## Today's agenda

↳ understanding sorting

↳ Problems on sorting

↳ Sorting techniques

**Sorting:** Arranging data in increasing/decreasing.
↓
on what Parameter.

Ex1: 2 4 10 15 27 → true

Ex2: 20 7 3 -5 -8 → dec order → true

Ex3: 1 2 3 7 4 9 6 → Not sorted on the basis of value.
→ sorted on the basis of factor count.

#factors: 1 2 2 2 3 3 4

- bubble sort
- selection sort
- insertion sort
- merge sort
- quick sort
- bucket sort
- radix sort

etc.

Steps to Solve Problem:

1 2 3 4

1 ↓ Sort the array
↓
inbuilt sorting function.
⇓
Arrays.sort(arr); → inc. order
↓
T.c: $O(N\log N)$ S.c: $O(N)$

worst T.c: $O(N^2)$

↳ How in levelup.

## Q) Order of removal

↳ Given n elements at every step remove an array element. Cost to remove element = Sum of array elements present. Find min cost to remove all elements.

Note: Add cost first and then remove.

Ex1: arr[3] = { $\overset{0}{3}$ $\overset{1}{2}$ $\overset{2}{5}$ }

remove 3: 10

remove 2: 7

remove 5: 5

22

arr[3] = { $\overset{0}{\cancel{3}}$ $\overset{1}{\cancel{2}}$ $\overset{2}{\cancel{5}}$ }

remove 2: 10

remove 5: 8

remove 3: 3

21

Ex2: arr[4] = { $\overset{0}{4}$ $\overset{1}{6}$ $\overset{2}{2}$ $\overset{3}{7}$ }

remove 7 : 4 + 6 + 2 + 7

remove 2 : 4 + 6 + 2

remove 6 : 4 + 6

remove 4 : 4

max Contribution : 0th index → min

2nd max Contri : 1st index → 2nd min

⋮

Array should be sorted in inc. order & remove from last.

arr[4] = $\{\overset{0}{4}\ \ \overset{1}{6}\ \ \overset{2}{2}\ \ \overset{3}{7}\}$

↓

$\{\overset{0}{2}\ \ \overset{1}{4}\ \ \overset{2}{6}\ \ \overset{3}{7}\}$

remove 7 : 2 + 4 + 6 + 7

remove 6 : 2 + 4 + $\underbrace{6}_{6*2}$

remove 4 : 2 + $\underbrace{4}_{4*3}$

remove 2 : $\underbrace{2}_{2*4}$

adding frequency = N-i

//Psuedo Code

```
int orderofremoval (int arr[N]){
    Arrays.Sort (arr);
    int ans =0;
    for(int i=0; i<N; i++){
        int temp = arr[i] * (N-i);
        ans= ans + temp;
    }
    return ans;
}
```

T.C: O(NlogN) + O(N)

= O(NlogN + ~~N~~)

= O(N logN)

S.C: O(N)

## Tracing

```
int    orderofremoval (int arr[N]){
    Arrays.sort(arr);
      int ans =0;
   for(int i=0; i<N; i++){
        int temp = arr[i] * (N-i);
          ans= ans + temp;

   }
          return ans;

}
```

arr[4] = { $\overset{0}{4}$ $\overset{1}{6}$ $\overset{2}{2}$ $\overset{3}{7}$ }

↓

$\overset{0}{2}$ $\overset{1}{4}$ $\overset{2}{6}$ $\overset{3}{7}$

ans = 0+8+12+12+7 = 39

| i | temp |
|---|---|
| 0 | 2*4 |
| 1 | 4*3 |
| 2 | 6*2 |
| 3 | 7*1 |

Q) Good Integer { only distinct }

↳ Given arr[N], Calculate no. of good integers.

An element is said to be good if

{No. of element < ele == ele itself }

$\downarrow i \quad == \quad \downarrow arr[i]$

|  | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Ex1 : | -1 | -4 | 3 | 5 | -15 | 4 |
| # less : | 2 | 1 | 3 | 5 | 0 | 4 |

⇒ ans : 3

Ex1 : { -1 -4 3 5 -15 4 } (indices 0 1 2 3 4 5)

⇓ sort in inc. order

{ -15 -4 -1 3 4 5 } (indices 0 1 2 3 4 5)

Count of elements smaller arr[i] = i

element itself = arr[i]

//Psuedo Code

```
int goodintegers (int arr[N]){
        Arrays. sort (arr);
        int Count=0;

        for (int i=0; i<N; i++){
            if (arr[i]==i) {Count ++; }
        }
        return Count;
}
```

T.C: $O(N\log N + N)$

$= O(N\log N)$

S.C: $O(N)$

{No. of element < ele == ele itself}

$\downarrow$ i == $\downarrow$ arr[i]

## Good Integers : { Data can repeat } → last idea won't work directly.

Ex1: { 0 2 2 3 3 8 } → ans = 3

| | | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| Ex1: | 0 | 2 | 2 | 3 | 3 | 8 |
| # less : | 0 | 1 | 1 | 3 | 3 | 5 |

Ex2: { -4 -2 3 3 5 5 5 5 8 8 8 10 17 }

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Ex2: | -4 | -2 | 3 | 3 | 5 | 5 | 5 | 5 | 8 | 8 | 8 | 10 | 17 |
| #less : | 0 | 1 | 2 | 2 | 4 | 4 | 4 | 4 | 8 | 8 | 8 | 11 | 12 |

↳ ans : 3

Obs1 : if element is the first occ. if (arr[i] != arr[i-1])

⇓

i == arr[i]

count of ele < ele

Obs2 : if element is the repeat element.

