Sorting Basics

Sorting :- Arrangement of data in particular order, based on some parameter.

Ex :- { 2, 3, 4, 12, 17, 19 }

* The above elements are sorted in ascending order on the basis of magnitude.

Ex :- { 19, 6, 5, 2, -1, -19 }

* The above elements are sorted in descending order on the basis of magnitude.

Quiz :- Is the array {1, 13, 9, 6, 12} sorted?

A[] = { 1, 13, 9, 6, 12 }
↓ ↓ ↓ ↓ ↓
1 2 3 4 6

Sorted on the basis count of factors.

Why Sorting?

Sorting is essential for

→ organising
→ Searching
→ Analysing
→ Presenting data effectively & efficiently in various applications & context

Question :- Given an array of n integers, minimize the cost to empty given array where cost of removing an element is equal to Sum of all elements left in an array.

Ex :- A[] = { 2, 1, 4 }

↓ removing 4 ⇒ 2+1+4 ⇒ 7

{ 2, 1 }

↓ removing 2 ⇒ 2+1 ⇒ 3

{ 1 }

↓ removing 1 ⇒ 1

{ }

Total Cost ⇒ 7 + 3 + 1 = 11

* Let try to remove in different order

A[] = { 2, 1, 4 }

↓ removing 2 ⇒ 2+1+4 ⇒ 7

{ 1, 4 }

↓ removing 1 ⇒ 1+4 ⇒ 5

{ 4 }

↓ removing 4 ⇒ 4

{ }

Total Cost = 7 + 5 + 4 ⇒ 16

* We can see the order of removal decides Cost.

Quiz 2 : Minimize cost to remove all elements from array { 4, 6, 1 }

A[] = { 4 , 6, 1 }

↓ removing 6 => 4+6+1 => 11

{ 4, 1 }

↓ removing 4 => 1+4 => 5

{ 1 }

↓ removing 1 => 1

Total Cost = 17

Quiz 3 : Minimize cost to remove all elements from array [] = { 3, 5, 1, -3 }

A[] = { 3, 5, 1, -3 }

↓ removing 5 => 3+5+1-3 => 6

{ 3, 1, -3 }

↓ removing 3 => 3+1-3 => 1

{ 1, -3 }

↓ removing 1 => 1-3 => -2

{ -3 }

↓ removing -3 => -3

Total Cost => 6+1-2-3 => 2

Observation

① Start removing the largest element.

[a, b, c, d]

removing a ⇒ a + b + c + d
removing b ⇒ b + c + d
removing c ⇒ c + d
removing d ⇒ d

Find Cost $a + 2b + 3c + 4d$

$a > b > c > d$

↳ The smallest no. should have largest coefficient & largest should have smallest coefficient.

Eg :- { 5, 3, 1, -3 }

(1)5 + (2)3 + (3)1 + (4)(-3)

⇒ 5 + 6 + 3 − 12 ⇒ 2

PSEUDO CODE

O(N log N) ← reverse. sort (arr)

O(N) ←

```
int ans = 0;
for (i = 0; i < arr.length; i++){
    ans += arr[i] * (i+1);
}
return ans
```

TC ⇒ O(N log N)
SC ⇒ O(1) → Assuming Sorting takes O(1) space

Question: Given an array of distinct elements of size n, find the count of Noble Integer

Note: arr[i] is noble if count of elements smaller than arr[i] is equal to arr[i]

Ex

A[] = { 1, -5, 3, 5, -10, 4 }

count of smaller elements → 2, 1, 3, 5, 0, 4

Ans = 3

Ques 4: Count the number of noble integers in the array A = { -3, 0, 2, 5 }

A = { -3, 0, 2, 5 }

count of smaller → 0, 1, 2, 3

Ans => 1

Brute force => For each Number try counting the smaller element

PSEUDO CODE

```
ans = 0
for (i = 0 ; i < A.length ; i++){
    count = 0
    for (j = 0 ; j < A.length ; j++){
        if (A[j] < A[i]){
            count++;
        }
    }
    if (count == A[i]){
        ans++;
    }
}
```

TC → O(N²)

SC → O(1)

OPTIMIZATION

Ex A[] = { 1, -5, 3, 5, -10, 4 }

	↓	↓	↓	↓	↓	↓
count of smaller elements →	2	1	3	5	0	4

Hint 1:- What extra we are doing in brute force

Hint 2:- Can we sorting anyhow?

