Sorting

TABLE OF CONTENTS

QUIZ

What are the two types of memory with respect to program execution.

Stack
Heap

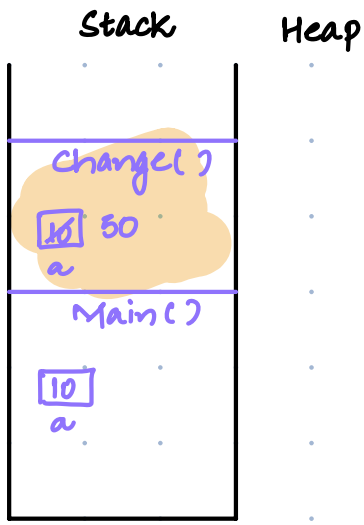
QUIZ

Java

```
static void change(int a) {
    a = 50;
}

public static void main(String args[]) {
    int a = 10;
    change(a);
    System.out.println(a);
}
```

Python

```
def change(a):
    a = 50

def main():
    a = 10
    change(a)
    print(a)

if __name__ == "__main__":
    main()
```



o/p : 10

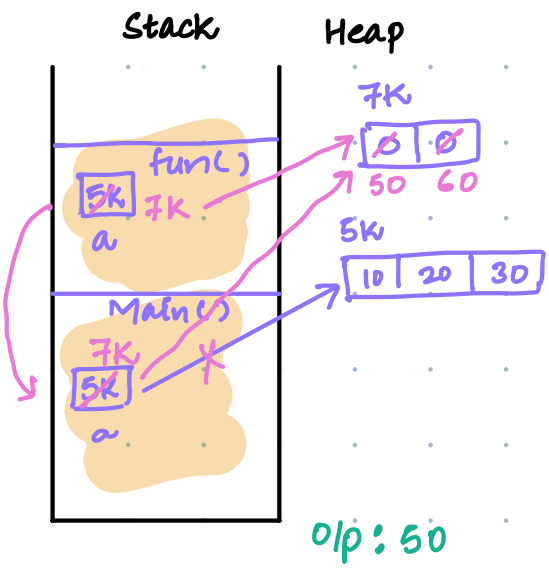
QUIZ

Java

```
static int[] fun(int[]a) {
    a = new int[2];
    a[0] = 50; a[1] = 60;
    return a;
}

public static void main(String args[]) {
    int[]a = {10,20,30};
    a = fun(a);
    System.out.println(a[0]);
}
```

Python

```
def fun(a):
    a = [0, 0]
    a[0] = 50
    a[1] = 60
    return a

def main():
    a = [10, 20, 30]
    a = fun(a)
    print(a[0])

if __name__ == "__main__":
    main()
```



o/p : 50

QUIZ

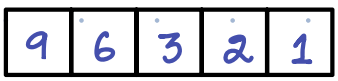
What happens to the call stack when a called function returns ?

Sorting

Sorting is arrangement of data in particular order on the basis of some parameter.

Parameter: Magnitude

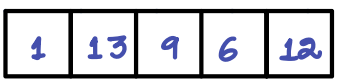
Sorted in Inc order



Parameter: Magnitude

Sorted in Dec order.

QUIZ:



No OF Factors: 1 2 3 4 6

Parameter: No OF factors

Sorted in INC order.

13 → 1, 13

9 → 1, 3, 9

6 → 1, 2, 3, 6

12 → 1, 2, 3, 4, 6, 12

Why sorting?

Sorting is essential for organizing, analyzing, searching, and presenting data efficiently and effectively in various applications and contexts.

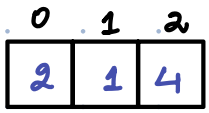
Question (Elements Removal)

Given N elements, at every step remove an array element.

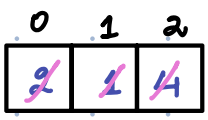
Cost to remove an element = Sum of array of elements present in an array before removal.

Find minimum cost to remove all elements.

