

Sorting

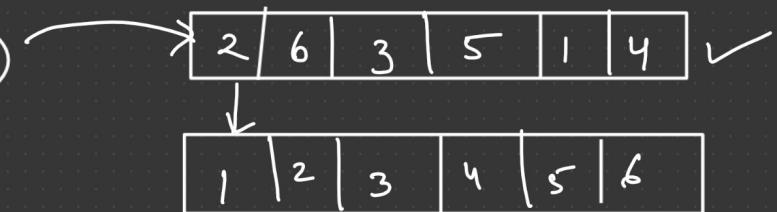
9555031137

1) No. of Comparison \downarrow

(2), 6, 3, 5, 1, 4
swap

2) No. of swap \downarrow

3) Memory usage \rightarrow In-place \checkmark



4) Stability :- maintains relative ordering of duplicate no. \checkmark

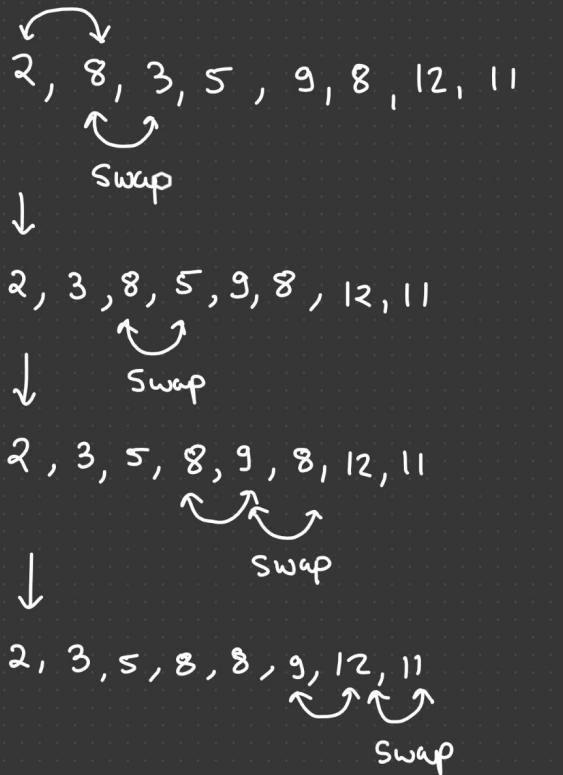
1, 4, 2, 3, 4
1, 2, 3, 4, 4

5) Adaptability :- \checkmark

6) Internal Sort :- RAM

7) External Sort :-

Ist Iteration :-



Bubble Sort

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

12

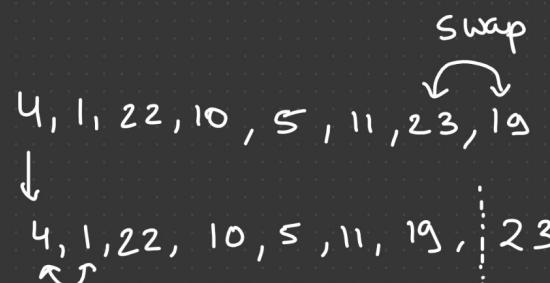
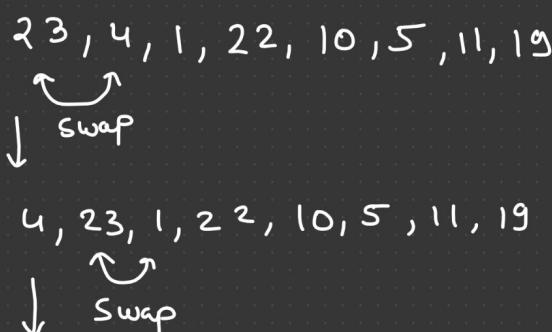
maximum element

