9555031137

Sorting

1) No. of Comparision ↓

2) No. of swap ↓

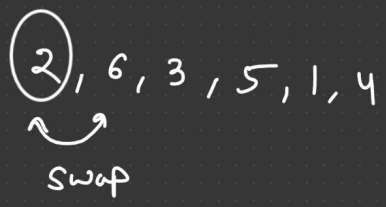
3) Memory usage → In-place ✓
   → x

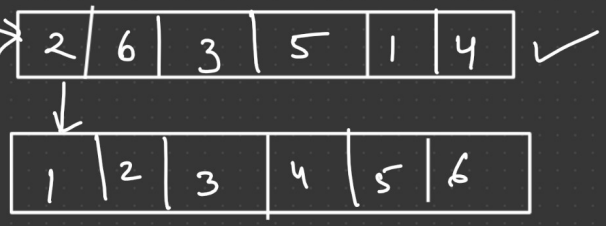
4) Stability :- Maintains relative ordering of duplicate no. ✓

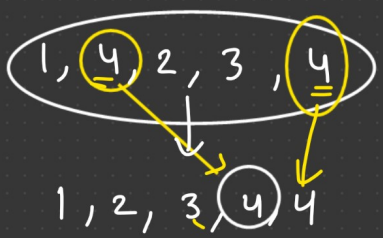
5) Adaptability :- ✓

6) Internal Sort :- RAM

7) External Sort :-







Bubble Sort

I st Iteration :-

2, 8, 3, 5, 9, 8, 12, 11

swap

2, 3, 8, 5, 9, 8, 12, 11

swap

2, 3, 5, 8, 9, 8, 12, 11

swap

2, 3, 5, 8, 8, 9, 12, 11

swap

2, 3, 5, 8, 8, 9, 11 | 12

maximum element now is in right place

II nd Iteration :-

Example:-

I st Iteration:-

23, 4, 1, 22, 10, 5, 11, 19

swap

4, 23, 1, 22, 10, 5, 11, 19

swap

4, 1, 23, 22, 10, 5, 11, 19

swap

4, 1, 22, 23, 10, 5, 11, 19

swap

4, 1, 22, 10, 23, 5, 11, 19

4, 1, 22, 10, 5, 23, 11, 19

swap

4, 1, 22, 10, 5, 11, 23, 19

4, 1, 22, 10, 5, 11, 19 | 23

II nd Iteration:-

1, 4, 22, 10, 5, 11, 19 | 23

swap

1, 4, 10, 22, 5, 11, 19 | 23

swap

1, 4, 10, 5, 22, 11, 19 | 23

swap

1, 4, 10, 5, 11, 22, 19 | 23

swap

1, 4, 10, 5, 11, 19 | 22, 23

swap

III rd Iteration :-

1, 4, 5, 10, 11, 19 | 22, 23 ✓

I → n

II → n-1

III → n-2

⋮

1

Time complexity :-

$n + (n-1) + (n-2) + \dots + 1$

$= \frac{n(n-1)}{2} = O(n^2)$

Radix Sort

$O(n)$

$O(n^2)$ ✓

$O(n \log n)$ ✓

$O(\log n)$ ✗

```
void bubbleSort( int a[] , int n)
{
   int i, j ;
  for( i= n-1 ; i>=0 ; i--)
   {
     for( j= 0 ; j<i ; j++)
     {
        if(a[j] > a[j+1])
        {
           int t = a[j];
               a[j] = a[j+1];
               a[j+1] = t;
        }
     }
} }
```

0, 1, 2, 3, 4, 5

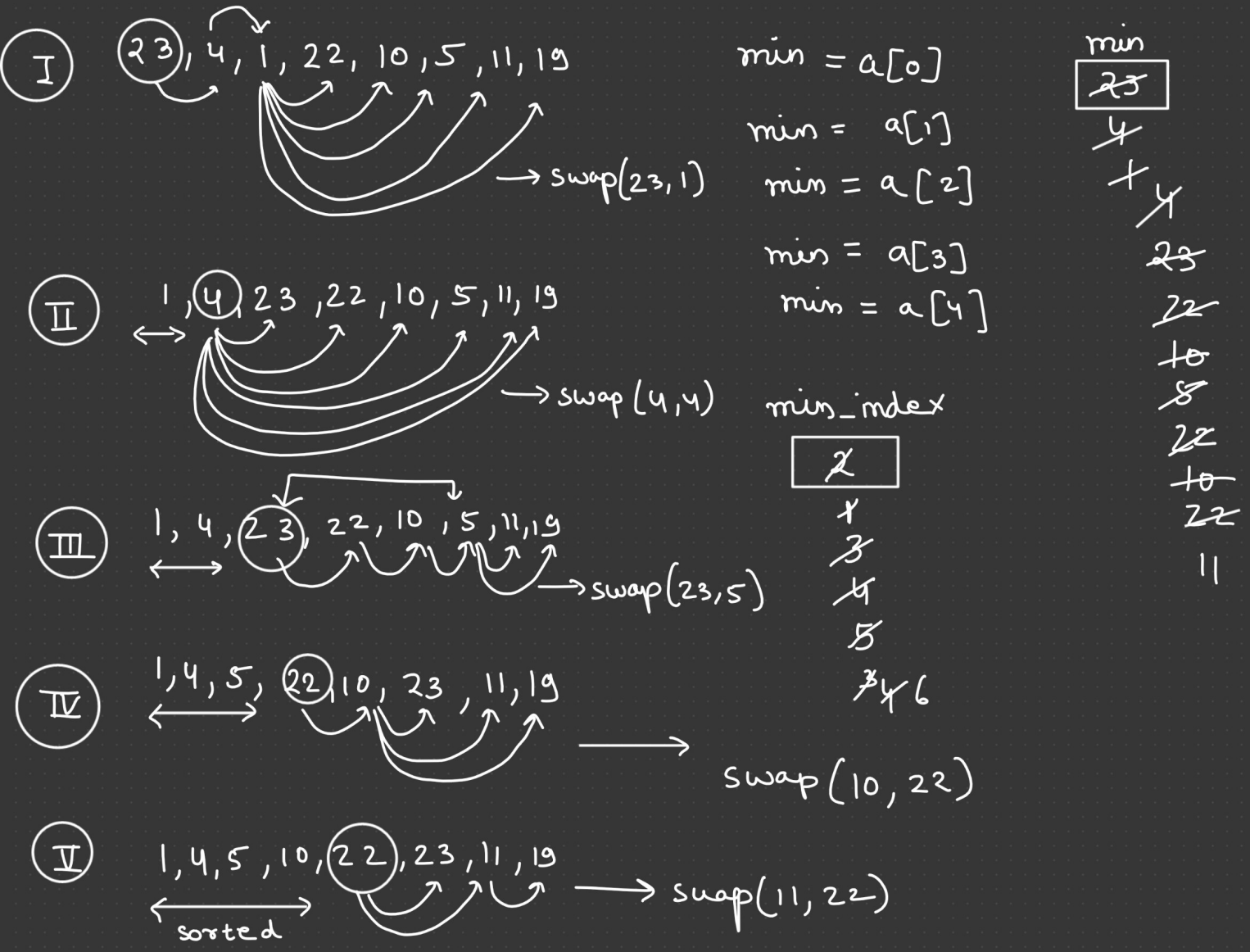
≡ [we can make it adaptable by using flag.]

