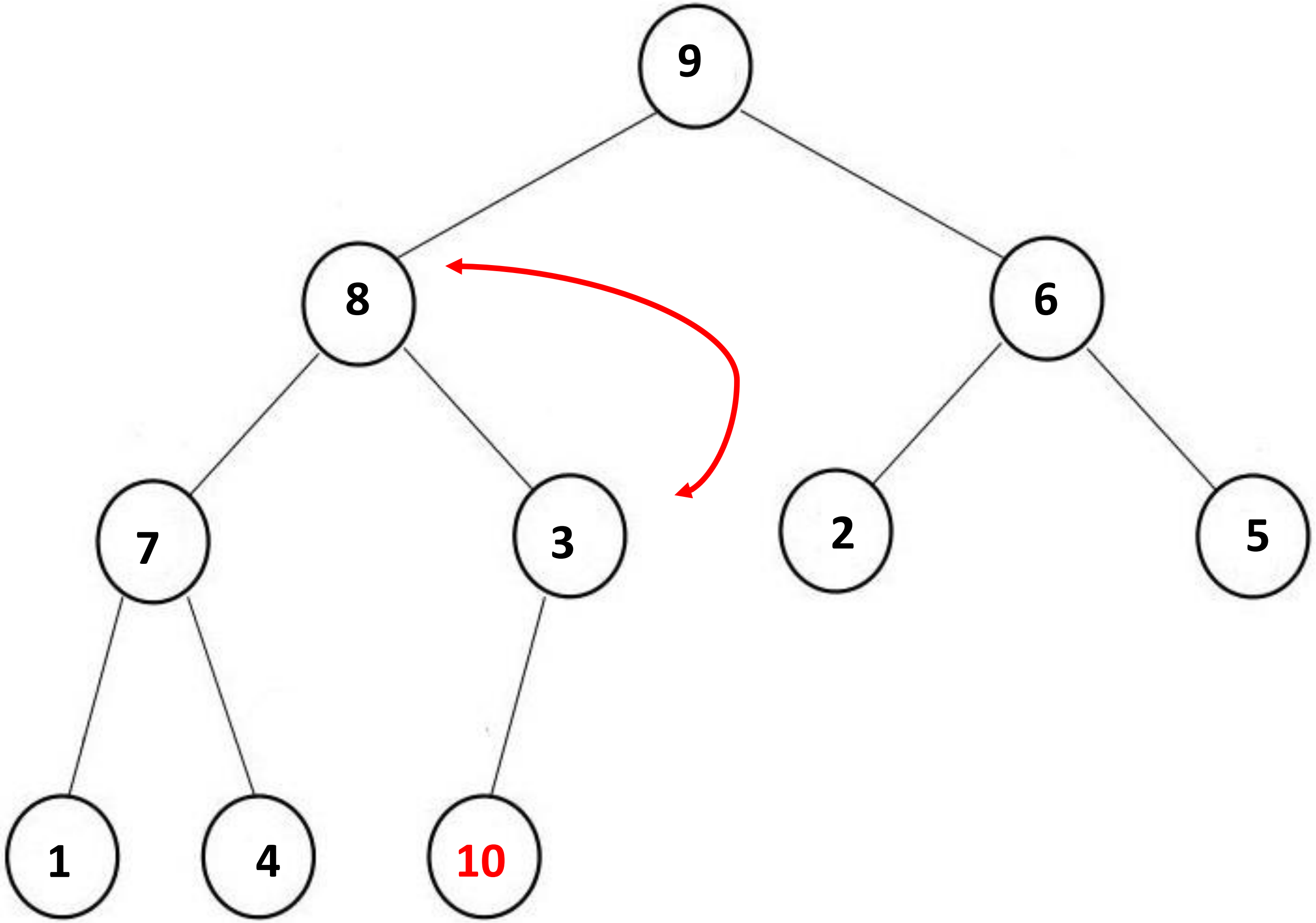


Heap Array:

9	3	6	7	8	2	5	1	4	10

Removed elements denoted red color.

Swap

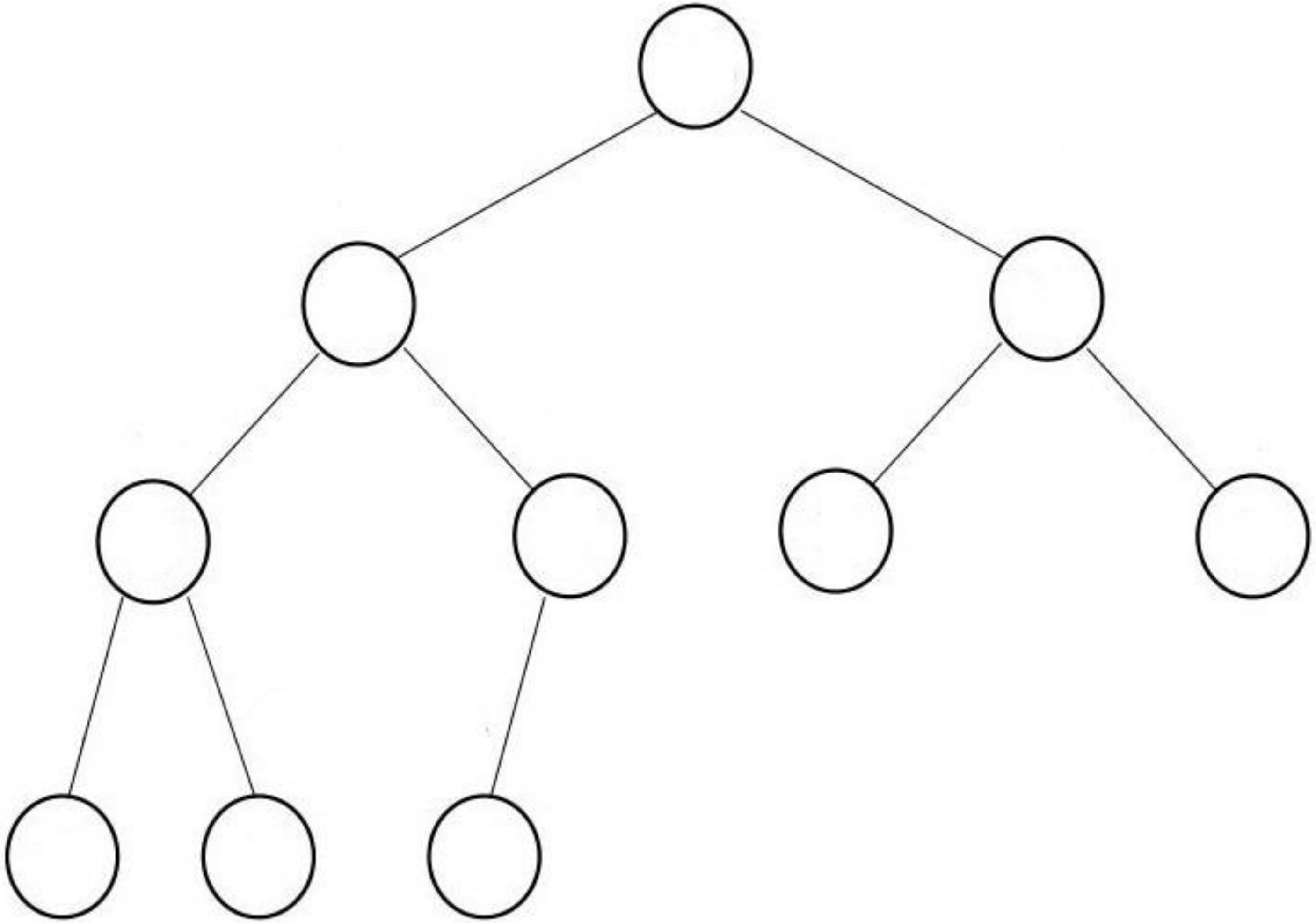


Heap Array:

9	8	6	7	3	2	5	1	4	10

This process contiune like this until all heap is done...

Removed elements denoted red color.



Heap Array:

1	2	3	4	5	6	7	8	9	10

Heap array is our sorted array.

1	2	3	4	5	6	7	8	9	10

Analysis (in heap array):

Number of comparisons	Number of displacements
28	20

* B is an ordered integer array with 10 elements from large to small

B =

10	9	8	7	6	5	4	3	2	1

In Heap sort first we put all element of array in an array based max heap, after creating max heap we remove element by one one and put them end of array in heap since that place empty after remove operation.

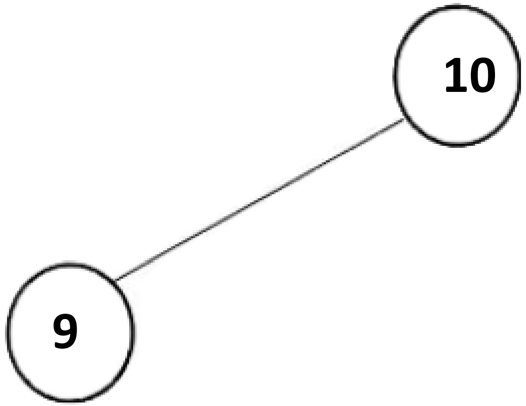
After all elements were removed we get sorted array.

Let’s simulate this...