now is in right place

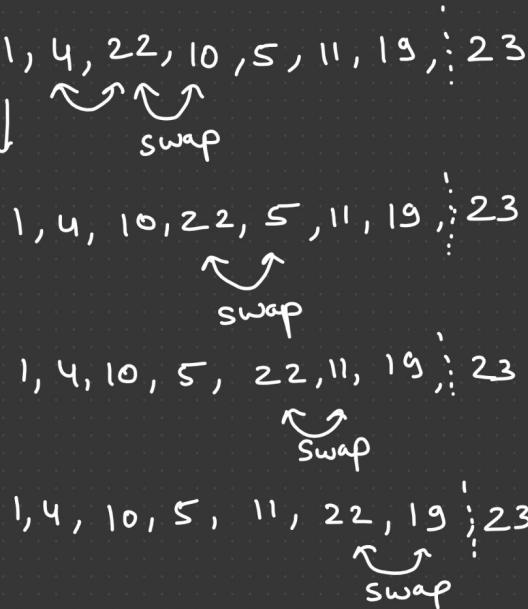
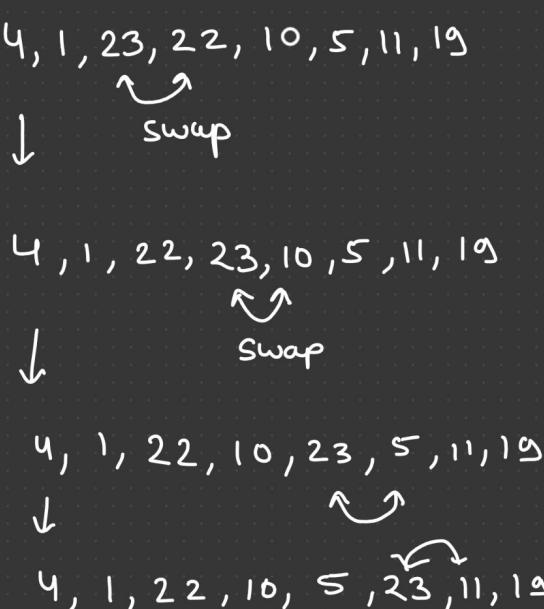
IInd Iteration :-