Observation

① Sorting the array

	0	1	2	3	4	5
A[] =	-10	-5	1	3	4	5
count of smaller →	0	1	2	3	4	5

② indexes are behaving like smaller count after sorting

PSEUDO CODE

O(N log N) ←

```
Arrays.sort(arr)
ans = 0;
for (i = 0; i < arr.length; i++){
    count = i;
    if (count == arr[i]){
        ans++;
    }
}
print(ans);
```

TC → O(N log N)

SC → O(1)

Question :- Same as prev question, but elements are not distinct.

Quiz 5 : Count the no. of noble integers in the array.

$A = \{-10, 1, 1, 3, 100\}$ (indices 0 1 2 3 4)

Count of Smaller → 0 1 1 3 4

observation ⇒ The previous approach might not help us because it was using indexes.

Quiz 6 : Count the no. of noble integers in the array.

$A = \{-10, 1, 1, 2, 4, 4, 4, 8, 10\}$ (indices 0 1 2 3 4 5 6 7 8)

Count of Smaller → 0 1 1 3 4 4 4 7 8

Ans = 5

Observation :- For first occurrence of each number the prev. approach is holding true.

Quiz 7 : Count the no. of noble integers in the array.

$A = \{-3, 0, 2, 2, 5, 5, 5, 5, 8, 8, 10, 10, 10, 14\}$ (indices 0 1 2 3 4 5 6 7 8 9 10 11 12 13)

Count of Smaller → 0 1 2 2 4 4 4 4 8 8 10 10 10 13

Find Observations

↳ For first occurence of each number the index is equal to smaller count

↳ If I'm equal to prev element then my smaller count will be same as that of prev.

PSEUDO CODE

```
Arrays.sort(arr);
int ans=0, count=0;
for(i=1; i<arr.length; i++){
    if(arr[i] != arr[i-1]){
        count = i;
    }
    if(arr[i] == count){
        ans++;
    }
}
Print(ans);
```

TC - O(N log N)

SC - O(1)

Edge Case :- when A[0] = 0

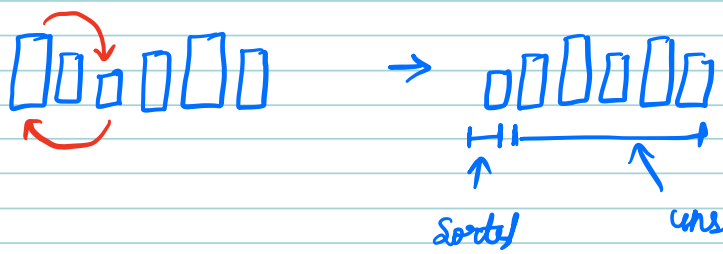
```
if(A[0] == 0){
    ans++
}
```

↳ Before for loop.

SORTING ALGORITHMS

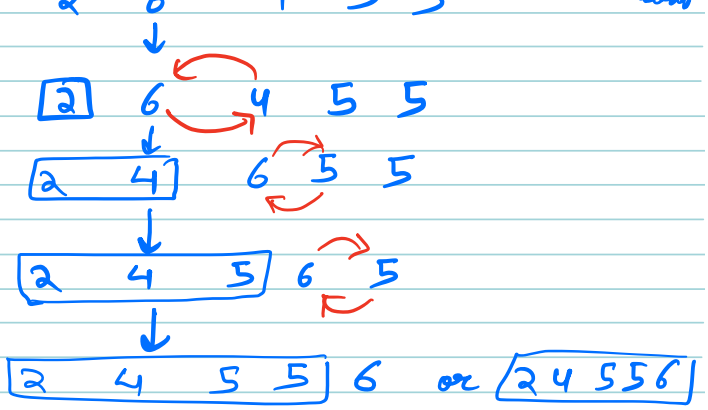
(I) SELECTION SORT

Ex:- Students standing in a row randomly. & we need to arrange in ascending order of height



