Today's agenda

- understanding Sorting
- Problems on Sorting
- Sorting techniques

Sorting: Arranging data in increasing / decreasing.
Wrt what Parameter.

Ex1: 2 4 10 15 27 → true

Ex2: 20 7 3 -5 -8 → true

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

#factors: 1 2 2 2 3 3 4 → sorted on the basis of factor count.

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

→ overload

arr[4]: 3 1 9 8

↓

Arrays.sort(arr); → inc. order.

↳ T.C: O(NlogN)

↓

arr[4]: 1 3 8 9

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] = {3 2 5}

Remove 3: 10
Remove 2: 7
Remove 5: 5
22

arr[3] = {3 2 5}

Remove 2: 10
Remove 5: 8
Remove 3: 3
21

Ex2: arr[4] = { 4 6 2 7 }

Remove 7: 4 + 6 + 2 + 7
Remove 2: 4 + 6 + 2
Remove 6: 4 + 6
Remove 4: 4

minimize overall cost

max contribution = 0th index = min ele
2nd max contribution = 1st index = 2nd min ele.
3rd min ele.

-> Array should be sorted in inc. order.

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

↓

{ 2 4 6 7 }
0 1 2 3

4 times 3 times 2 times 1 time

→ removal for i : N-i

//Pseudo code

```
int removalOrder (int arr[N]){
    Arrays.sort(arr);
    int ans = 0;
    for(int i=N-1; i>=0; i--){
        ans = ans + (arr[i]*(N-i));
    }
    return ans;
}
```

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

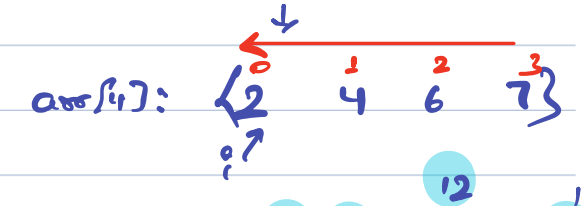
S.C: O(1)

```
int removalOrder(int arr[N]){
    Arrays.sort(arr);
    int ans = 0;

    for(int i=N-1; i>=0; i--){
        ans = ans + (arr[i]*(N-i));
    }

    return ans;
}
```

arr[4]: 4 6 2 7



ans = 0 + 7 + 12 + 12

8 + 12 + 12 + 7 = 39

i = 3 : 2 + 4 + 6 + 7

i = 2 : 2 + 4 + 6

i = 1 : 2 + 4

i = 0 : 2

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}

i → arr[i]

Ex1: { -1 -4 3 5 -15 4 } ⇒ ans = 3

idx: 0 1 2 3 4 5

#less: 2 1 3 5 0 4

//idea1

↳ for every number, count smaller than that number and check if they are same. {nested loop}.

T.C: $O(n^2)$ S.C: $O(1)$

//idea2

↳ { -1 -4 3 9 -15 7 }

idx: 0 1 2 3 4 5

{ -15 -4 -1 3 7 9 }

idx: 0 1 2 3 4 5

After sorting, count of elements smaller than arr[i] = i

//Pseudo 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(NlogN)

S.C: O(1)

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

Good Integers : {Data can repeat}

Ex1: { 0 2 2 3 3 8 } → ans: 3

(0) (1) (2) (3) (4) (5)

#less : 0 1 1 3 3 5

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

(0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12)

edge case → (0), i → (12)

#less : 0 0 2 2 4 4 4 4 8 8 8 11 12

After sorting

↳ ans: 5

Obs1: All the occurrences of same number will be either good or bad.

Obs2: if ele is 1st occ. → arr[i] != arr[i-1]

⇓

No. of ele < ele == i

i == arr[i]

Obs3: if Ele is not the 1st occ.

//Pseudo code

```
int goodintegerduplicate (int arr[N]){
    Arrays.sort (arr);
    int count = 0;

    int lesscount = 0;
    for (int i=1; i<N; i++){
        if (arr[i] != arr[i-1]){ //first occ.
            lesscount = i;
        }
        else{ // next occ.

        }

        if (arr[i] == lesscount) { count ++; }
    }

}
```

T.C: O(N log N)

S.C: O(1)