Selection Sort

Step:-

1) Find the minimum element in the list.
2) Swap it with value in the current position.
3) Repeat this process for all the elements in the entire array.

Ex :-



VI) 1, 4, 5, 10, 11, (23), 22, 19 — min = a[5]

Sorted ⟷ → swap(23, 19)

min	index
~~23~~	~~5~~
~~22~~	~~6~~
19	7

VII) 1, 4, 5, 10, 11, 19, (22), 23 — min = a[6]

sorted ⟷

min	index
22	6

1, 4, 5, 10, 11, 19, 22, 23 → sorted

Time complexity = $O(n^2)$

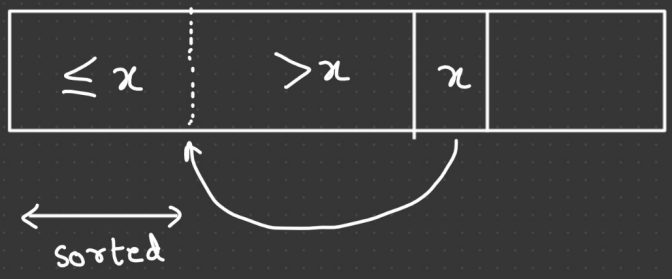
```
void selectionSort( int a[], int n)
{
   int i, j, min, t;
   for(i=0 ; i<n-1; i++)  ⟶ for maintaining the iteration
   {
         min = i;
     for(j=i+1; j<n; j++)
      {
        if(a[min] > a[j])
             min = j;
      }
                          ⟶ swap(a[i], a[min])
       t= a[i];
      a[i] = a[min];
      a[min] = t;
   }
}
```

Insertion Sort

Step:-

1) Select a pivot let say 2nd element of array.

[∵ we assume that 1st element is already sorted]



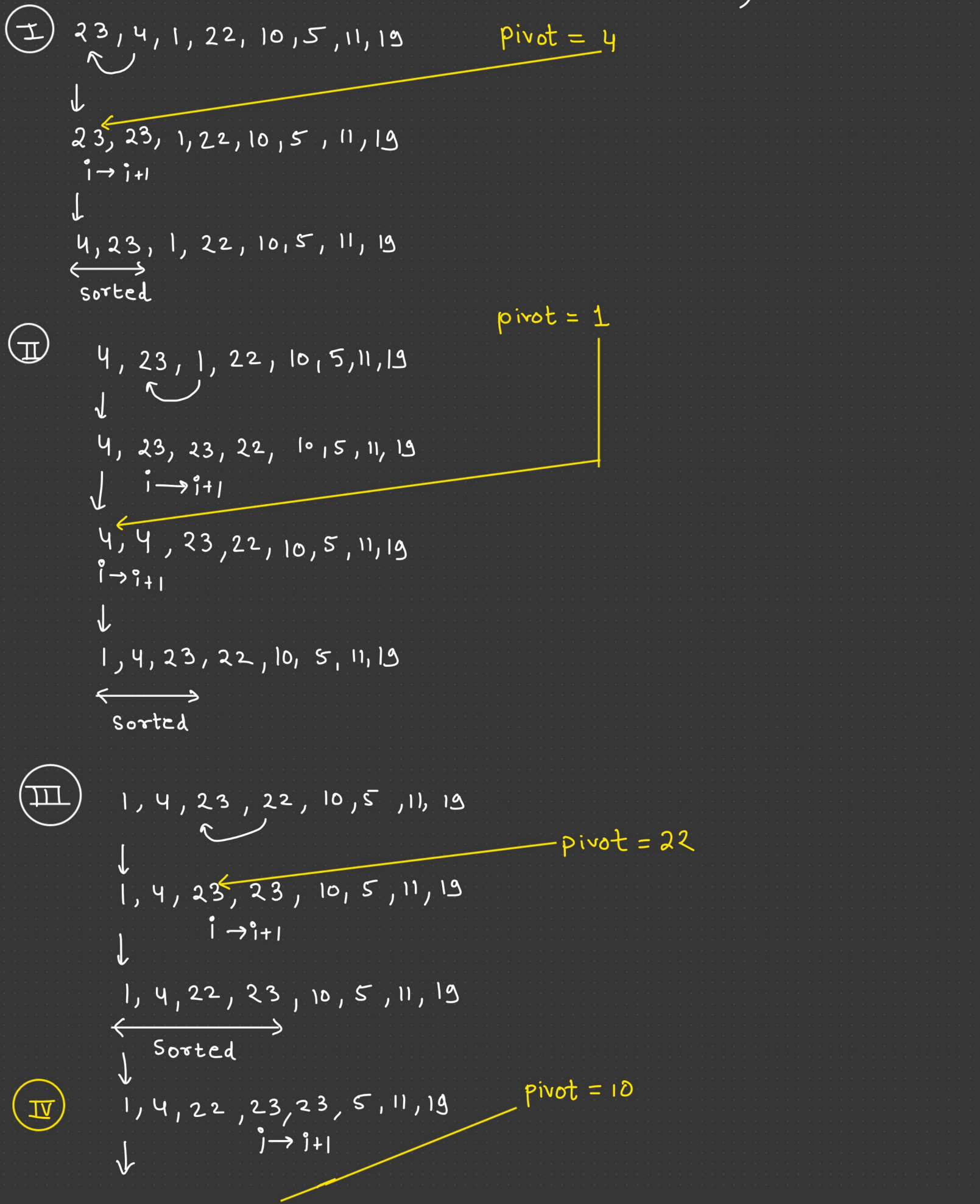
2) we compare pivot with its left sub array and if (pivot < a[i]) then shift a[i] ⟶ a[i+1]