Example:-

Ist Iteration:-



IInd Iteration :-



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

IIIrd 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(n²) ✓

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
 ↘ ↘ ↘ ↘ ↘ ↘

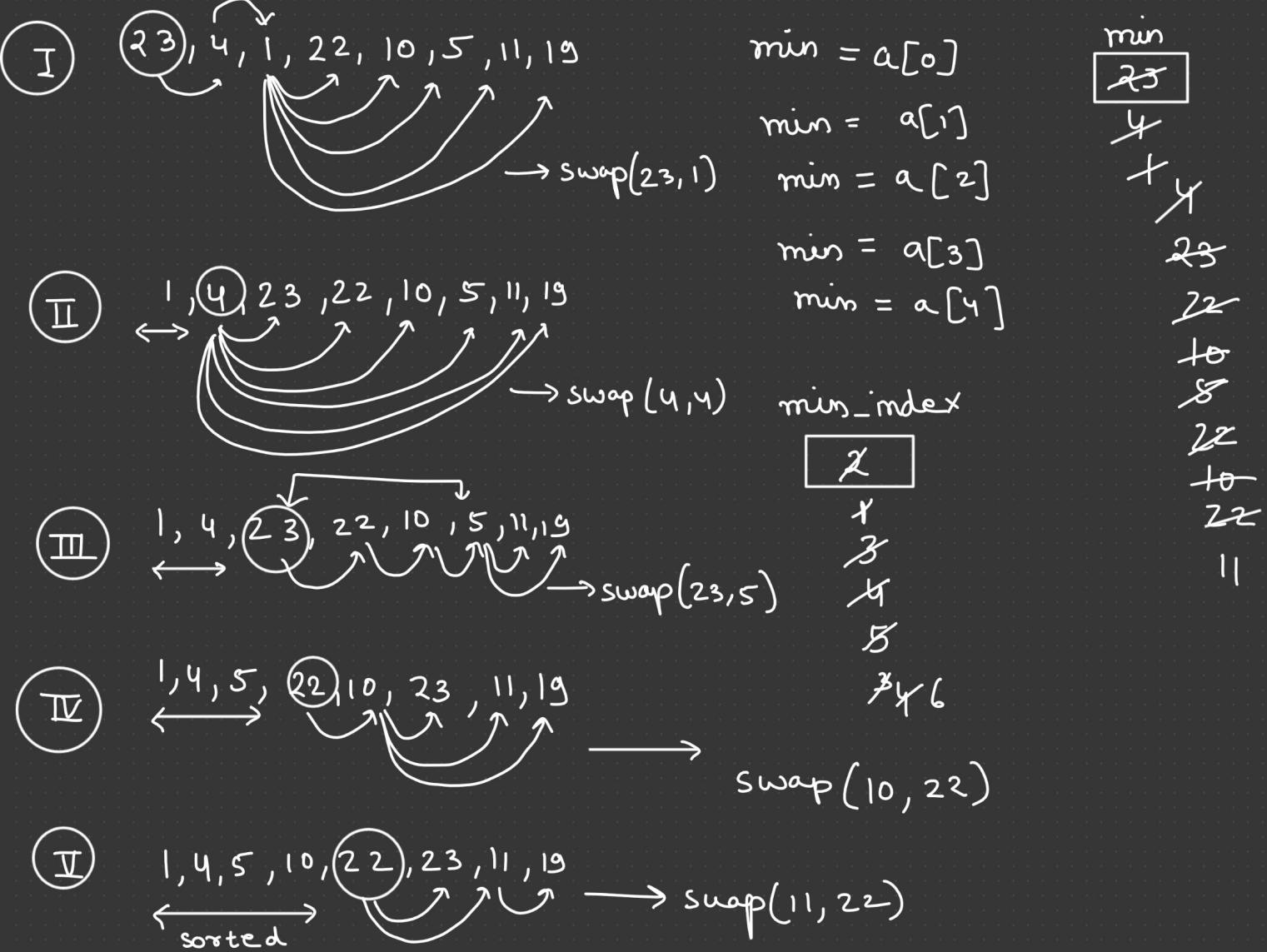
✗ [we can make it
 adaptable by using flag.]

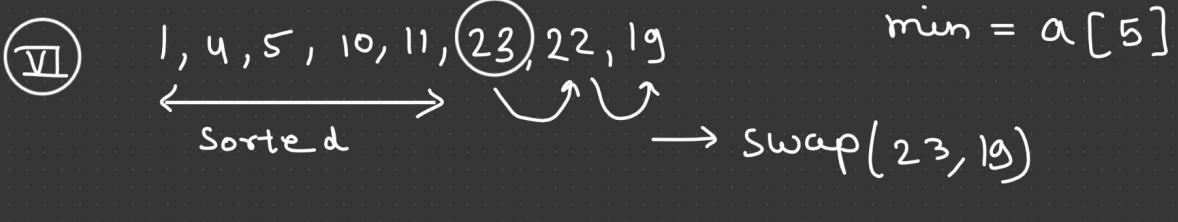
Selection Sort

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 :-





1, 4, 5, 10, 11, 19, 22, 23 → sorted

Time Complexity = O(n²)

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;

{

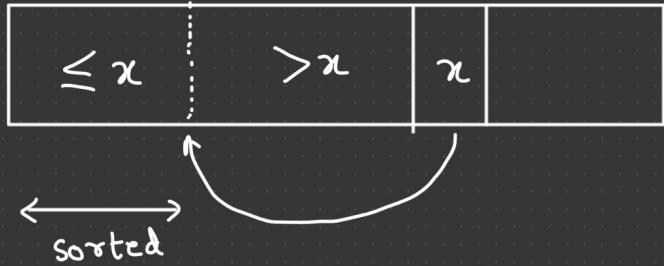
}

Insertion Sort

Step:-

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

[\because we assume that 1st element is already sorted]



2) we compare pivot with its left subarray and
if ($\text{pivot} < a[i]$) then shift $a[i] \rightarrow a[i+1]$

3) Repeat step 2 till the condition satisfies.