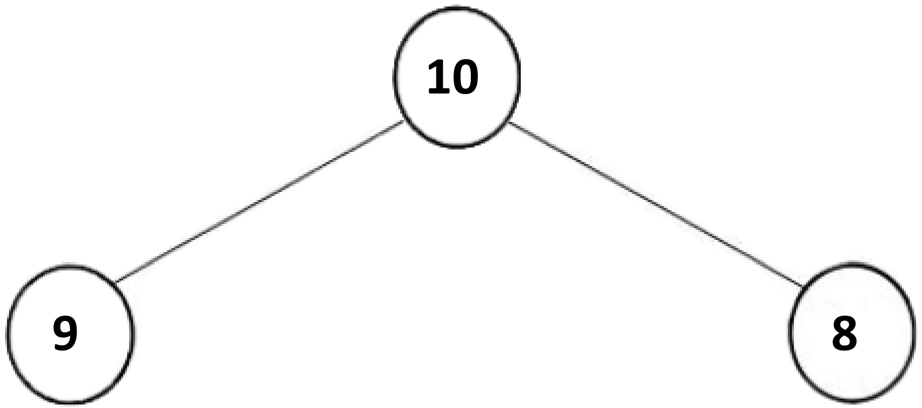
Add 10

10

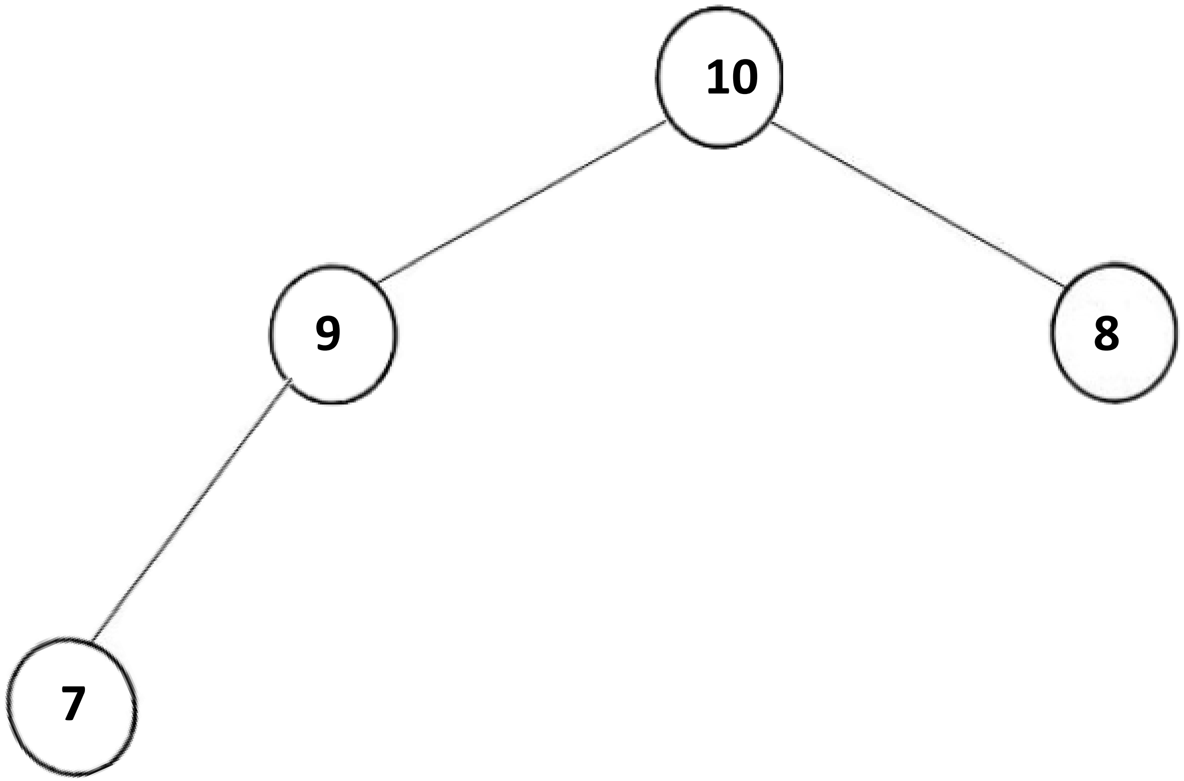
Add 9



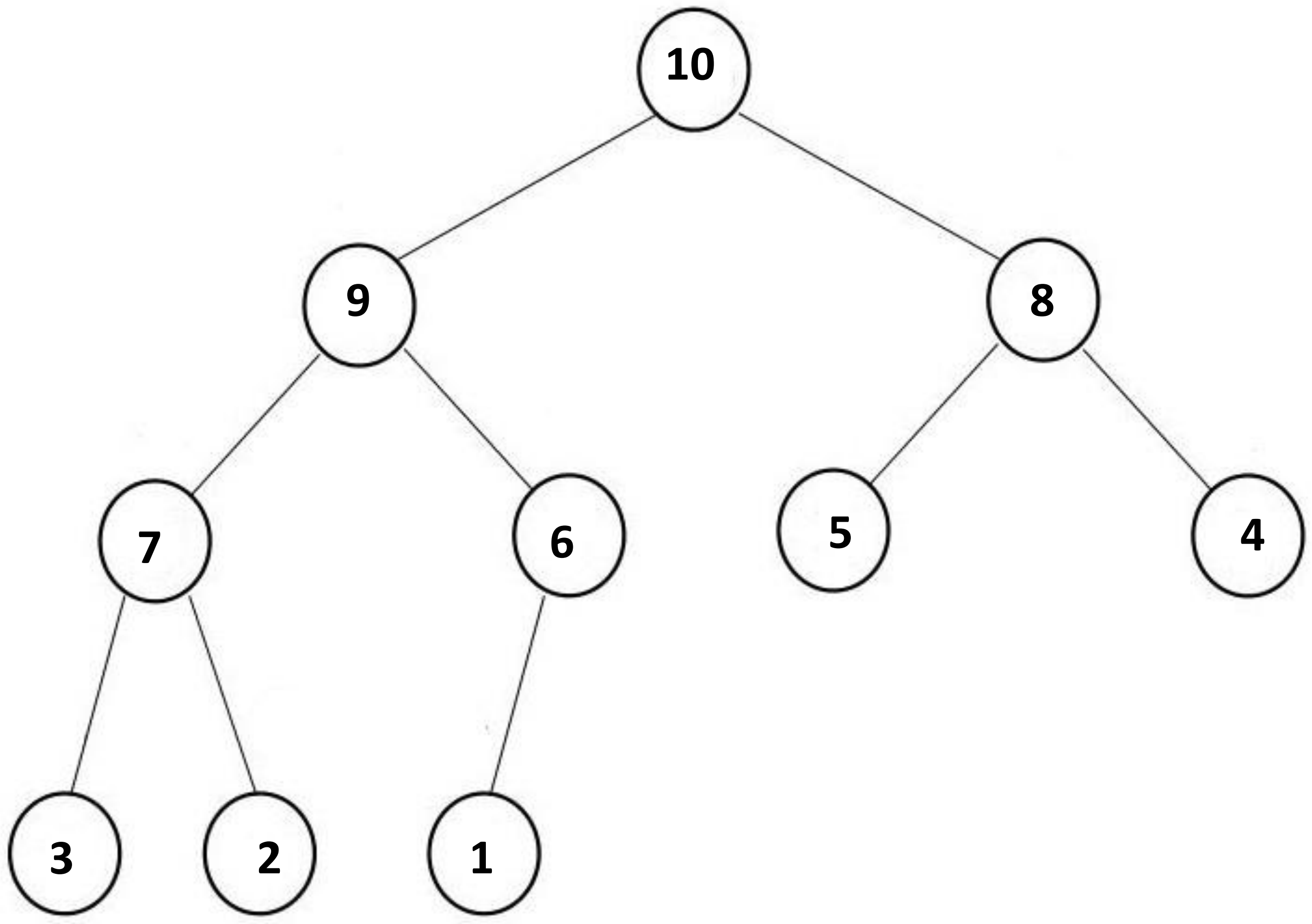
Add 8



Add 7



This process contiune like this until all array is done...



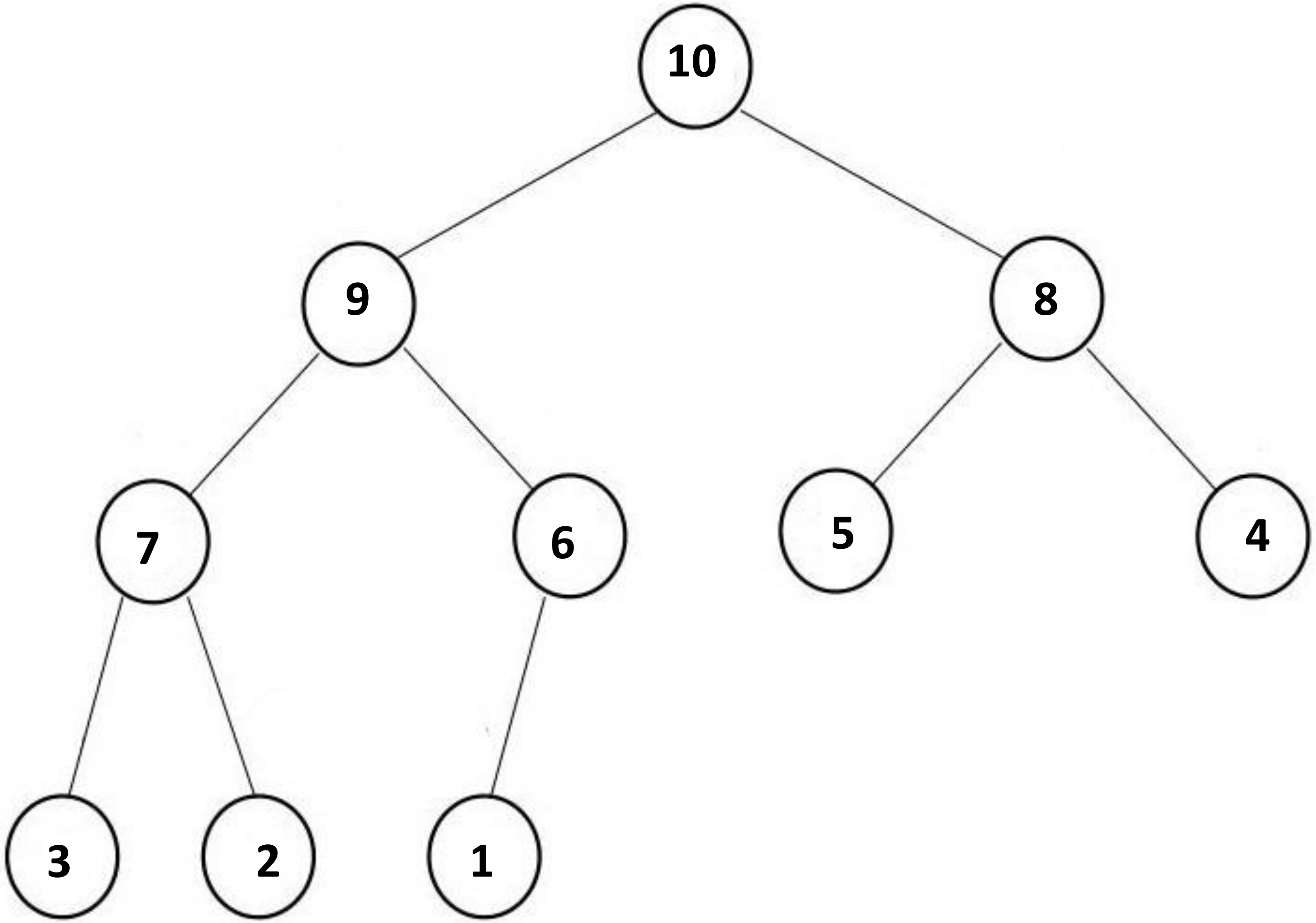
Heap Array:	10	9	8	7	6	5	4	3	2	1

Now we remove root element from heap array by one by and put removed element end of the array.

To remove root element we swap it with last element of heap(array) and we continue swapping last element to adjust max heap property then since last place of array(heap) is empty we can put removed root element to that place.

Removed elements denoted red color.

Remove 10

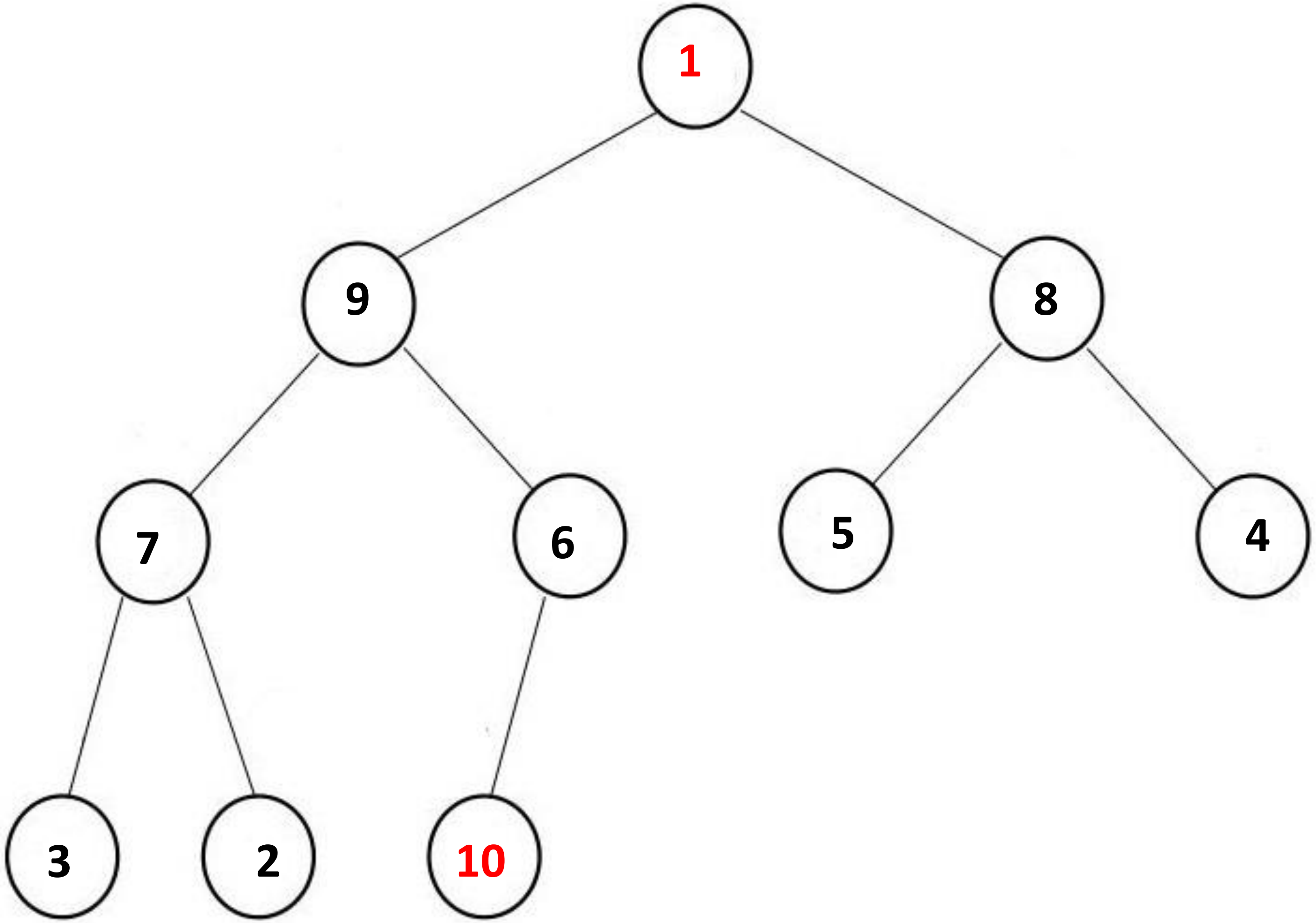


Heap Array:

10	9	8	7	6	5	4	3	2	1

Removed elements denoted red color.