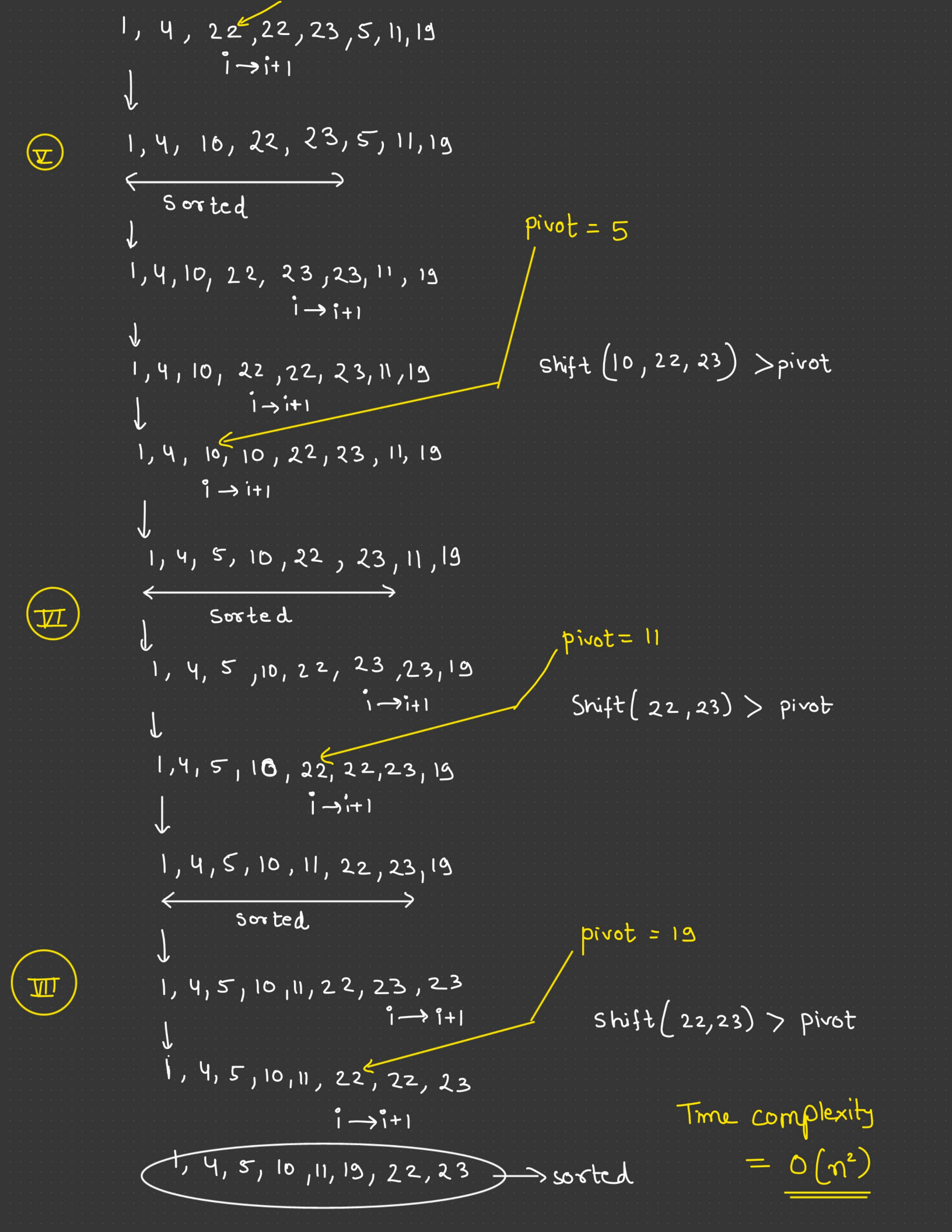
3) Repeat step 2 till the condition satifies.

4) At the end of each iteration, pivot will be placed in its correct position.

5) At the end of all the iteration, Complete array will be sorted.

Example:- (∵ 1st index is already sorted)




1, 4, 22, 22, 23, 5, 11, 19
i → i+1
V
1, 4, 10, 22, 23, 5, 11, 19
Sorted
pivot = 5
1, 4, 10, 22, 23, 23, 11, 19
i → i+1
1, 4, 10, 22, 22, 23, 11, 19
i → i+1
Shift (10, 22, 23) > pivot
1, 4, 10, 10, 22, 23, 11, 19
i → i+1
VI
1, 4, 5, 10, 22, 23, 11, 19
Sorted
pivot = 11
1, 4, 5, 10, 22, 23, 23, 19
i → i+1
Shift (22, 23) > pivot
1, 4, 5, 10, 22, 22, 23, 19
i → i+1
1, 4, 5, 10, 11, 22, 23, 19
sorted
VII
pivot = 19
1, 4, 5, 10, 11, 22, 23, 23
i → i+1
Shift (22, 23) > pivot
1, 4, 5, 10, 11, 22, 22, 23
i → i+1
1, 4, 5, 10, 11, 19, 22, 23
sorted
Time complexity
= O(n²)

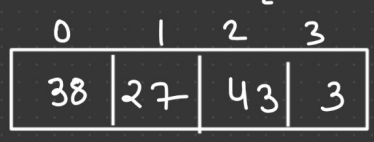
```
void insertionSort(int a[], int n)
{
    for(int i=1; i<n; i++)        ⟶ for selecting the pivot
    {
        int pivot = a[i];
        int j = i-1;
        while(a[j] > pivot && j>=0)
                  C1          C2
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = pivot;
    }
}
```

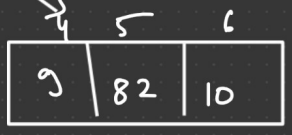
Merge Sort

It is an example Divide & Conquer.