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)

I 23, 4, 1, 22, 10, 5, 11, 19 pivot = 4



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

i → i+1



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

↔
sorted

pivot = 1

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



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

↓

i → i+1

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

↓

i → i+1



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

↔
sorted

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



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

↓

i → i+1



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

↔
sorted



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

↓

j → i+1

pivot = 22

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

pivot = 10

1, 4, 22, 22, 23, 5, 11, 19
 $i \rightarrow i+1$

↓
V
1, 4, 10, 22, 23, 5, 11, 19
← →
sorted

↓
1, 4, 10, 22, 23, 23, 11, 19
 $i \rightarrow i+1$

↓
1, 4, 10, 22, 22, 23, 11, 19
 $i \rightarrow i+1$

↓
1, 4, 10, 10, 22, 23, 11, 19
 $i \rightarrow i+1$

↓
1, 4, 5, 10, 22, 23, 11, 19
← →
sorted

pivot = 5

shift (10, 22, 23) > pivot

↓
VI
1, 4, 5, 10, 22, 23, 23, 19
 $i \rightarrow i+1$

pivot = 11

Shift (22, 23) > pivot

↓
1, 4, 5, 10, 22, 22, 23, 19
 $i \rightarrow i+1$

↓
1, 4, 5, 10, 11, 22, 23, 19
← →
sorted

pivot = 19

↓
1, 4, 5, 10, 11, 22, 23, 23
 $i \rightarrow i+1$

↓
1, 4, 5, 10, 11, 22, 22, 23
 $i \rightarrow i+1$

shift (22, 23) > pivot

1, 4, 5, 10, 11, 19, 22, 23
→ sorted

Time complexity
 $= O(n^2)$

```
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(  $\frac{a[j]}{c_1} > \text{pivot}$  &&  $\frac{j}{c_2} \geq 0$  )
```

```
{
```

```
        a[j+1] = a[j];
```

```
        j--;
```

```
}
```

```
    a[j+1] = pivot;
```

```
}
```

```
}
```

Merge Sort

It is an example Divide & Conquer.