Swap

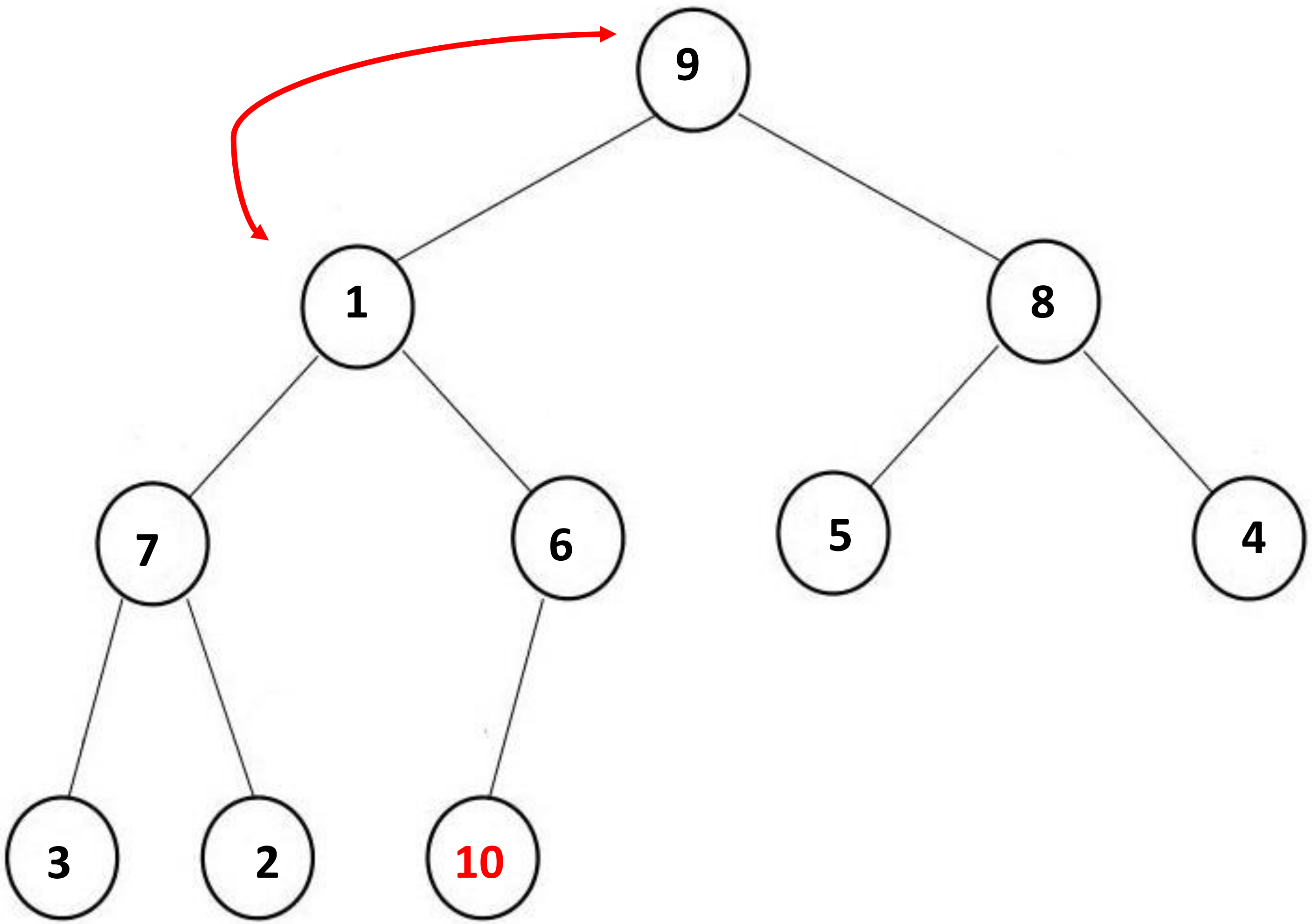


Heap Array:

1	9	8	7	6	5	4	3	2	10

Removed elements denoted red color.

Swap

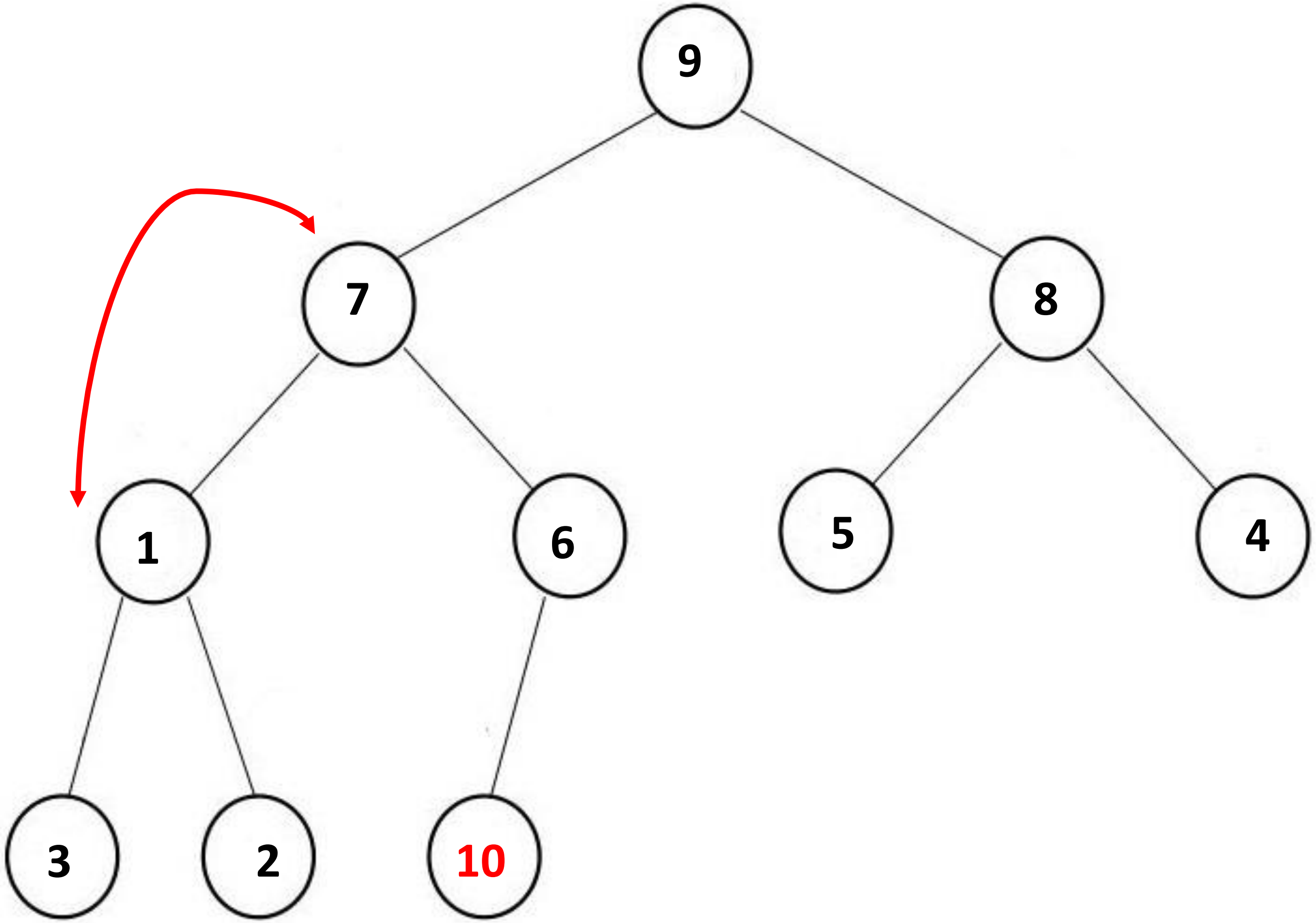


Heap Array:

9	1	8	7	6	5	4	3	2	10

Removed elements denoted red color.

Swap

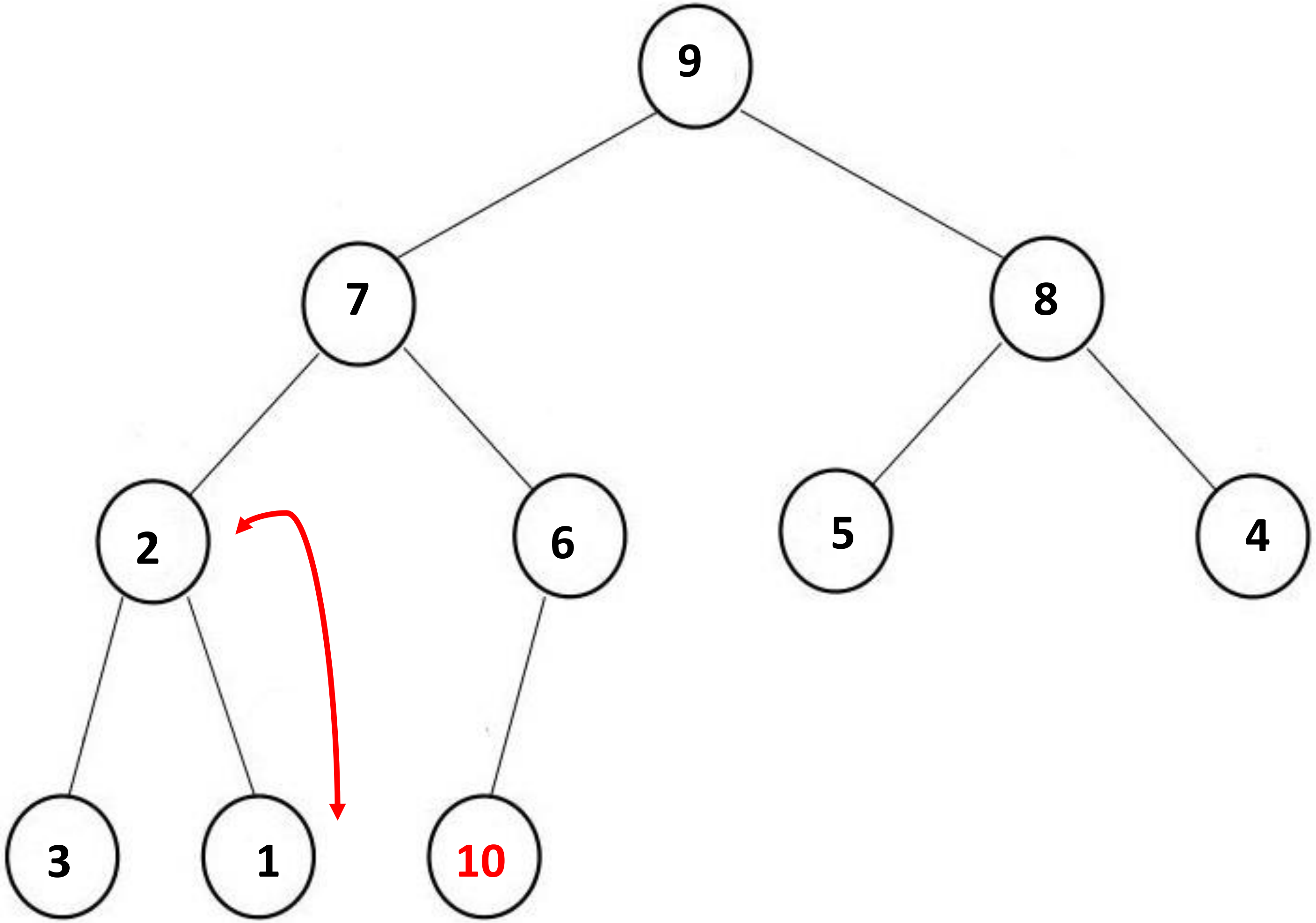


Heap Array:

9	7	8	1	6	5	4	3	2	10

Removed elements denoted red color.