$$mid = \frac{start + end}{2}$$

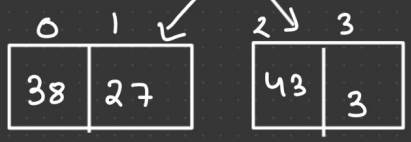
$$= \frac{0+6}{2} = 3$$

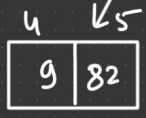


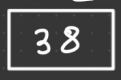


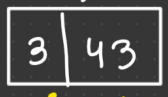
$$mid = \frac{0+3}{2} = 1$$

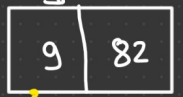
$$mid = \frac{4+6}{2} = 5$$











3<27

27<43

38<43

3<9 1

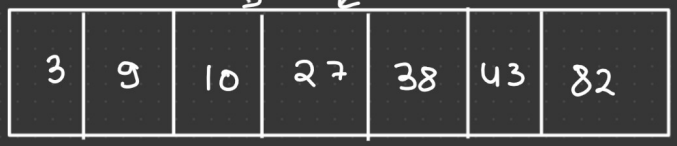
9<27 2

10<27 3

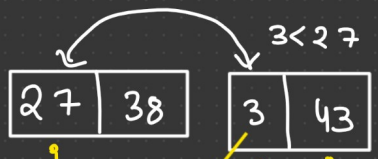
27<82 4

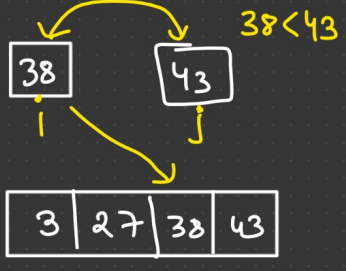
38<82 5

43<82 6

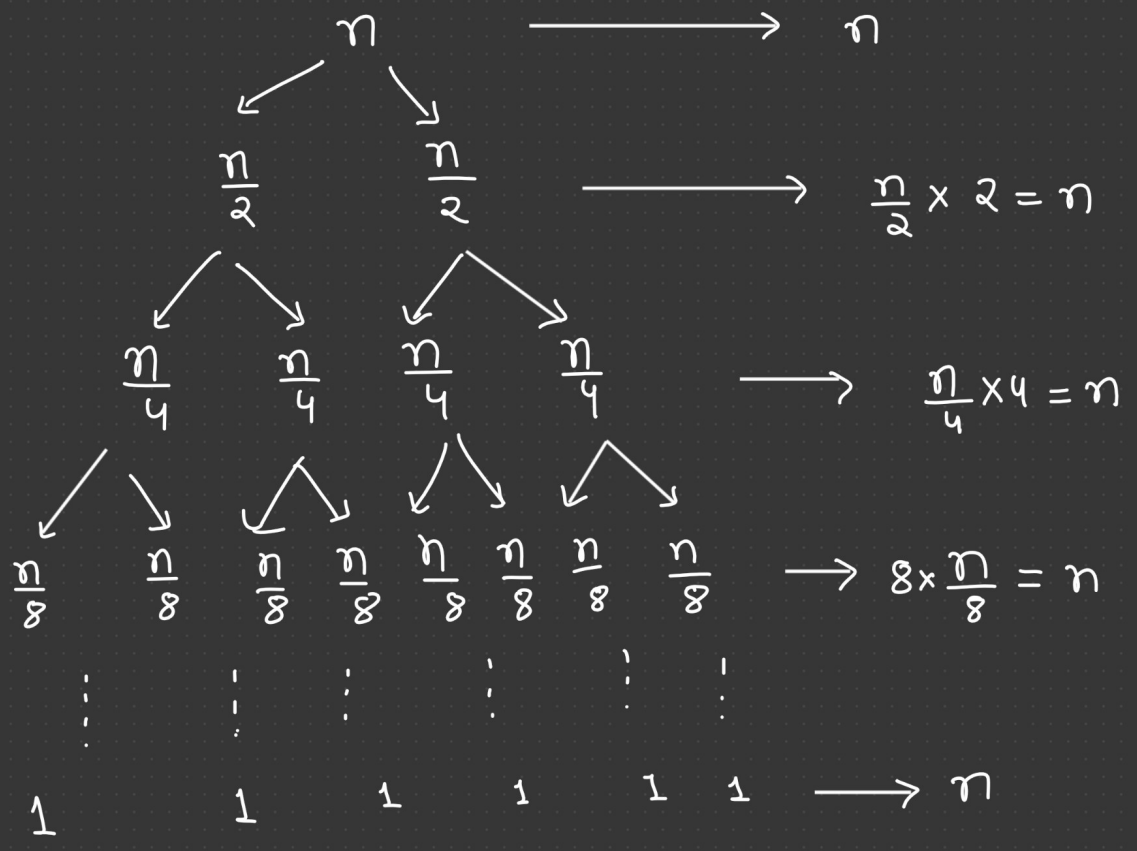


Sorted





Time Complexity:-



Time Complexity :- $n + n + n + \ldots$ (upto height of tree)

$\because$ Height of Tree $= \log n$

$= n(\log n) \Rightarrow O(n \log n)$

$n + n + n = n(1+1+1) = n \times 3$

$n + n + n + n = n(1+1+1+1) = n \times 4$

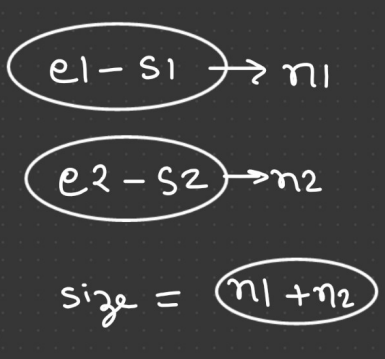
$n + n + \ldots$ $\log n$ time $= n(1 + 1 + 1 + \ldots \log n \text{ times}) = n \times \log n$

```
void mergesort( int a[], int i, int j)
{
    int mid = (i+j)/2;

    if (i< j)
    {
        mergesort(a, i, mid);
        mergesort(a, mid+1, j);
        merge(a, i, mid, mid+1, j);
    }
}
```


```
void merge( int a[], int s1, int e1, int s2, int e2)
{
    int temp[50], i=s1, j=s2, k=0;

    while( i<=e1 && j<=e2)
    {
        if(a[i] < a[j])
        {
            temp[k] = a[i];
            i++, k++;
        }
        else
        {
            temp[k]=a[j];
            j++, k++;
        }
    }
```