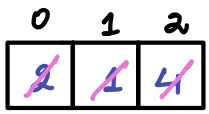
NOTE : First add the cost of removal and then remove it.



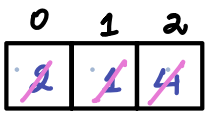
ans : 11



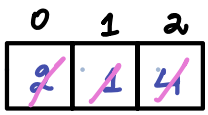
2+1+4 = 7
1+4 = 5
4
16



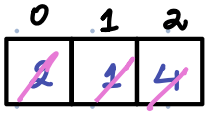
2+1+4 = 7
1+4 = 5
1
13



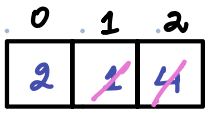
4+1+2 = 7
1+2 = 3
2
12



4+1+2 = 7
2+1 = 3
1
11



2+1+4 = 7
2+4 = 6
4
17



2+1+4 = 7
2+4 = 6
2
15

QUIZ:

0	1	2
4	6	1

$6+4+1=11$

$4+1=5$

1

17

QUIZ:

0	1	2	3
3	5	1	-3

$5+3+1+(-3)=6$

$3+1+(-3)=1$

$1-3=-2$

-3

2

0	1	2	3
a	b	c	d

Remove a → a + b + c + d

" b → b + c + d

" c → c + d

" d → d

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

Min Cost

largest

Smallest

Observations:

Order of elements: $a > b > c > d$

i^{th} index element is occuring $(i+1)$ times in sum.

Contribution of i^{th} element : $(i+1) * a[i]$

Total sum = $\sum$ contribution of all elements

$$= \sum_{i=0}^{N-1} (i+1) * a[i]$$

Approach: Sort the data in DEC order

Iterate on array & add contribution of each term.

```
int MinCost(int arr[], int n){
    ans = 0;
    // Sort in dec order
    reverse_sort(arr);          → N log N

    // Add contribution
    for(i=0; i<n; i++){
        ans += (i+1) * arr[i];
    }
    return ans;
}
```

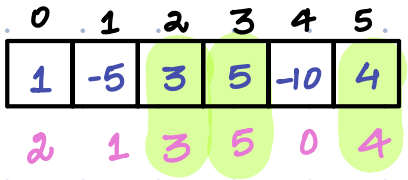
T.C: O(N logN + N)
= O(N logN)

S.C: O(1)

Question (Noble Integers) { Distinct data }

Given N array elements, calculate number of noble integers.

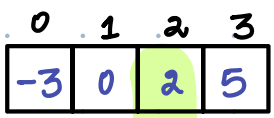
An element ele in arr [ ] is said to be noble if { count of smaller elements = ele itself }



No of smaller elements : 2 1 3 5 0 4

ans : 3

QUIZ



No of smaller elements : 0 1 2 3

ans : 1

BruteForce Solution : For each element, find count of smaller elements

```
int NobleCount( int arr[], int n){
    ans=0;
    for (i=0; i<n; i++){
        count=0;
        for (j=0; j<n; j++){
            if (arr[j] < arr[i]){
                count++;
            }
        }
        if (count == arr[i]){
            ans++;
        }
    }
    return ans;
}
```

T.C : $O(N^2)$

S.C : $O(1)$

Observations:

0	1	2	3	4	5
1	-5	3	5	-10	4

↓ Sort

0	1	2	3	4	5
-10	-5	1	3	4	5

Total No of Smaller elements = index

Optimised Approach: Sort the array in INC order.