Swap

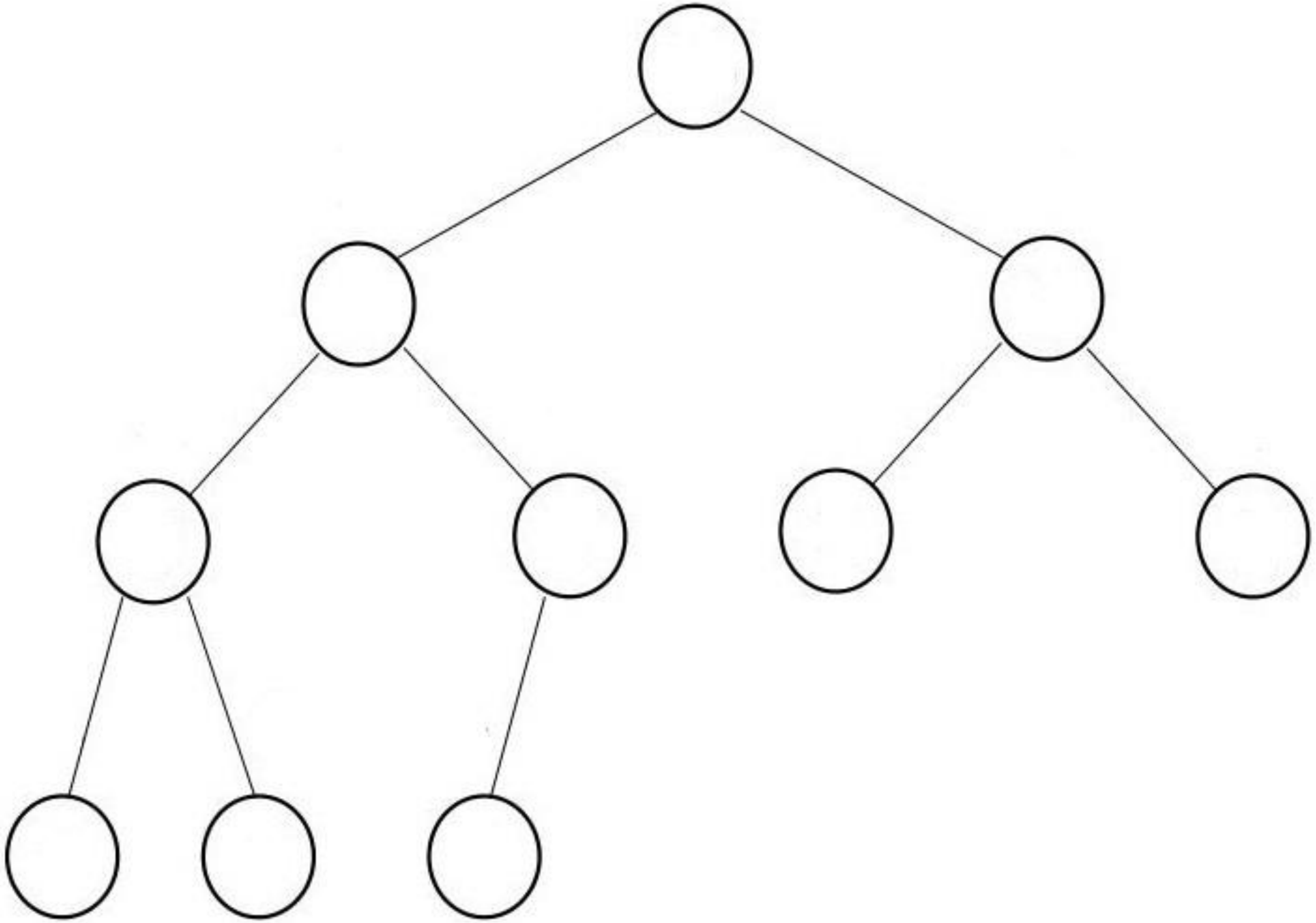


Heap Array:

9	7	8	2	6	5	4	3	1	10

This process contiune like this until all heap is done...

Removed elements denoted red color.



Heap Array:

1	2	3	4	5	6	7	8	9	10

Heap array is our sorted array.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Analysis(in heap array) :

Number of comparisons	Number of displacements
28	22

* C = {5, 2, 13, 9, 1, 7, 6, 8, 1, 15, 4, 11}

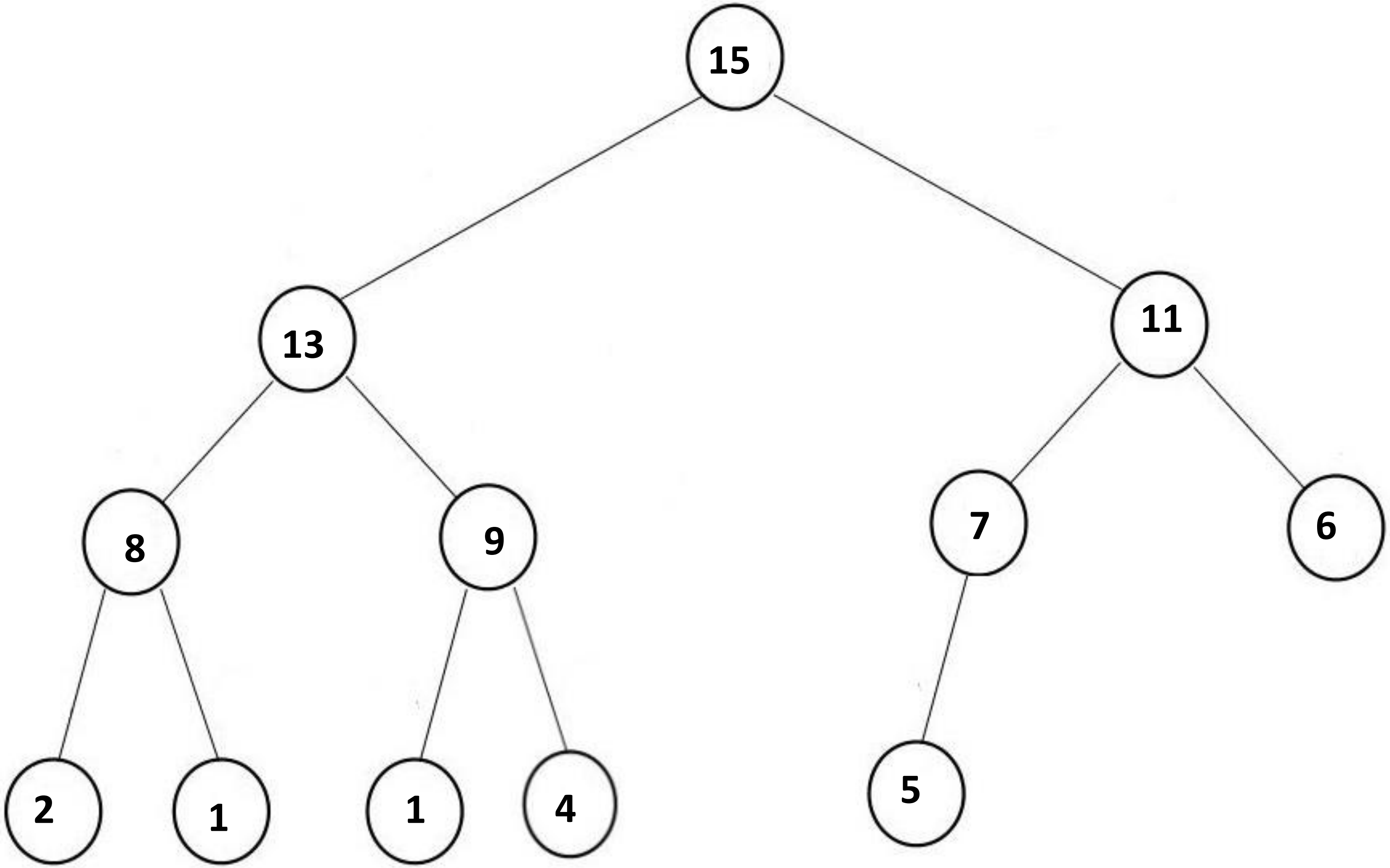
C =

5	2	13	9	1	7	6	8	1	15	4	11

In Heap sort first we put all element of array in an array based max heap, after creating max heap we remove element by one one and put them end of array in heap since that place empty after remove operation.

After all elements were removed we get sorted array.

Let’s apply same process...

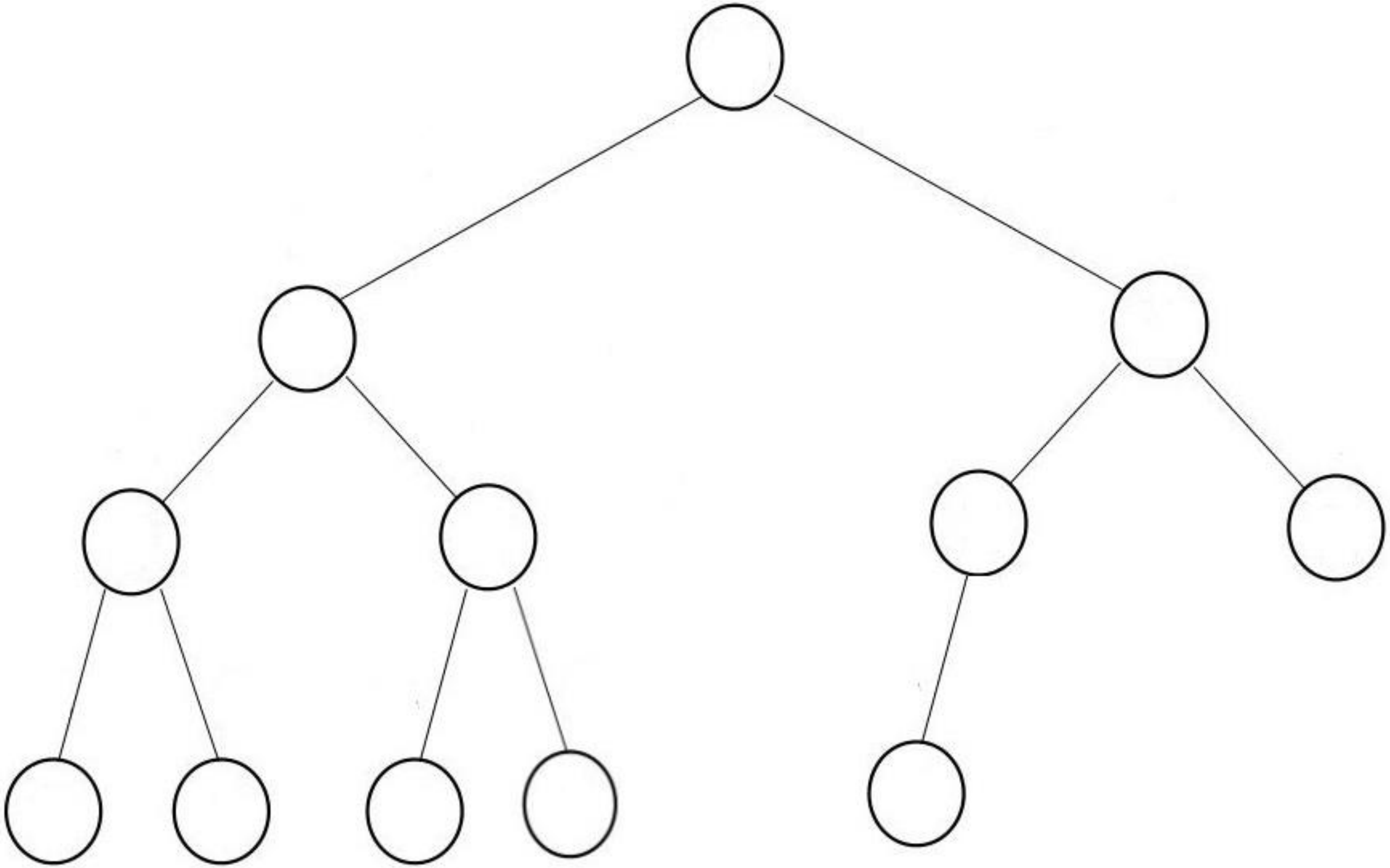


Heap Array:

15	13	11	8	9	7	6	2	1	1	4	5

Now we remove root element from heap array by one by and put removed element end of the array.

To remove root element we swap it with last element of heap(array) and we continue swapping last element to adjust max heap property then since last place of array(heap) is empty we can put removed root element to that place.



Heap Array:

1	1	2	4	5	6	7	8	9	11	13	15

Heap array is our sorted array.