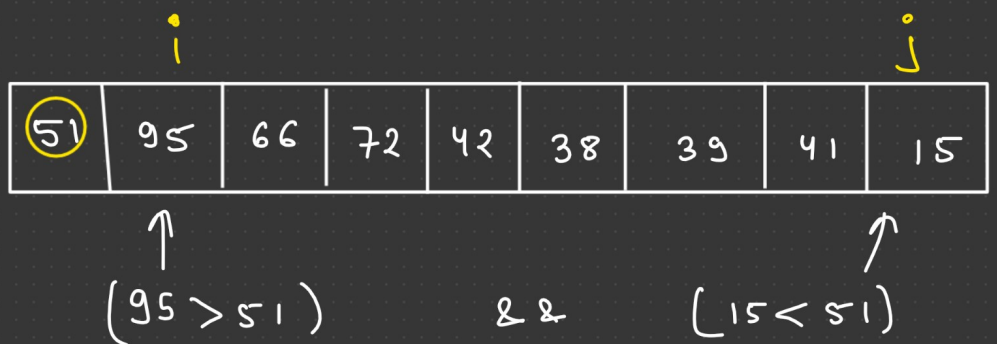
```
    while (i <= e1)
    {
        temp[k] = a[i];
        i++, k++;
    }
    while (j<=e2)
    {
        temp [k] = a[j];
        j++, k++;
    }
for (k=0, i = s1 ; i<=e2   ; i++, k++)
{
    a[i] = temp[k];
}
}
```

9555031137

Quick Sort

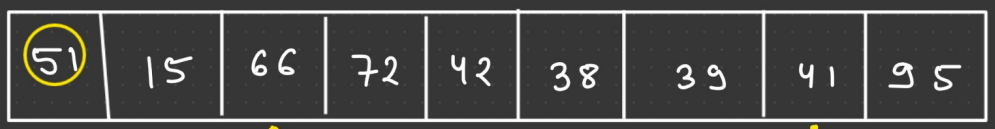
pivot = 51

if(a[i] > pivot && a[j] < pivot)
Swap(a[i], a[j])
i++, j--



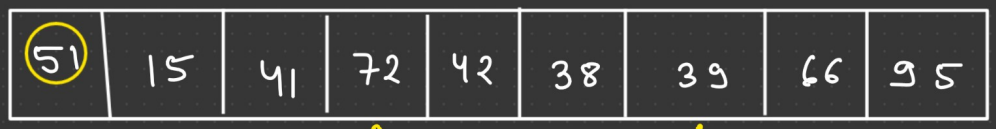
(95 > 51) && (15 < 51)

swap(95, 15)



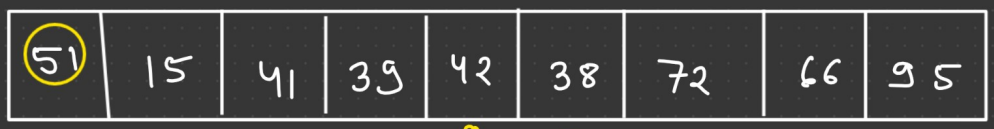
i (66 > 51) && (41 < 51) j

swap(66, 41)



i j

(72 > 51) && (39 < 51) swap(72, 39)



i j

No swap

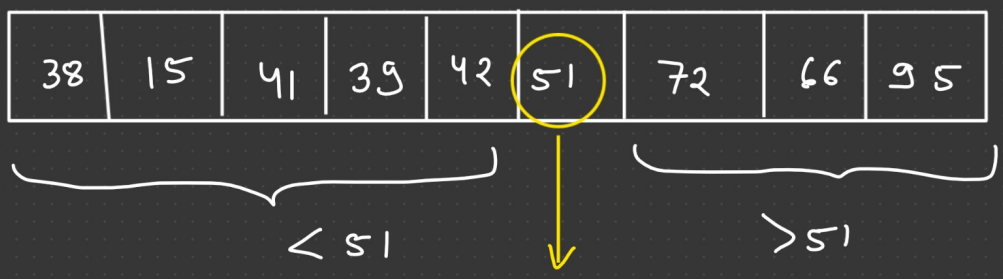
① i > p j < p

② i < p → i++

③ j > p → j--

(42 < 51) && (38 < 51)

Swap(51, 38)

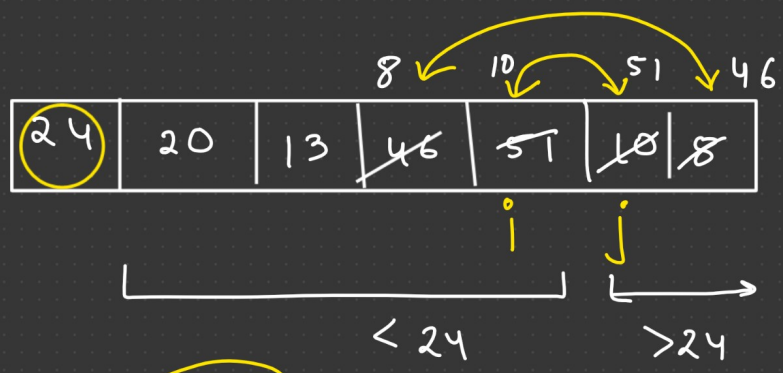


Now 51 is at correct position.