DRY RUN

Ex:- 2 6 4 5 5 ← Height of Student



PSEUDO CODE:-

```
minIdx = 0
for ( i = 0 ; i < n-1 ; i++){
    minIdx = i ;
```

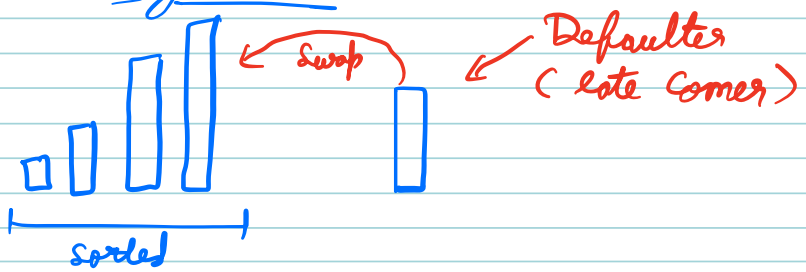
```
for (j = i+1 ; j < n ; j++){
    if (arr[minIdx] > arr[j]){
        minIdx = j;
    }
}
swap ( arr , i , minIdx);
}
```

TC → O(N*N)

SC → O(1)

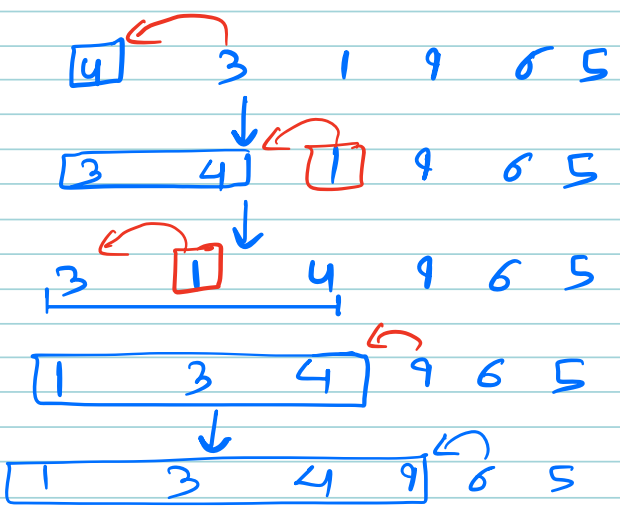
II INSERTION SORTING

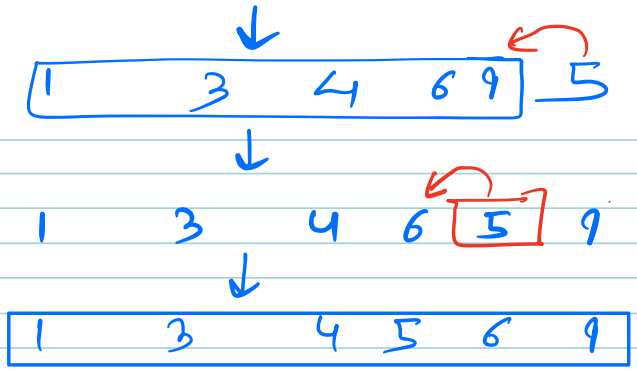
Ex:- Assembly line.



* late comer will keep on swaping with front guy till he reach on correct position.