C =

1	**1**	**2**	**4**	**5**	**6**	**7**	**8**	**9**	**11**	**13**	**15**

<u>Analysis(in heap array) :</u>

Number of comparisons	Number of displacements
38	29

* D = {‘S’, ‘B’, ‘I’, ‘M’, ‘H’, ‘Q’, ‘C’, ‘L’, ‘R’, ‘E’, ‘P’, ‘K’}

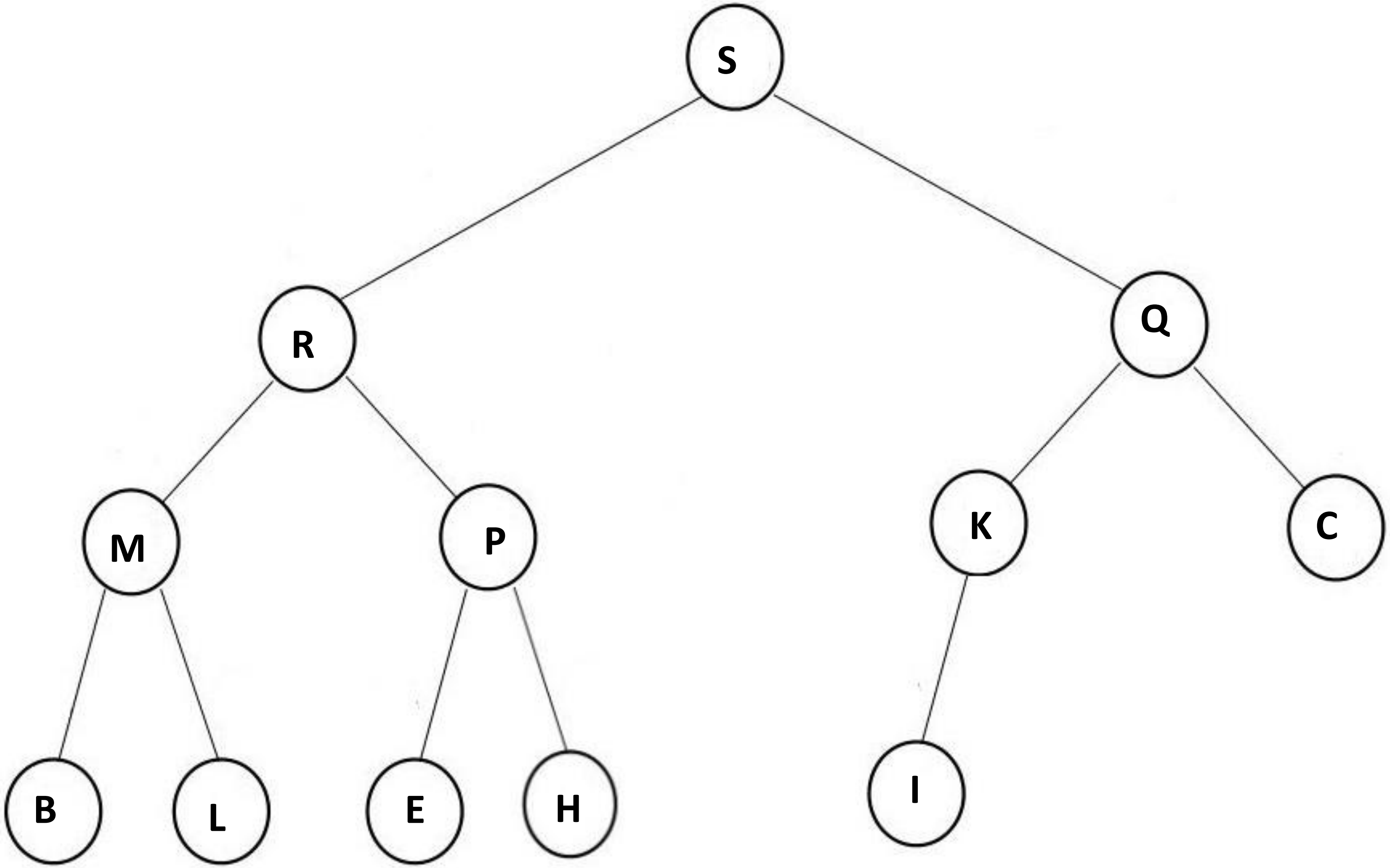
D =

S	B	I	M	H	Q	C	L	R	E	P	K

In Heap sort first we put all element of array in an array based max heap, after creating max heap we remove element by one one and put them end of array in heap since that place empty after remove operation.

After all elements were removed we get sorted array.

Let's apply same process...

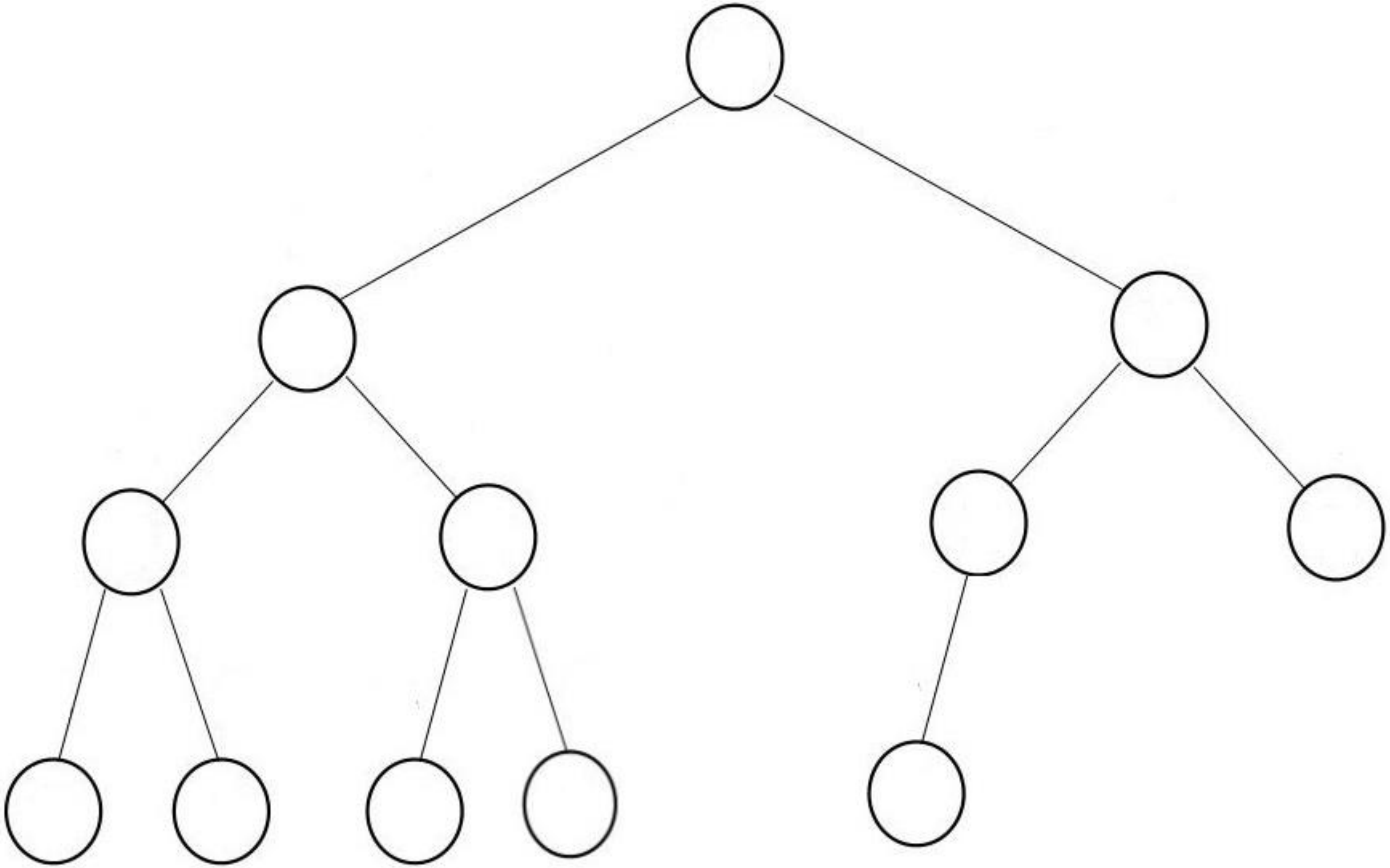


Heap Array:

S	R	Q	M	P	K	C	B	L	E	H	I

Now we remove root element from heap array by one by and put removed element end of the array.

To remove root element we swap it with last element of heap(array) and we continue swapping last element to adjust max heap property then since last place of array(heap) is empty we can put removed root element to that place.



Heap Array:

B	C	E	H	I	K	L	M	P	Q	R	S

Heap array is our sorted array.

D =

B	C	E	H	I	K	L	M	P	Q	R	S

<u>Analysis(in heap array) :</u>

Number of comparisons	Number of displacements
40	30

Quick Sort step by step

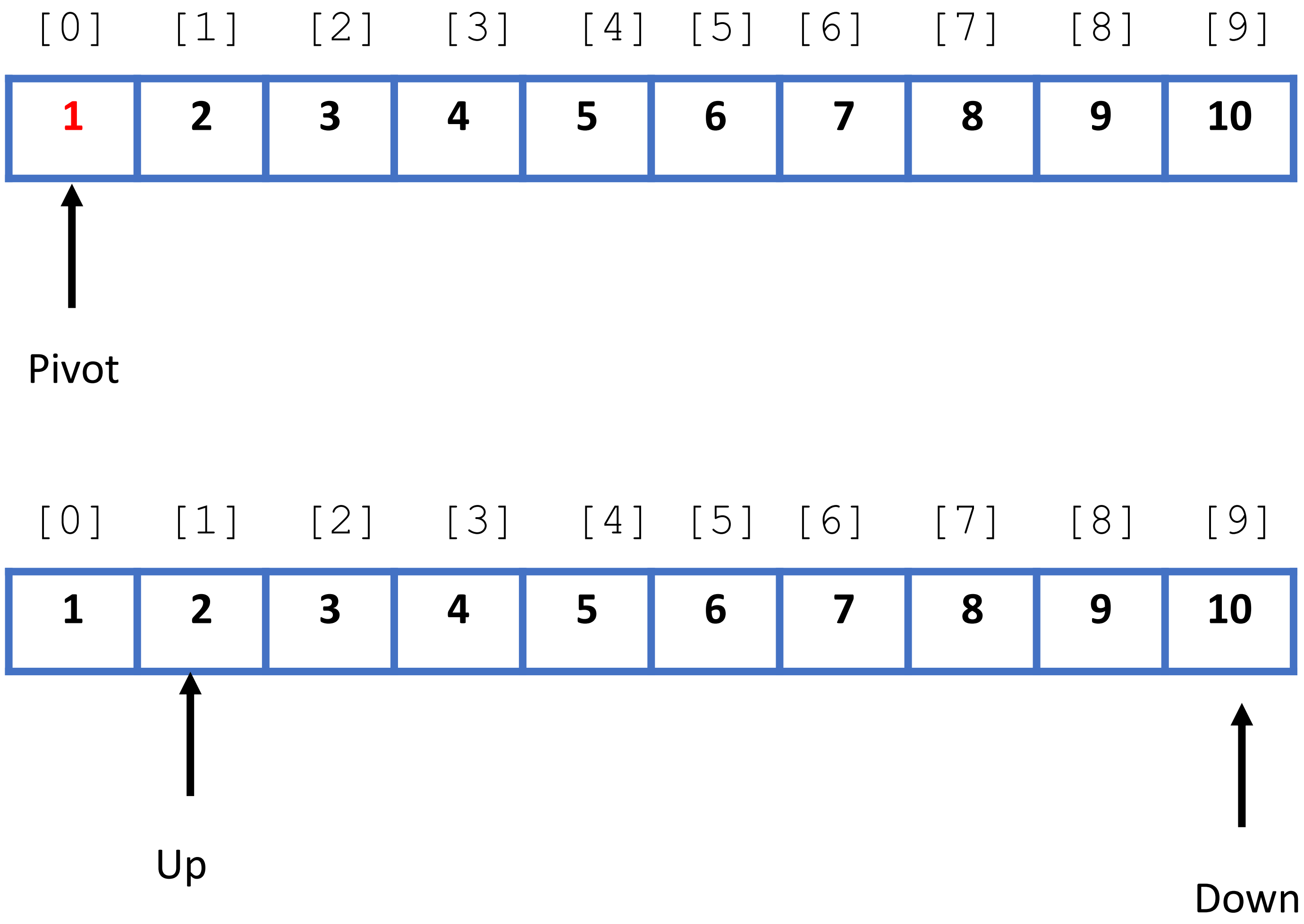
* A is an ordered integer array with 10 elements from small to large