15	38	41	39	42	51	72	66	95

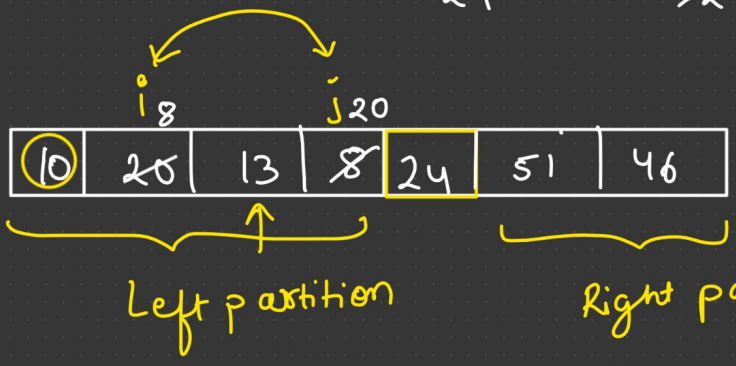
15	38	41	39	42	51	72	66	95

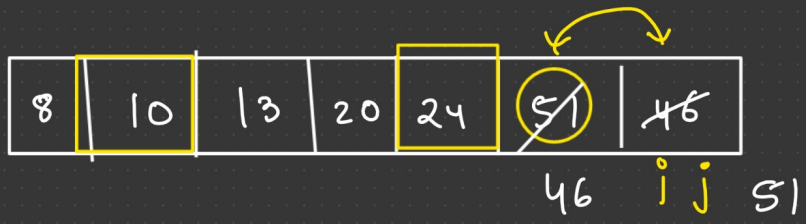
Ex:-

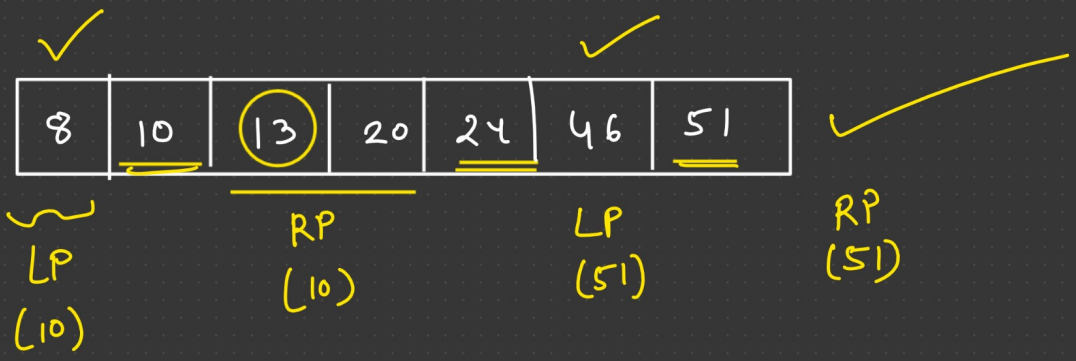


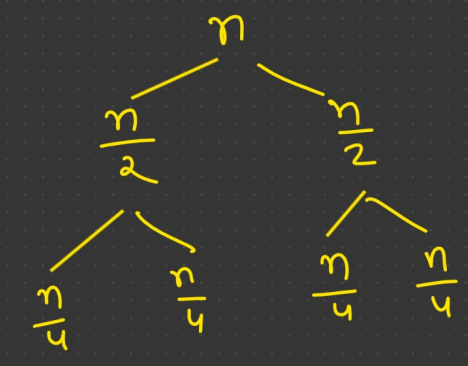
i<P → i++

j>P → j--

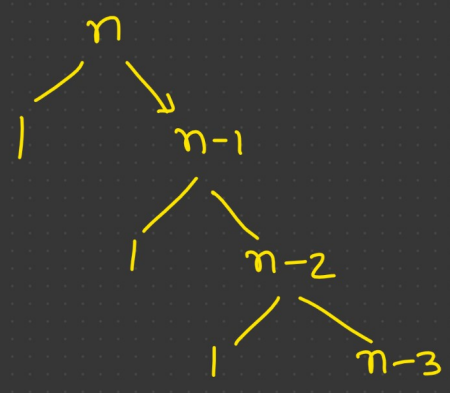








Best = $\Omega(n \log n)$

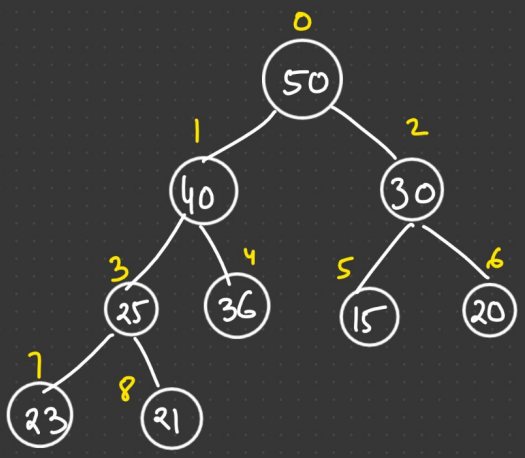


worst case

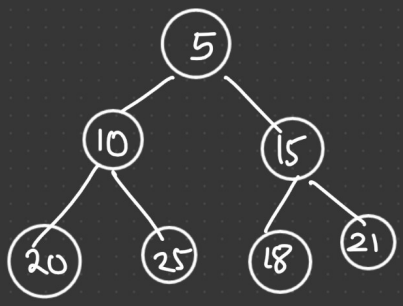
$n + (n-1) + (n-2) + \cdots + 1 = O(n^2)$