↳ count of ele < ele will remain same as count of first occ.

Note : To check the first occ : arr[i] != arr[i-1]

//PSuedo Code

```
int  goodIntegers (int arr[N]){
        Arrays. sort (arr);
        int  Count=0;

        int  less Count = 0;
        //H.w -> handle for 0th inden
     for (int i=1; i<N; i++){
            if (arr[i] != arr[i-1]){
                  less Count = i;
            }
            else{ x
                  //Nothing
            }

         if (arr[i]== less Count){
                Count ++;
         }
     }

}
```

T.C: O(NlogN)

S.C: O(N)

```
int Count=0;

int less Count =0;

for (int i=1; i<N; i++){
        if (arr[i] != arr[i-1]){
                less Count = i;
        }
        else{ ×
                //nothing
        }

      if (arr[i]== less Count){
              Count ++;
      }
}
```

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ex2: { | -4 | -2 | 3 | 3 | 5 | 5 | 5 | 5 | 8 | 8 | 8 | 10 | 17 | } |
| #less: | 0 | 1 | 2 | 2 | 4 | 4 | 4 | 4 | 8 | 8 | 8 | 11 | 12 |  |

i

Count=0

less count: 0

| i | less count |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |
| 4 | 4 |
| 5 | 4 |
| 6 | 4 |
| 7 | 4 |
| 8 | 8 |

Break till 9:50 Pm

## //Sorting techniques

### ① Bubble sort

↳ Sort the array in asc. order but we can swap adjacent elements only.

```
                0   1   2   3   4    5   6   7
arr[8]: {       5   7   5   4   10  -2   6   3 }
```

iter 0: { 5 ~~7~~ ~~5~~ ~~4~~ ~~10~~ ~~-2~~ ~~6~~ ~~3~~ } → {0, N-2}

5 ~~7~~4 7 -2 ~~10~~6 ~~10~~3 10

{ 5 5 4 7 -2 6 3 10 }

↙i

iter 1: { 5 ~~5~~ ~~4~~ ~~7~~ ~~-2~~ ~~6~~ ~~3~~ 10 } → {0, N-3}

4 5 -2 ~~7~~6 ~~7~~3 7

.

.

.

iter~~4~~ :

↳ N-1 iterations

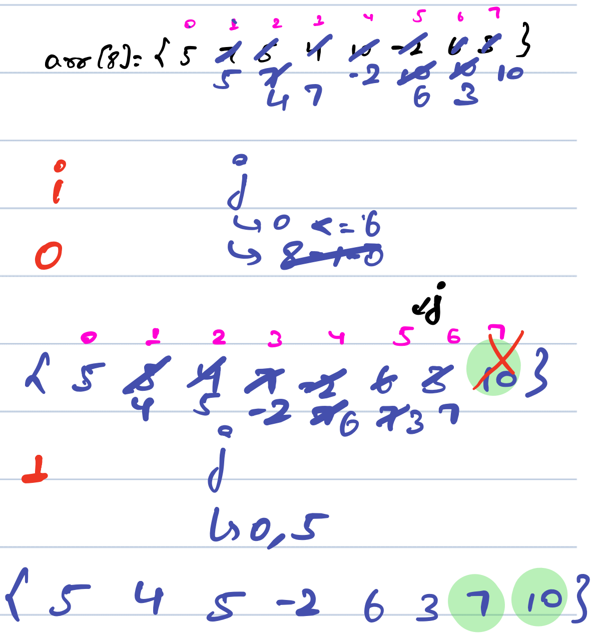
//Psuedo Code

```
void bubbleSort(int arr[n]){
    for(int i=0; i<N-1 ; i++){ //N-1 iterations
        for(int j=0; j<N-1-i;j++){
            if(arr[j] >arr[j+1]){
                int temp= arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

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

S.C : $O(1)$

```
void  bubblesort (int arr[n]){
                    i<=N-2
    for(int i=0;  i<N-1 ; i++){ //N-1 iterations
                         j<=N-2-i
        for (int j=0; j<N-1-i;j++){
            if (arr[j] >arr[j+1]){
                int temP: arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temP;
            }
        }
    }
}
```



N elements

N-1 elements fix

Nth will be fixed auto

[0,N-2] → N-1 iteration

you didn't come this far only to come this far.

$-2^{31}$ $2^{31}-1$

⇓

$2^{31}$