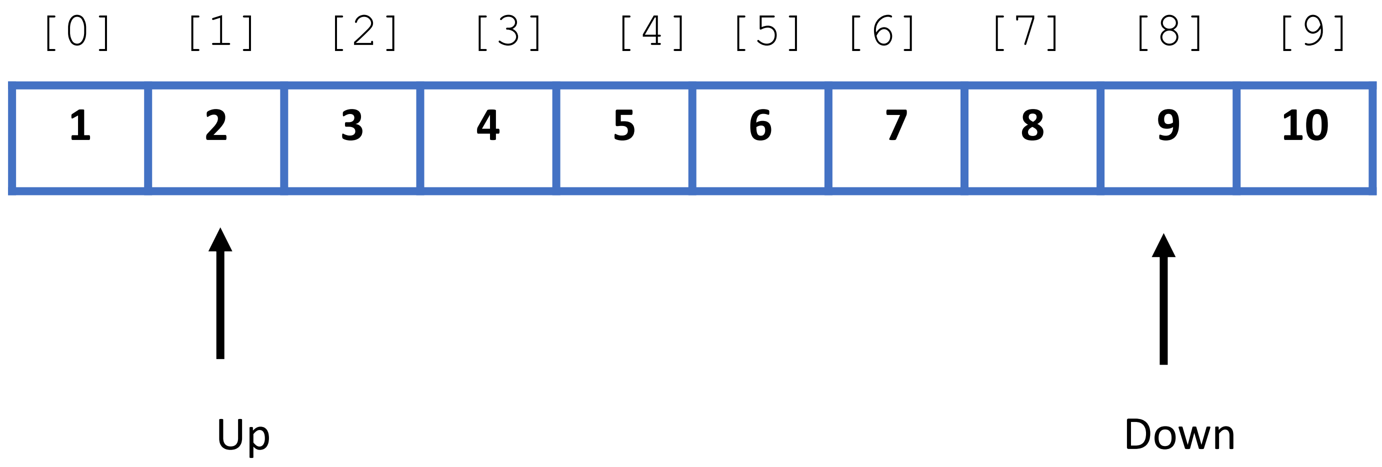
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
A =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

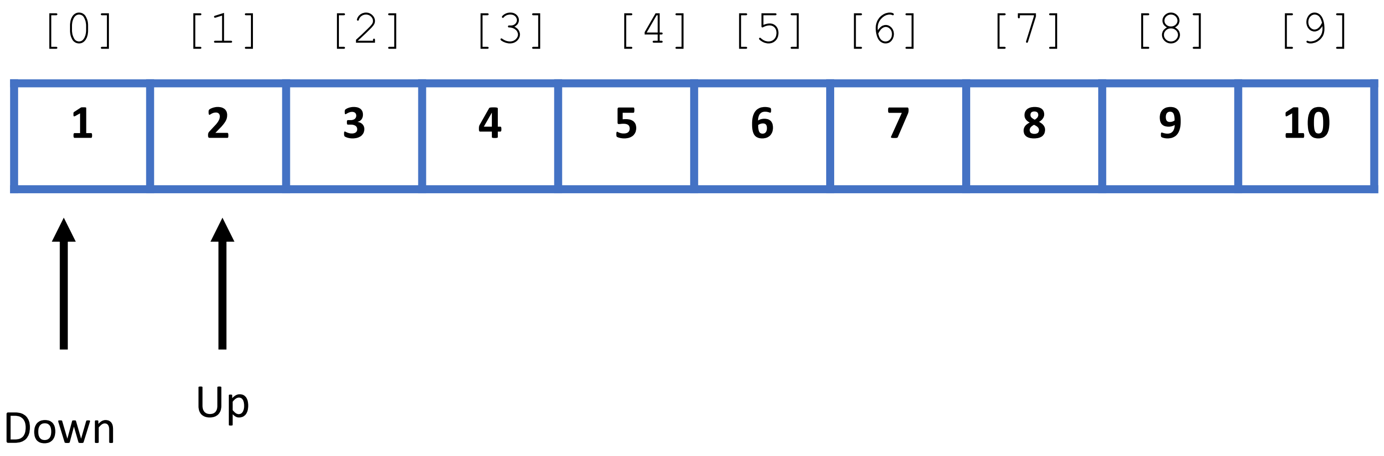
Like Merge Sort, **QuickSort** is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways. We pick first element as pivot as book did.

In partition part of algorithm we keep two value these are **up** and **down.** In algorithm we go upward by using up and downward while going if the up value is **bigger** than pivot value and down value **less** than pivot value we swap them.This process contuine until down and uo watches **if they are in same place** then if down value **less** than pivot ,we swap the pivot value in that place and return that index value.

Let’s simulate the partition part...


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
1 2 3 4 5 6 7 8 9 10
Pivot
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
1 2 3 4 5 6 7 8 9 10
Up
Down


[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
1
2
3
4
5
6
7
8
9
10
Up
Down



Down passes the up so Exchange table[first] and table[down] thus putting the pivot value where it belongs.
We contiune this process by dividing array from pivot and for each part.

Divide array and apply same process(partition)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Pivot

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Pivot

We apply same process until end of the array and we get sorted array but **note that** since our Array already sorted and pivot is first element we get O(n^2) running time.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
A =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

<u>Analysis :</u>

Number of comparisons	**Number of displacements**
72	0

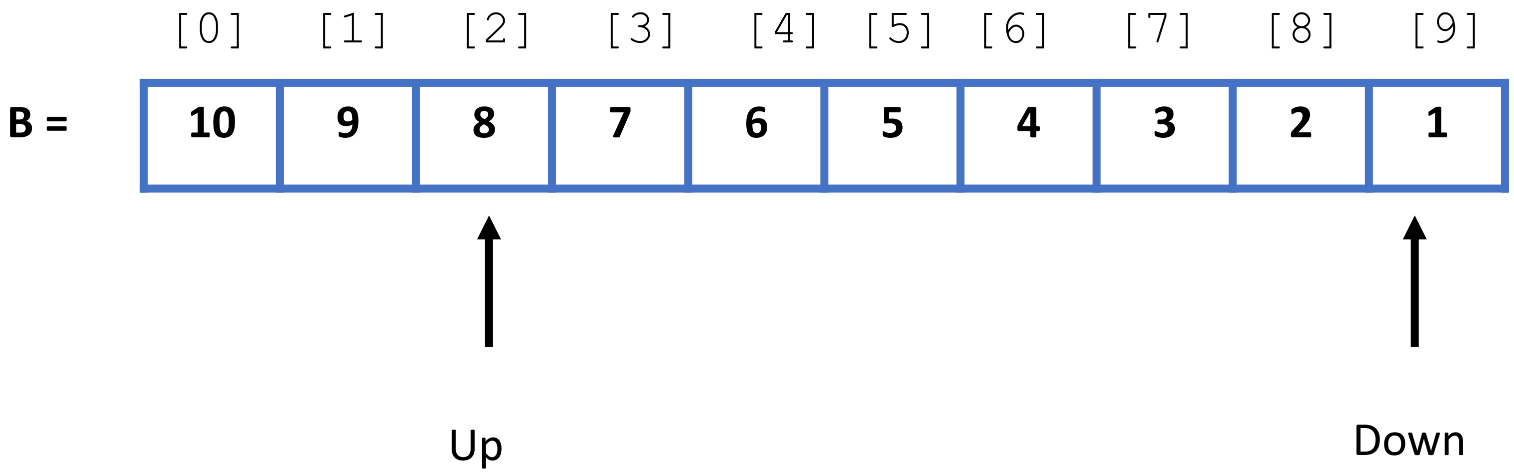
* B is an ordered integer array with 10 elements from large to small

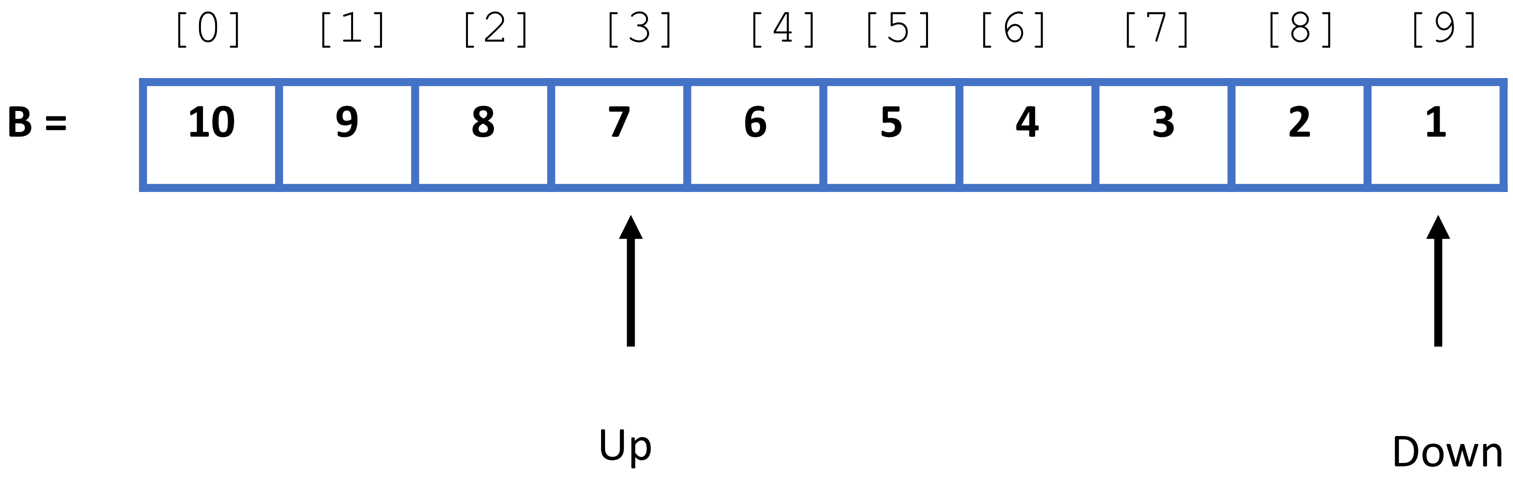
Apply quick sort algorithm that is mentioned before

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
B =	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
	Pivot									

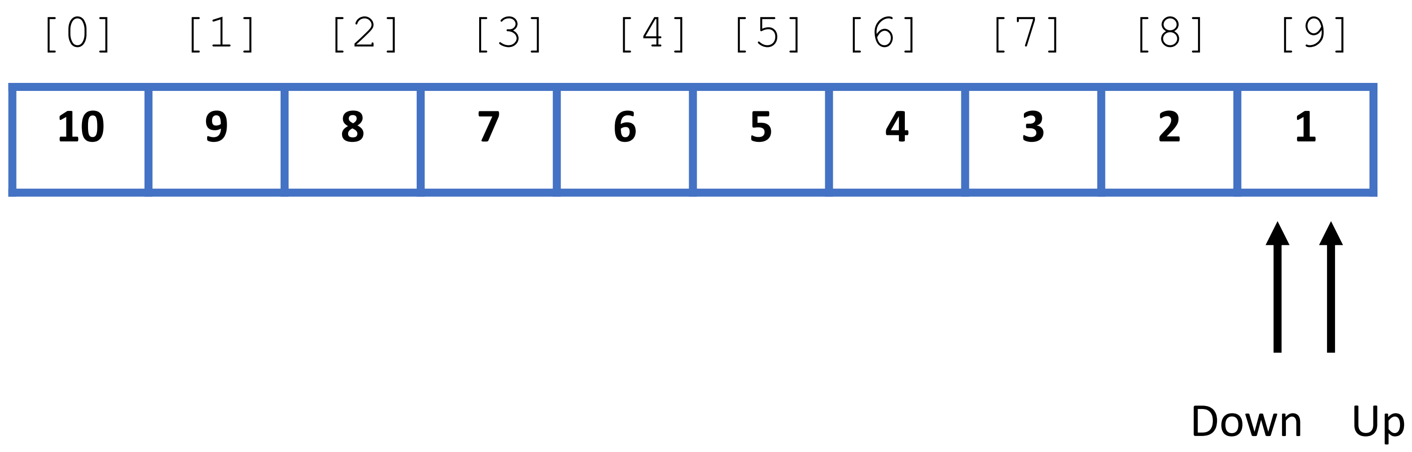
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
B =	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
		Up								Down

Swap them


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
B =
10 9 8 7 6 5 4 3 2 1
Up
Down


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
B =
10 9 8 7 6 5 4 3 2 1
Up
Down

Down passes the up so Exchange table[first] and table[down] thus putting the pivot value where it belongs.
We contiune this process by dividing array from pivot and for each part.