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

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

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

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

$$= 1$$



$3 < 27$

$27 < 43$

$38 < 43$

$K \rightarrow K+1$



$3 < 9$

$9 < 27$

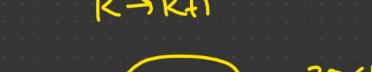
$10 < 27$

$27 < 82$

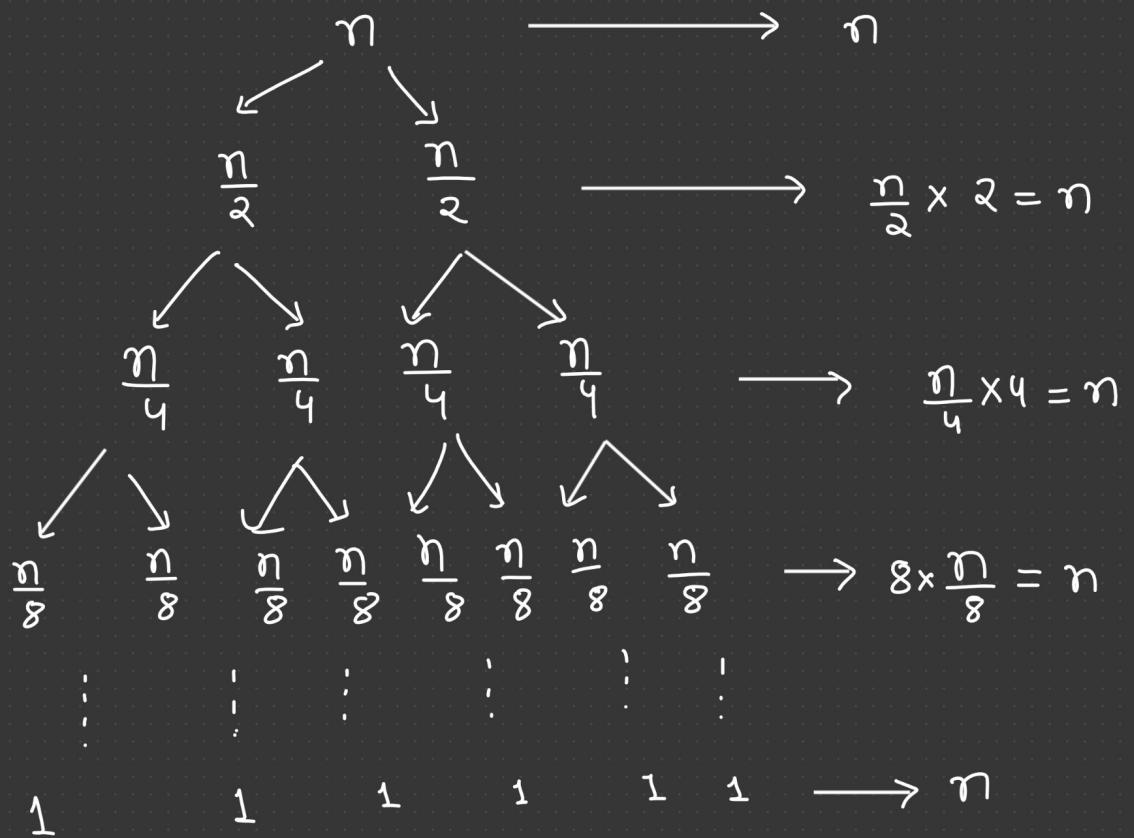
$38 < 82$

$43 < 82$

Sorted



Time Complexity:-



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

\therefore Height of Tree = $\log n$

$$= n(\log n) \Rightarrow \underline{\underline{O(n \log n)}}$$

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

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

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

```
Void mergesort( int a[], int i, int j)
```

```
{
```

```
    int mid =  $\frac{i+j}{2}$ ;
```



```
    ... | ... | ... | ...
```

```
    i     mid   mid+1   j
```

```
    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++;
```

```
}
```

```
}
```

$$e1 - s1 \rightarrow n_1$$

$$e2 - s2 \rightarrow n_2$$

$$\text{size} = n_1 + n_2$$

```
while ( i <= c1 )
{
    temp[k] = a[i];
    i++, k++;
}
```

```
while ( j <= c2 )
{
    temp[k] = a[j];
    j++, k++;
}
```

```
for ( k=0, i = s1 ; i <= c2 ; i++, k++ )
{
    a[i] = temp[k];
}
```

9555 031137

Quick Sort

pivot = 51

i		j						
51	95	66	72	42	38	39	41	15

$(95 > 51)$ $\&&$ $(15 < 51)$
 \downarrow
 swap(95, 15)

$\text{if } a[i] > \text{pivot} \text{ & } a[j] < \text{pivot}$
 $\text{swap}(a[i], a[j])$
 $i++, j--$

i	15	66	72	42	38	39	41	95	j
---	----	----	----	----	----	----	----	----	---

$(66 > 51) \&& (41 < 51)$
 swap(66, 41)

i	15	41	72	42	38	39	66	95	j
---	----	----	----	----	----	----	----	----	---

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

$(42 < 51) \&& (38 < 51)$
 Swap(51, 38)

38	15	41	39	42	51	72	66	95
< 51					↓	> 51		

Now 51 is at correct position.