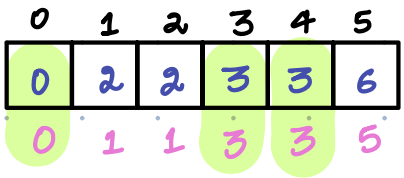
IF arr[i] == i → Noble element

```
int NobleCount( int arr[], int n){

    // sort
      arr.sort();
       ans = 0;

    for (i=0; i<n; i++) {
        if( arr[i] == i) {
            ans++;
        }
    }
    return ans;
}
```

T.C : O(N log N)

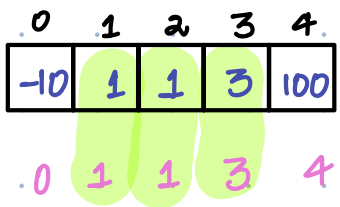
S.C : O(1)

Question (Noble Integers) : { Data can repeat }



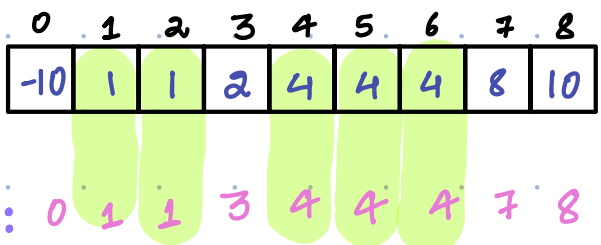
ans : 3

QUIZ!



ans : 3

QUIZ!



ans : 5

Observations:

Case I: If curr element != prev element

No. of smaller elements = index

Case II: If curr element == prev element

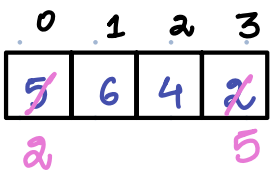
No of smaller elements for curr element = No of smaller elements for prev element.

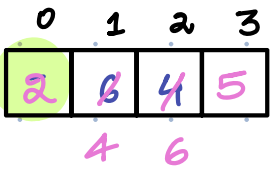
```
int NobleCount( int arr[], int n){
    ans=0;
    // Sort the array
     sort(arr);
     if(arr[0]==0) ans++;
     int count=0;
    // Check for smaller elements
     for(i=1; i<n; i++){
        if( arr[i] != arr[i-1]){
            count = i;
        }

        if(count == arr[i])
            ans++;
     }
    return ans;
}
```

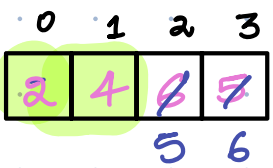
T.C: O(N log N)
S.C: O(1)

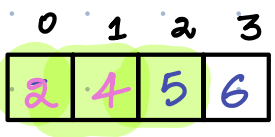
Selection Sort

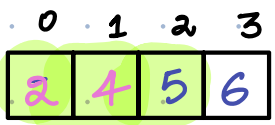
idea : Select the minimum element and send that elements to correct position by swapping.











i	Iterate	Minval	Min_idx	Swap
0	[0 3]	2	3	arr[Min_idx] & arr[0]
1	[1 3]	4	2	arr[Min_idx] & arr[1]
2	[2 3]	5	3	arr[Min_idx] & arr[2]
3	[3 3]	6	3	arr[Min_idx] & arr[3]

< / > Code

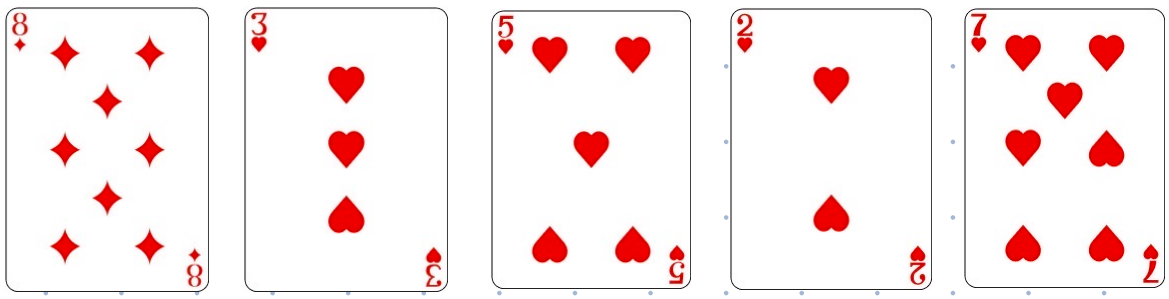
```
void SelectionSort( int arr[] ){
    n = arr.size();

  for(i=0; i<n; i++){
    int min_val = INT.MAX;
    int min_idx = -1;
    // Find Min value
    for(j=i; j<n; j++){
      if(arr[j] < min_val){
        min_val = arr[j];
        min_idx = j;
      }
    }
    // Bring Min val to correct position by swapping
    swap(arr[min_idx], arr[i]);
  }
}
```