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**10**

Pivot

Divide array and apply same process...(without dividing on same array)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Pivot

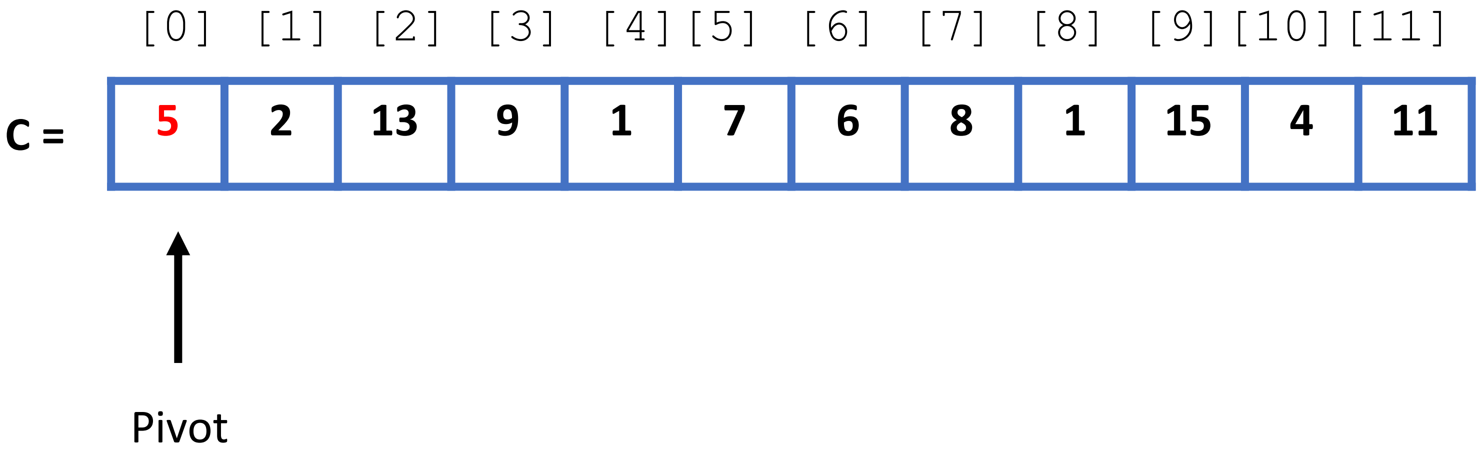
We apply same process until end of the array and we get sorted array but **note that** since our already sorted after first partition and pivot is first element we get O(n^2) running time.

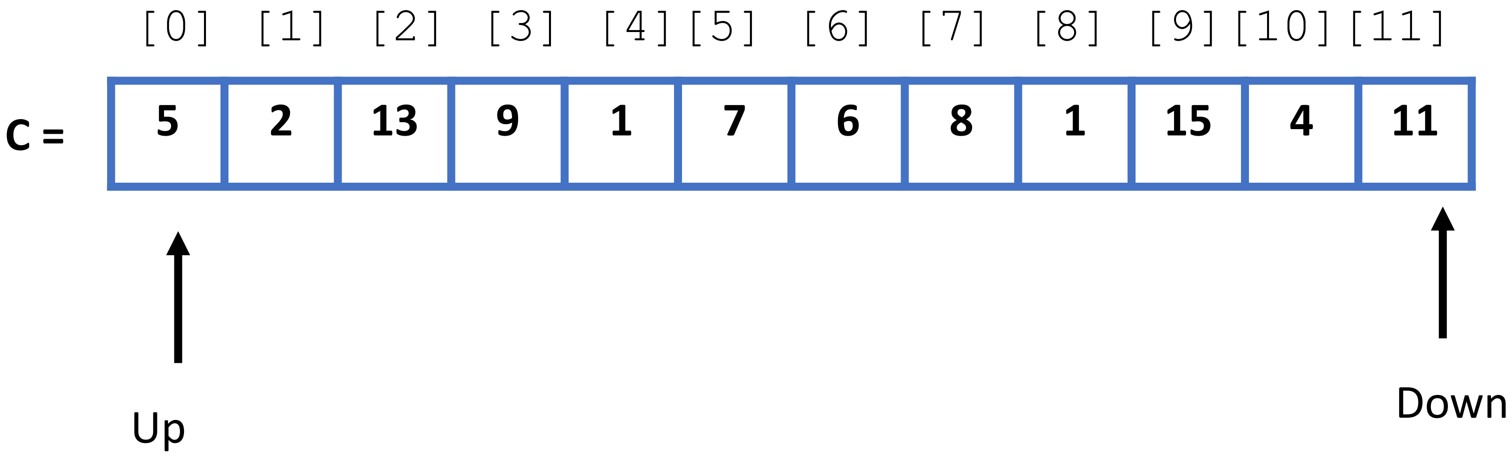
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
B =	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

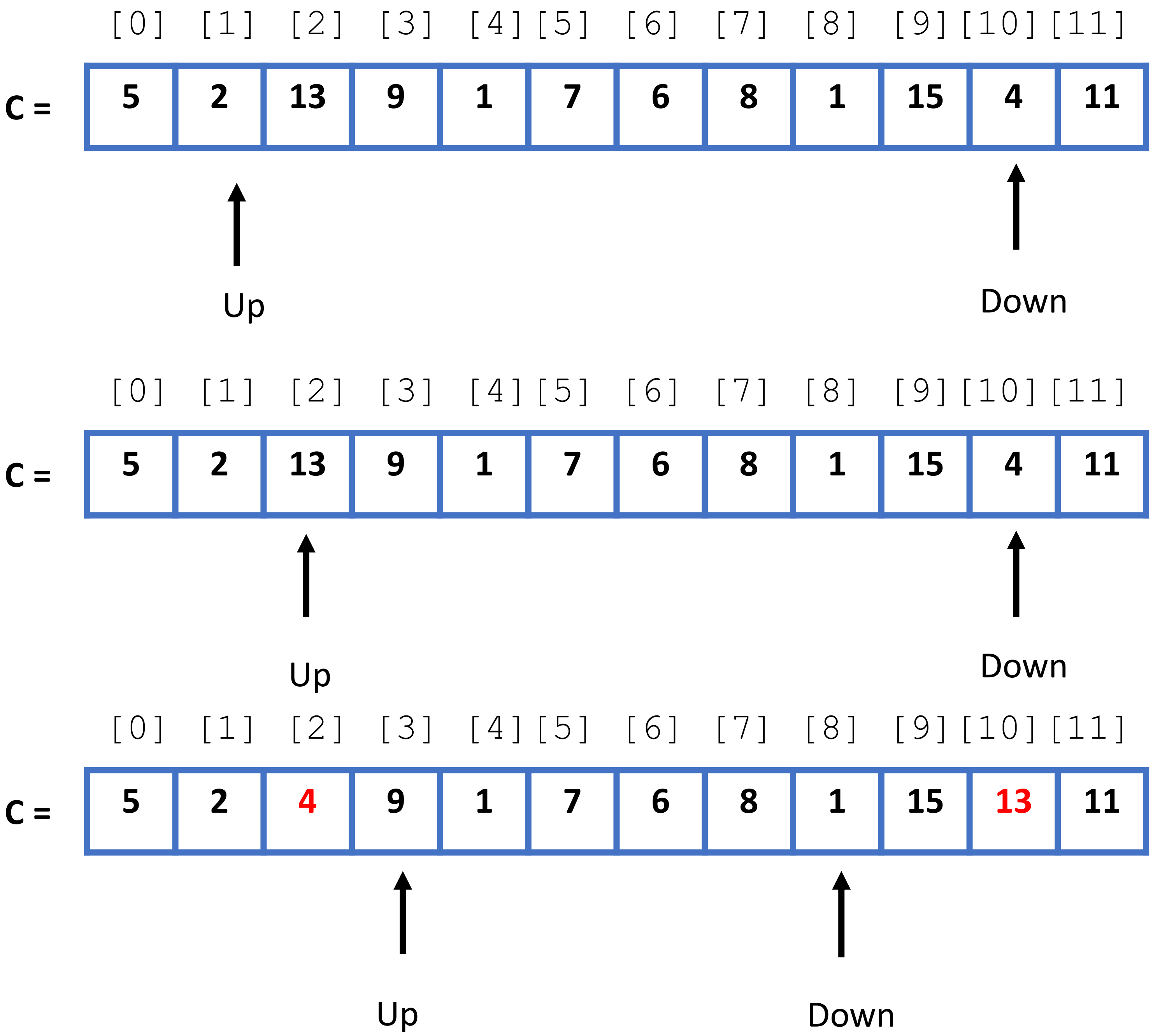
Analysis :

Number of comparisons	Number of displacements
67	5

* $C = \{5, 2, 13, 9, 1, 7, 6, 8, 1, 15, 4, 11\}$


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
C =
5 2 13 9 1 7 6 8 1 15 4 11
Pivot


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
C =
5 2 13 9 1 7 6 8 1 15 4 11
Up
Down



[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
C = 5 2 13 9 1 7 6 8 1 15 4 11
Up
Down
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
C = 5 2 13 9 1 7 6 8 1 15 4 11
Up
Down
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
C = 5 2 4 9 1 7 6 8 1 15 13 11
Up
Down

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	**5**	**2**	**4**	**1**	**1**	**7**	**6**	**8**	**9**	**15**	**13**	**11**
					Up			Down				

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	**5**	**2**	**4**	**1**	**1**	**7**	**6**	**8**	**9**	**15**	**13**	**11**
					Down	Up						

Down passes the up so Exchange table[first] and table[down] thus putting the pivot value where it belongs.
We contiune this process by dividing array from pivot and for each part.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	**1**	**2**	**4**	**1**	**5**	**7**	**6**	**8**	**9**	**15**	**13**	**11**

Pivot

Divide and do same process

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	**1**	**2**	**4**	**1**	**5**	**7**	**6**	**8**	**9**	**15**	**13**	**11**