Heap Sort

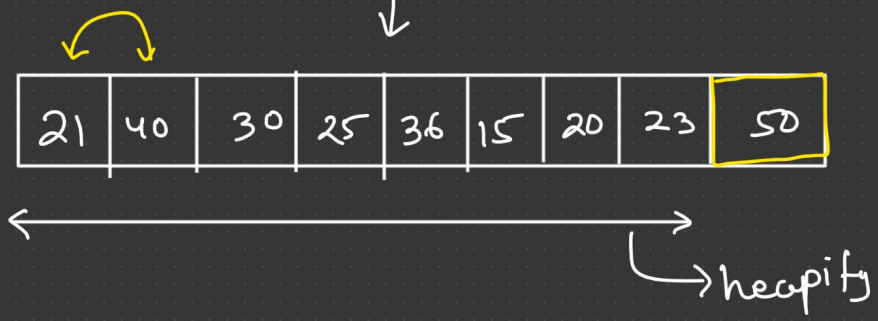
Max Heap

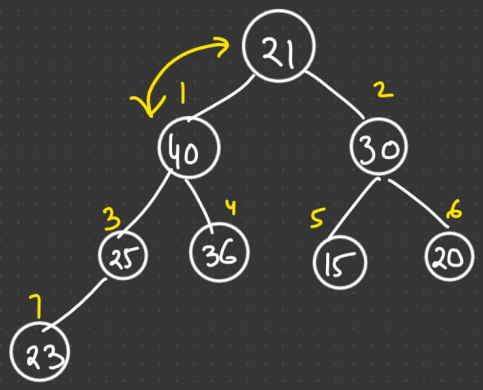


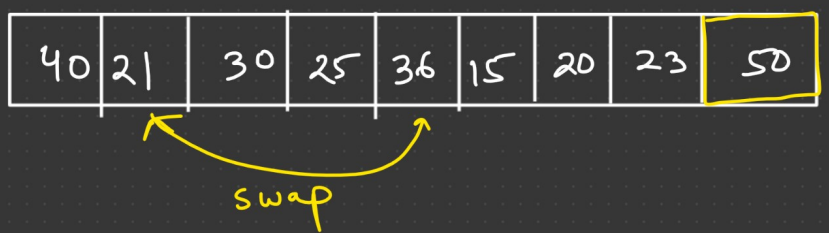
Min Heap

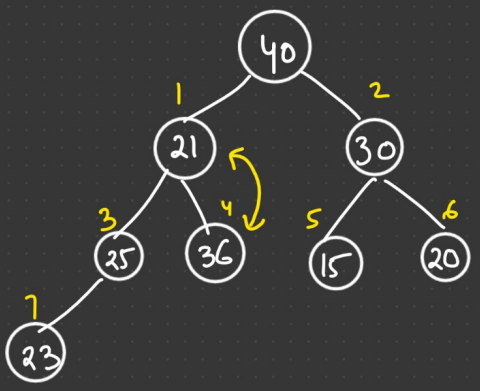


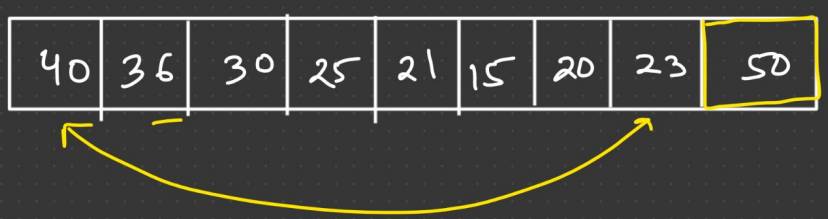
It must be complete tree

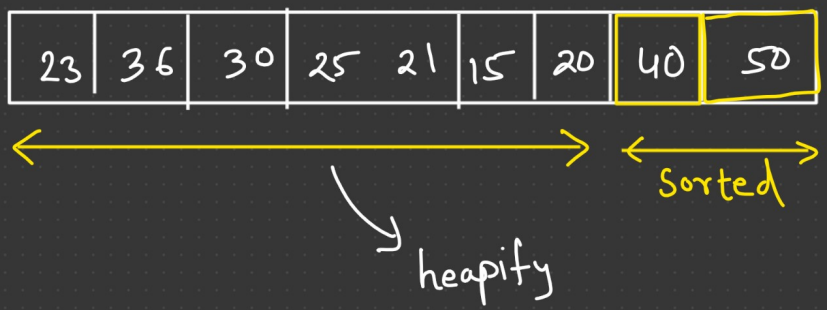


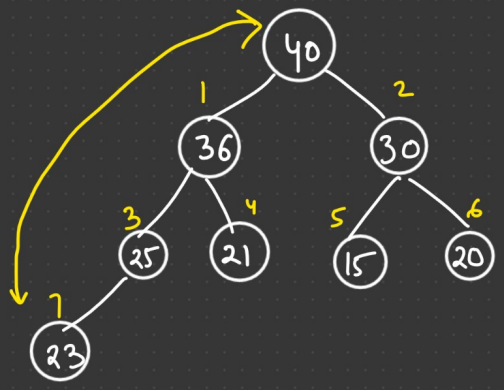




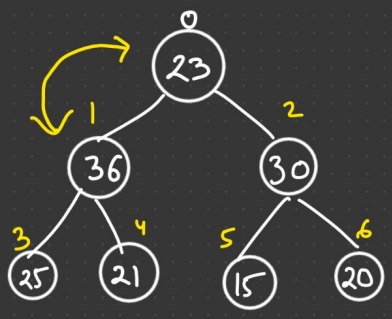




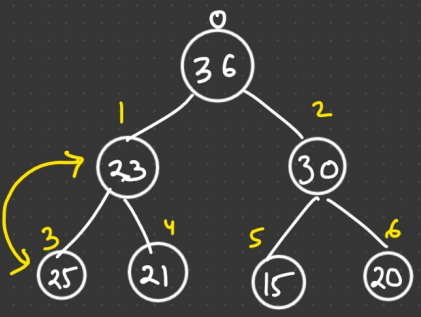

23 36 30 25 21 15 20 40 50
Sorted
heapify


40
36
30
25
21
15
20
23

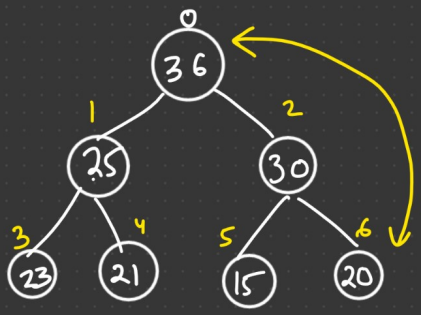

23 36 30 25 21 15 20 40 50

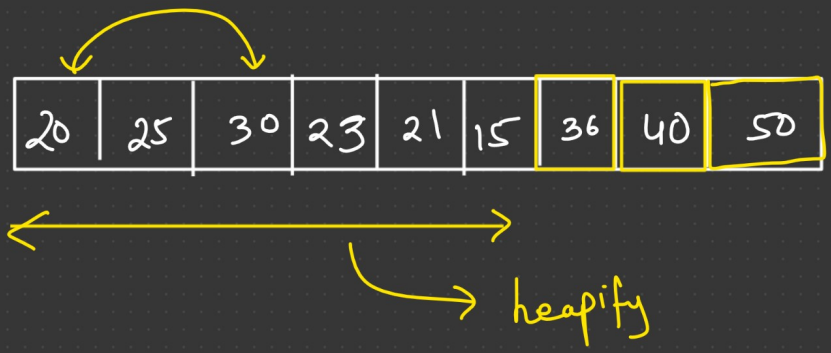

23
36
30
25
21
15
20

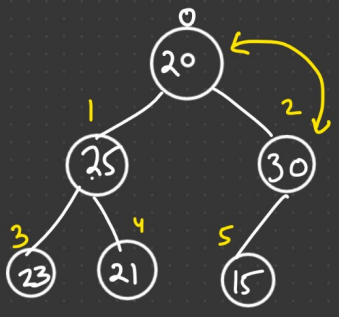

36 23 30 25 21 15 20 40 50
swap


36
23
30
25
21
15
20

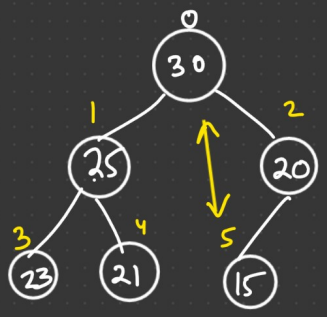

36 25 30 23 21 15 20 40 50

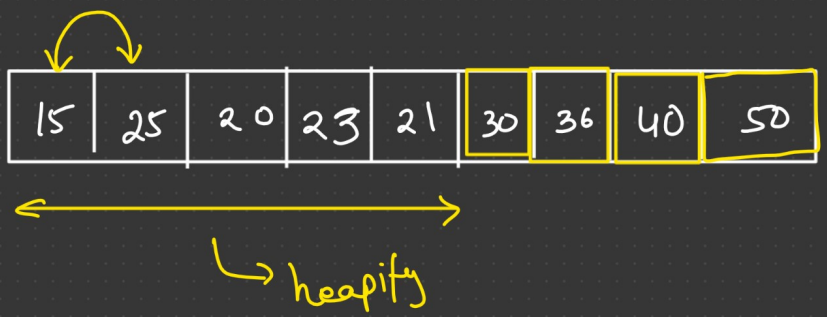

36
25
30
23
21
15
20

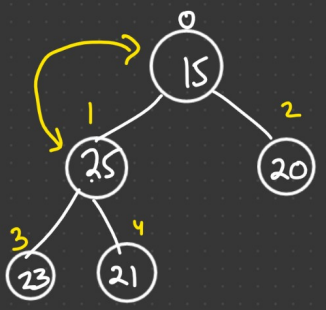

20 25 30 23 21 15 36 40 50
heapify