Ex:-





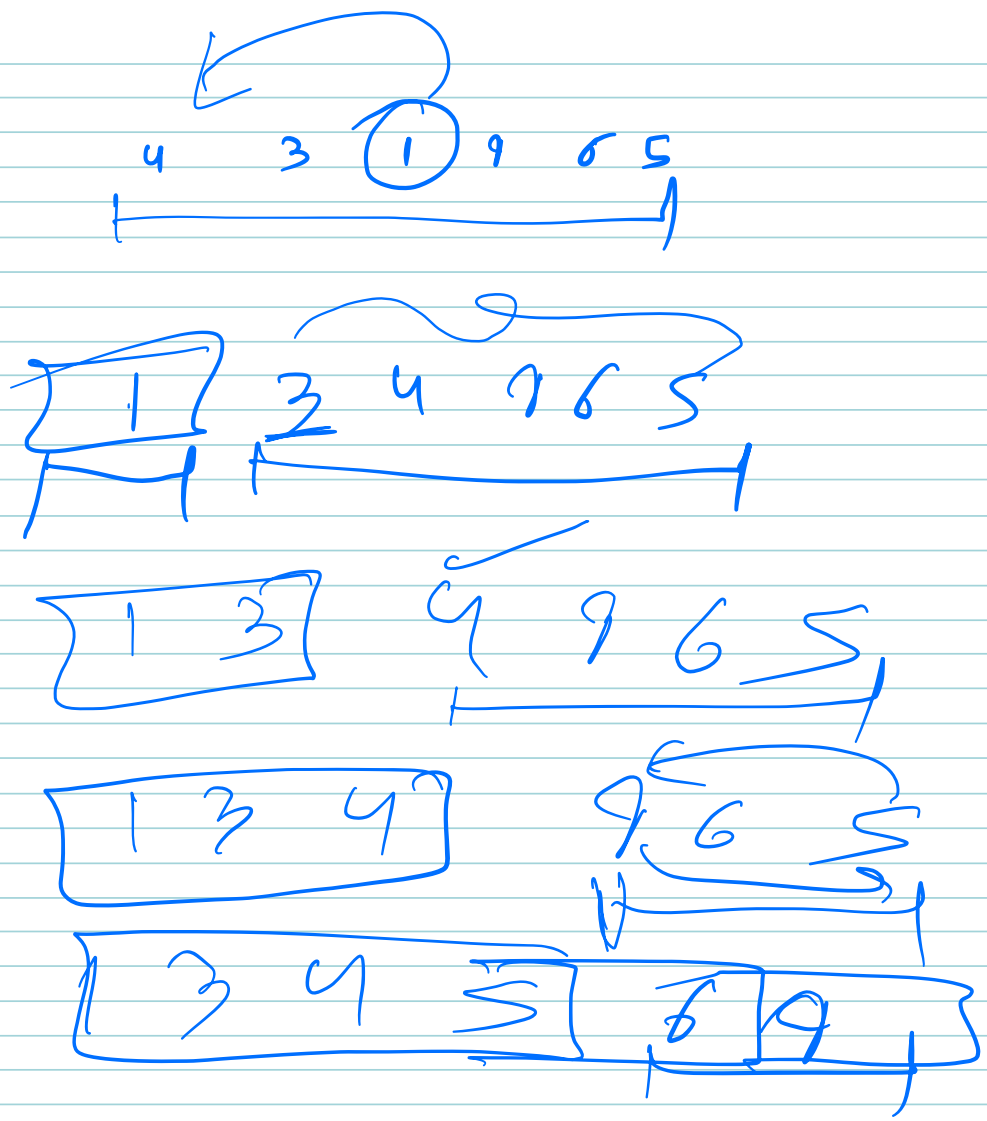
PSEUDO CODE

```
for ( i = 1 ; i < n ; i++ ){
    j = i-1;
    while ( j >= 0 && arr[j] > arr[j+1] ){
        swap ( arr, j, j+1 );
        j--;
    }
}
```

TC → O(N²) ← worst case (Descending order)
↳ O(N) ← Best case (ascending order)

SC → O(1)

CONCLUSION :- Both Selection & Insertion sort are Inplace sorting Algos


4 3 1 9 6 5
1 3 4 9 6 5
1 3 4 9 6 5
1 3 4 9 6 5
3 4 5 6 9