Pivot

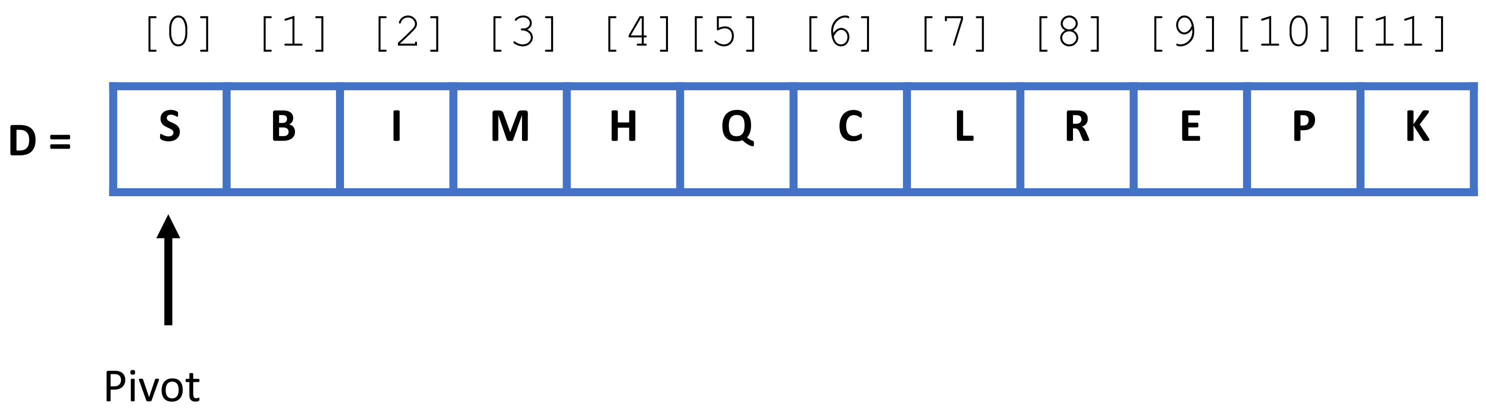
Pivot

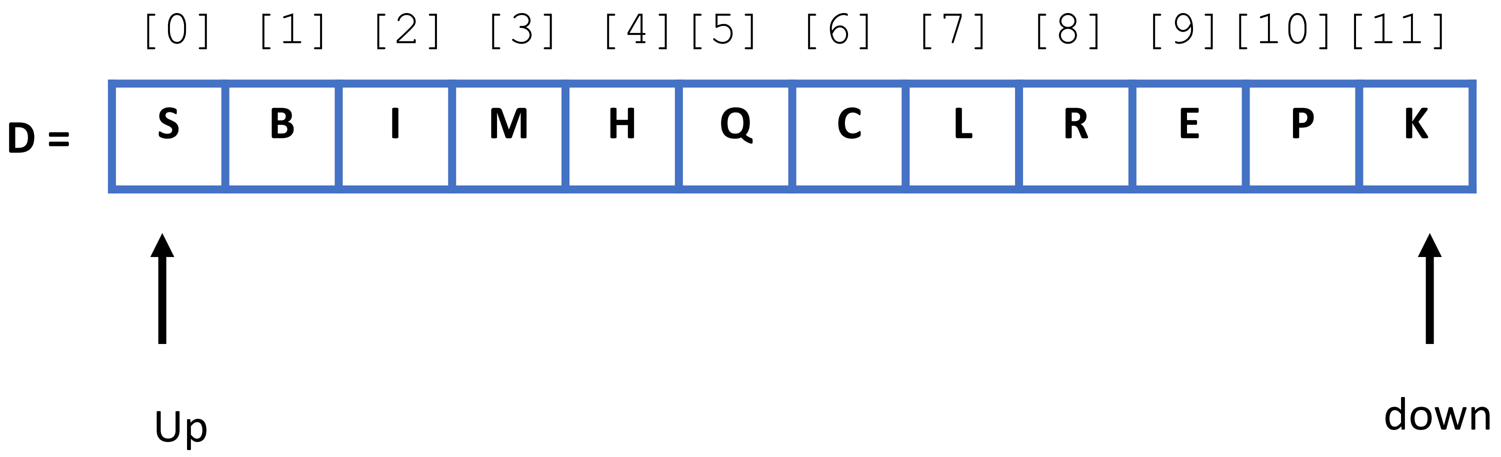
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C =	**1**	**1**	**2**	**4**	**5**	**6**	**7**	**8**	**9**	**11**	**13**	**15**

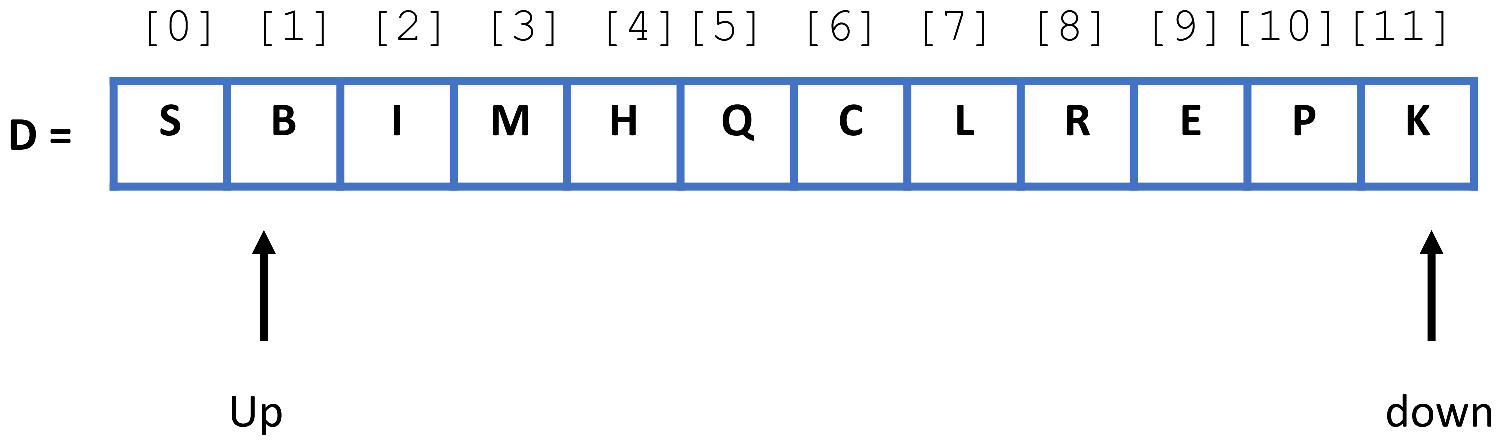
Analysis :

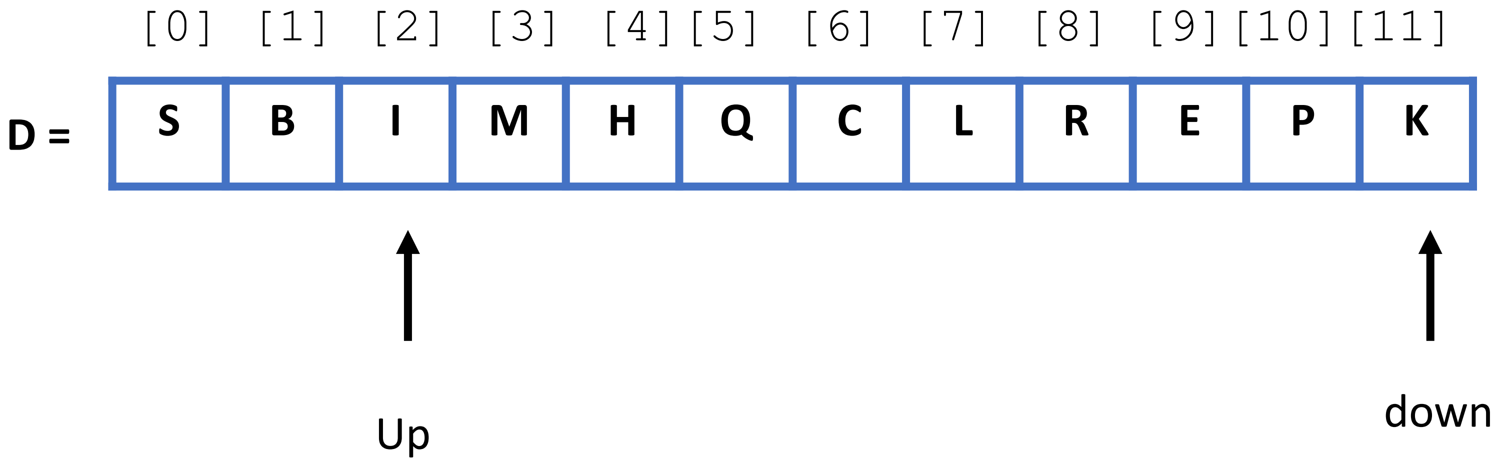
Number of comparisons	**Number of displacements**
59	8

* D = {‘S’, ‘B’, ‘I’, ‘M’, ‘H’, ‘Q’, ‘C’, ‘L’, ‘R’, ‘E’, ‘P’, ‘K’}


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
D =
S B I M H Q C L R E P K
Pivot


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
D =
S B I M H Q C L R E P K
Up
down


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
D =
S B I M H Q C L R E P K
Up
down


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
D =
S B I M H Q C L R E P K
Up
down

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
D =	**S**	**B**	**I**	**M**	**H**	**Q**	**C**	**L**	**R**	**E**	**P**	**K**

Up down

Up equal to last so Exchange table[first] and table[down] thus putting the pivot value where it belongs.
We contiune this process by dividing array from pivot and for each part.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
D =	**K**	**B**	**I**	**M**	**H**	**Q**	**C**	**L**	**R**	**E**	**P**	**S**

Pivot

Divide and do same process

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
D =	**K**	**B**	**I**	**M**	**H**	**Q**	**C**	**L**	**R**	**E**	**P**	**S**

Pivot

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
D =	**K**	**B**	**I**	**M**	**H**	**Q**	**C**	**L**	**R**	**E**	**P**	**S**

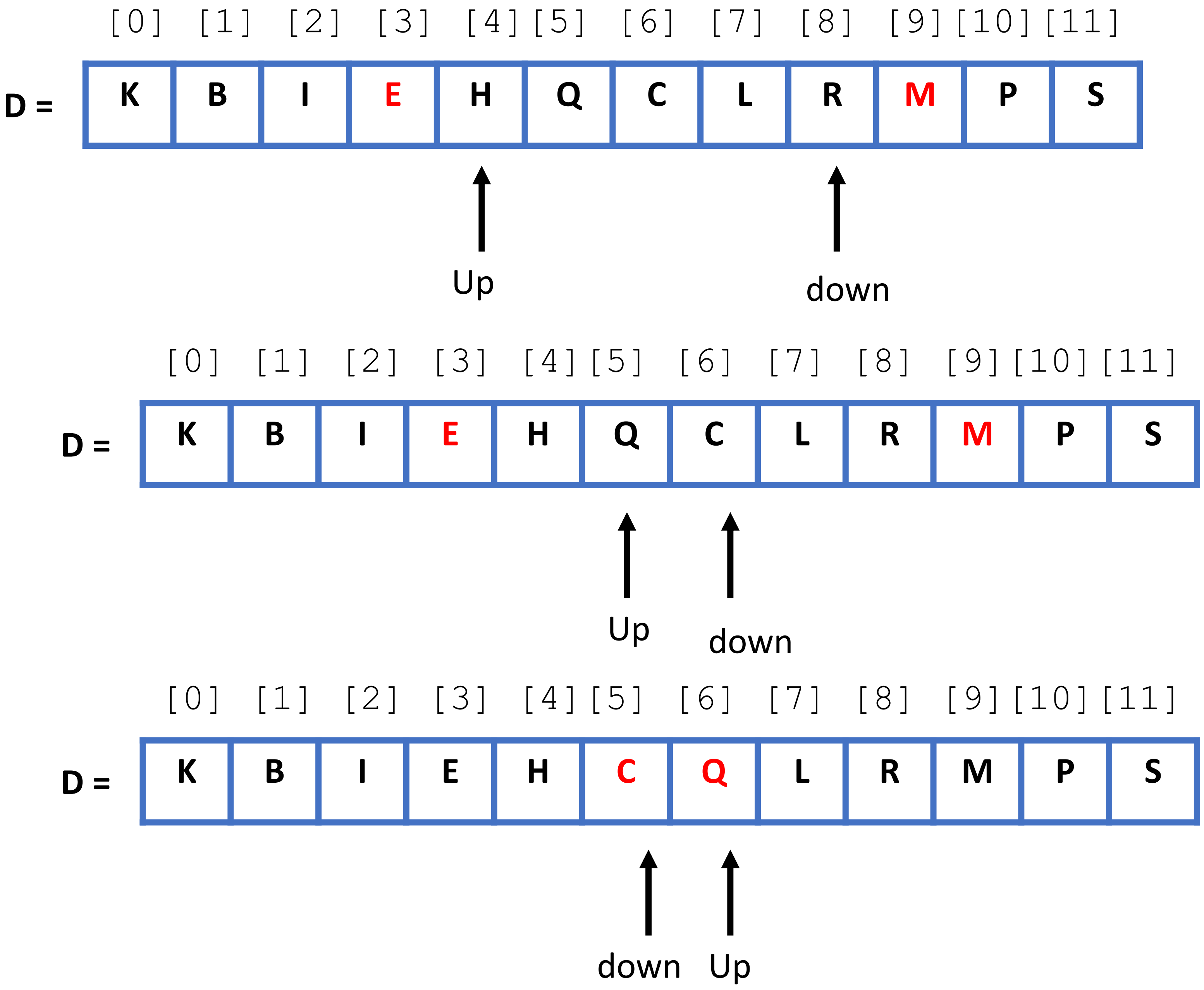
Up

down

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
D =	**K**	**B**	**I**	**M**	**H**	**Q**	**C**	**L**	**R**	**E**	**P**	**S**

Up

down


[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
D = K B I E H Q C L R M P S
Up
down
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
D = K B I E H Q C L R M P S
Up
down
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
D = K B I E H C Q L R M P S
down
Up

Down passes the up so Exchange table[first] and table[down] thus putting the pivot value where it belongs.
We contiune this process by dividing array from pivot and for each part.

D =

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	K	Q	L	R	M	P	S

Pivot

Divide and do same process

D =

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	K	Q	L	R	M	P	S

Pivot

Pivot

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
D =	**B**	**C**	**E**	**H**	**I**	**K**	**L**	**M**	**P**	**Q**	**R**	**S**

<u>Analysis :</u>

Number of comparisons	**Number of displacements**
58	10