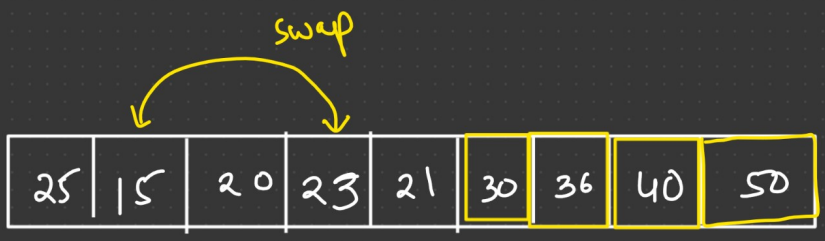

20
25
30
23
21
15

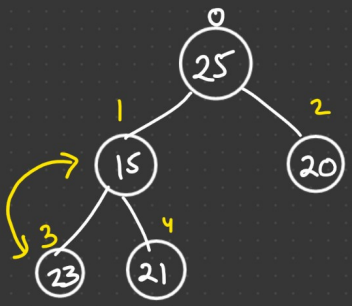

30 25 20 23 21 15 36 40 50

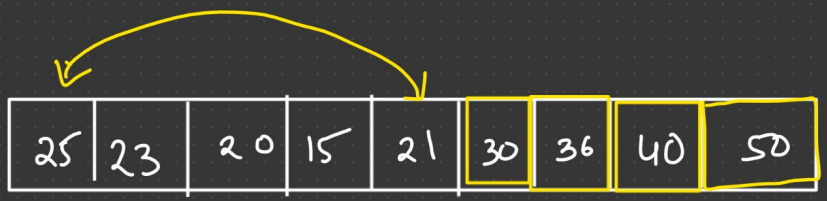

30
25
20
23
21
15
1
2
3
4
5

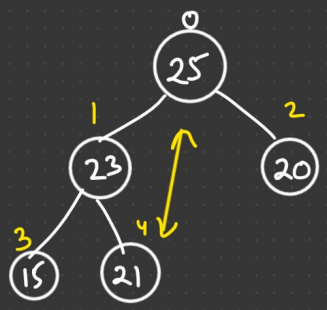

15 25 20 23 21 30 36 40 50
heapify

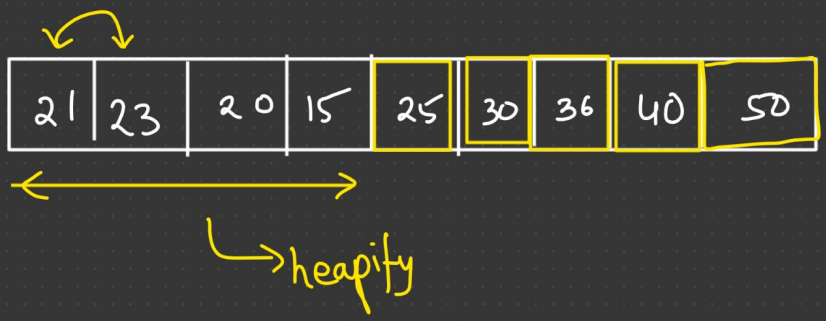

15
25
20
23
21
1
2
3
4

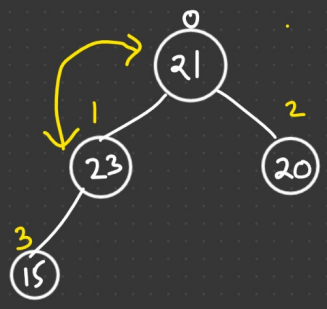

swap
25 15 20 23 21 30 36 40 50


25
15
20
23
21
1
2
3
4

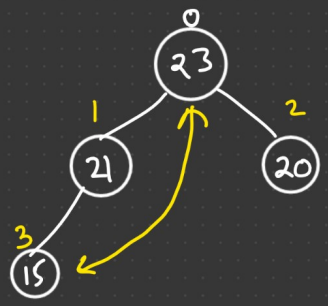

25 23 20 15 21 30 36 40 50


25
23
20
15
21
1
2
3
4

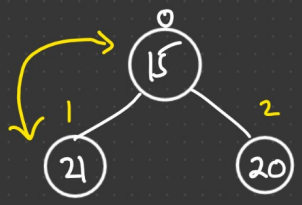

21 23 20 15 25 30 36 40 50
heapify

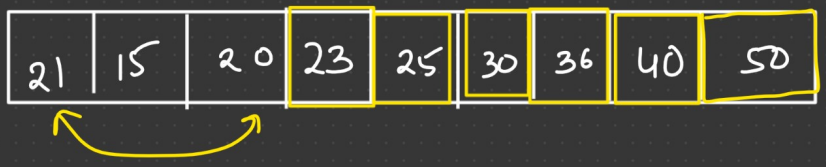

21
23
20
15
1
2
3

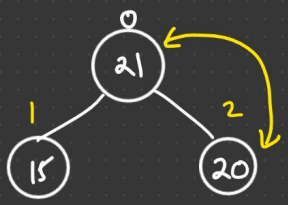

23 21 20 15 25 30 36 40 50


0
23
1
21
2
20
3
15

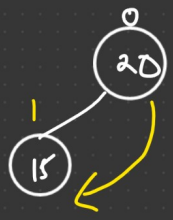

15 21 20 23 25 30 36 40 50
heapify


0
15
1
21
2
20


21 15 20 23 25 30 36 40 50


0
21
1
15
2
20


20 15 21 23 25 30 36 40 50
heapify


0
20
1
15


15


15 20 21 23 25 30 36 40 50