T.C : $O(N^2)$

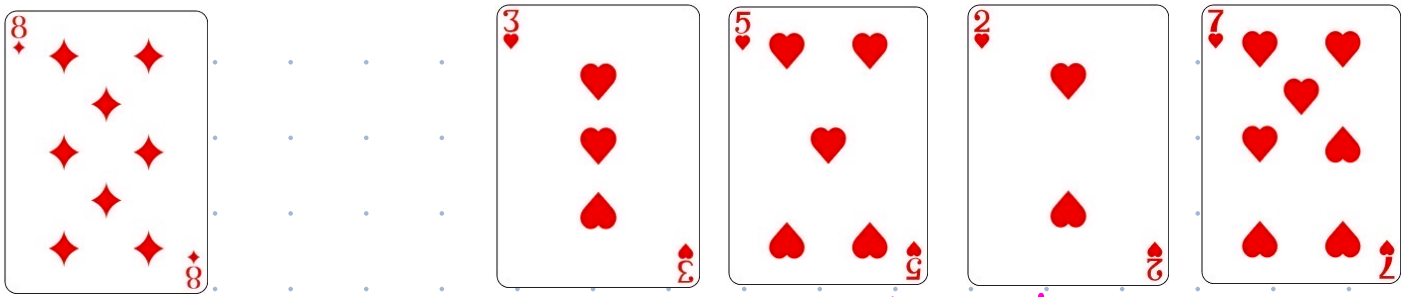
S.C : $O(1)$

Insertion Sort (Arrangement of playing cards)

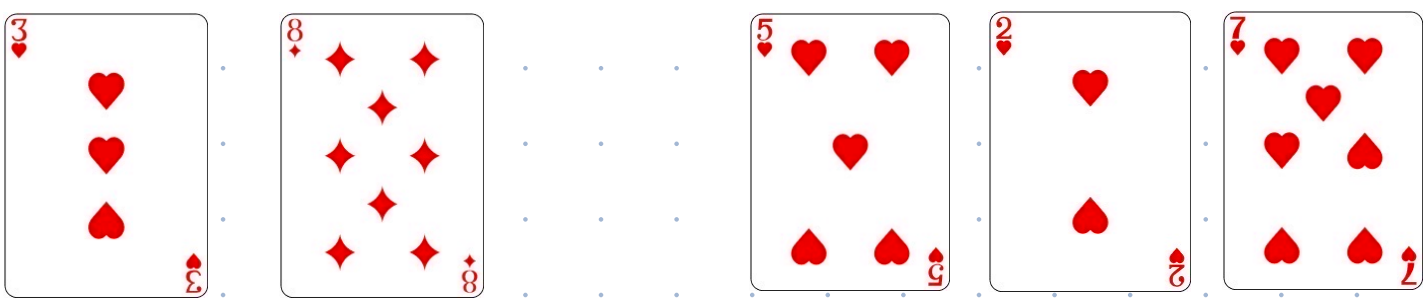
Pick a card from UNSORTED GROUP & put it in right place in SORTED GROUP.



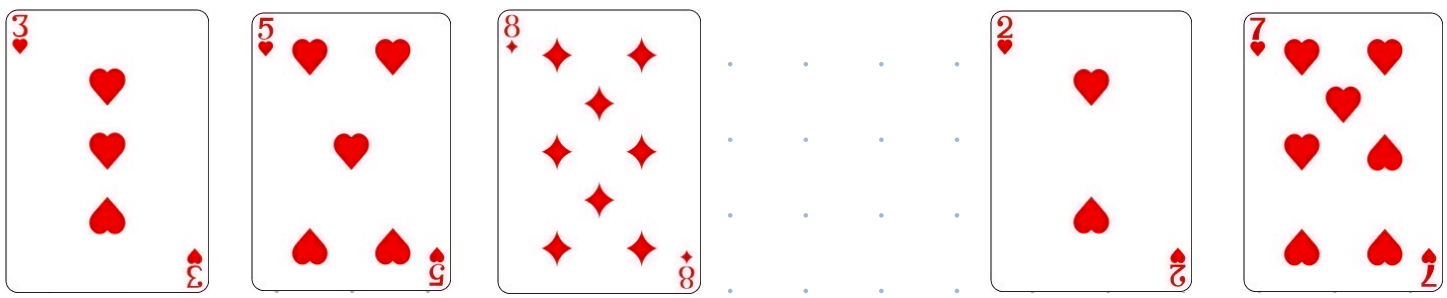
Unsorted



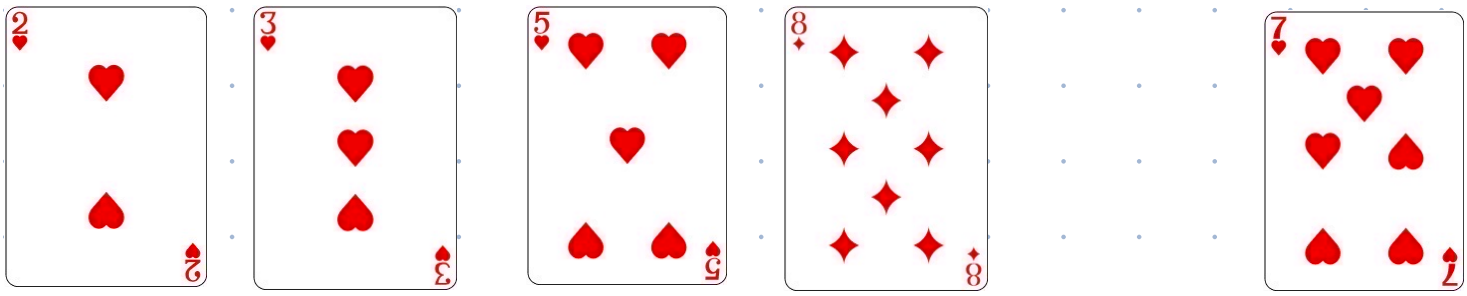
Sorted Unsorted



Sorted Unsorted

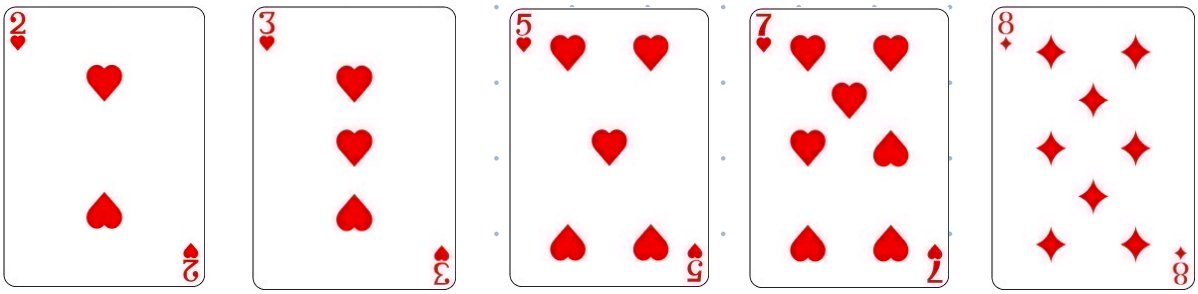


Sorted Unsorted



Sorted

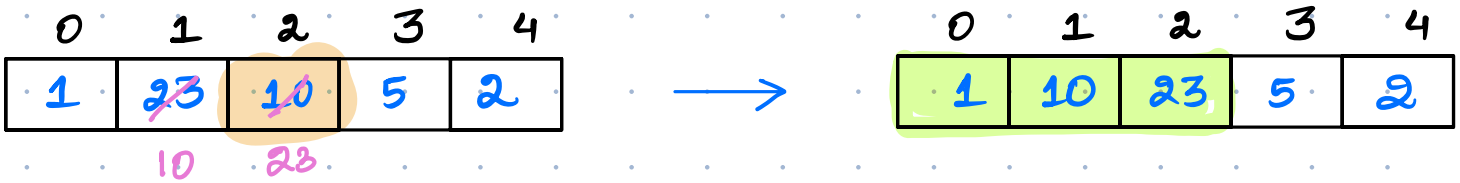
Unsorted



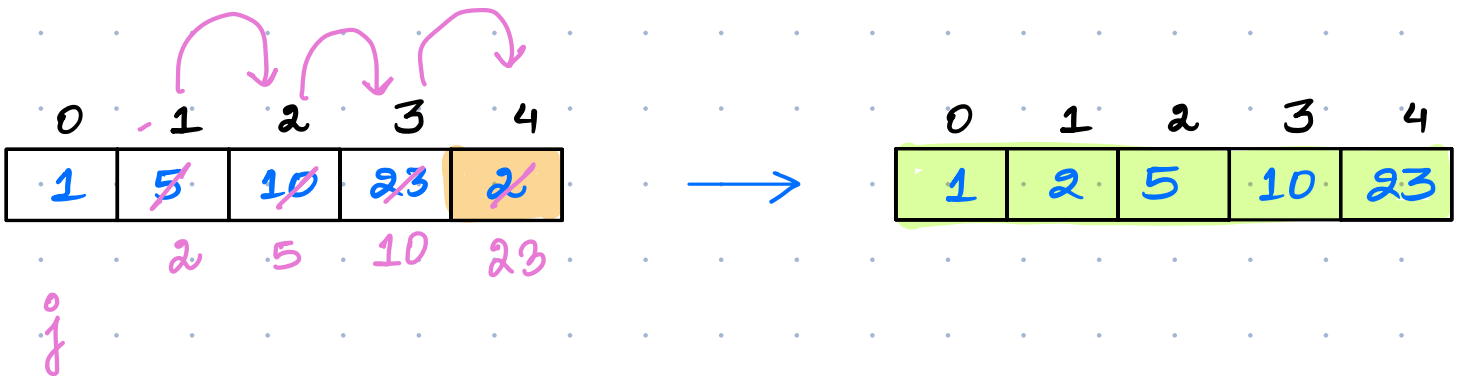
Sorted


0 1 2 3 4
23 1 10 5 2
0 1 2 3 4
23 1 10 5 2


0 1 2 3 4
23 1 0 5 2
1 23
0 1 2 3 4
1 23 10 5 2


0 1 2 3 4
1 23 10 5 2
10 23
0 1 2 3 4
1 10 23 5 2


0 1 2 3 4
1 10 23 5 2
5 10 23
0 1 2 3 4
1 5 10 23 2


0 1 2 3 4
1 5 10 23 2
2 5 10 23
j
0 1 2 3 4
1 2 5 10 23

< / > Code

```
void InsertionSort(int[] arr, int n){

    for(i=1; i<n; i++){
        int key = arr[i];
        int j = i-1;

        while(j>=0 && arr[j]>key){
            arr[j+1] = arr[j];
            j--;
        }

        // Place key at its correct position
        arr[j+1] = key;
    }
}
```

0	1	2	3	4
23	10	5	2	1

T.C: $O(N^2)$ in Reverse case.

0	1	2	3	4
1	2	5	10	23

T.C. : O(N) in Best case (when data is already SORTED)