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

less: 0 0 2 2 4 4 4 4 8 8 8

```
int goodIntegerDuplicate (int arr[N]){
    Arrays.sort(arr);
    int count = 0;
    if(arr[0]==0){count++;}
    int lessCount = 0;
    for(int i=1; i<N; i++){
        if(arr[i] != arr[i-1]){ //first occ.
            lessCount = i;
        }
        else{ // next occ.
            -> nothing
        }
        if(arr[i] == lessCount){count++;}
    }
}
```

count = 0 2 8 4 5

lessCount = 0 2 4 8

//Sorting techniques

① Bubble Sort

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

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

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

{ 5 5 4 7 -2 6 3 10 }

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

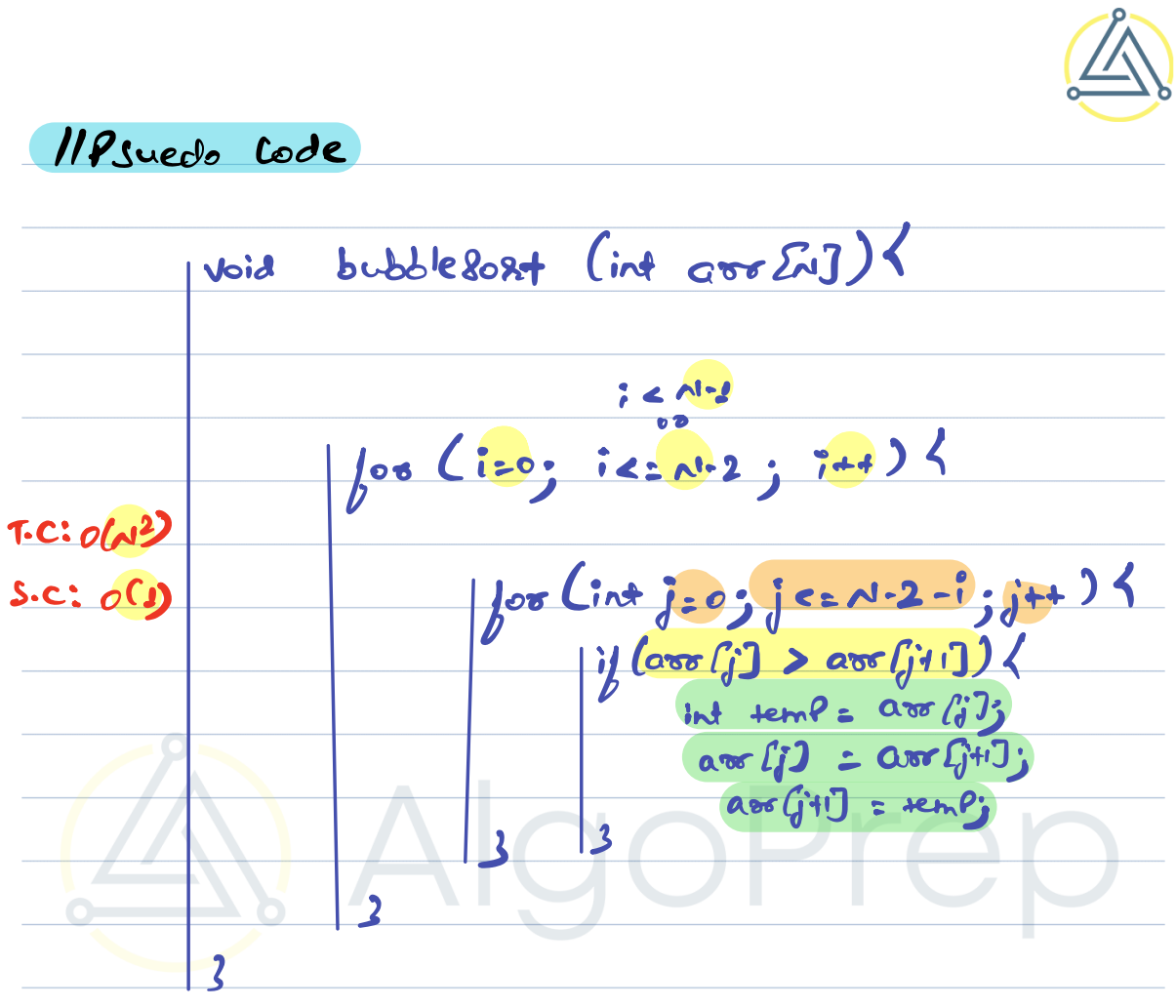
{ 5 4 5 -2 6 3 7 10 }

iter 2 : {0, N-4}

iter 6

iter 7 ??

↳ N length → N-1 iteration
{0, N-2}


//Pseudo code
void bubbleSort (int arr[n]){
for (i=0; i<=n-2; i++){
for (int j=0; j<=n-2-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)
i<n-1

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

```
void bubbleSort (int arr[n]){
    for (i=0; i<=n-2 ; i++){
        for (int j=0; j<=n-2-i ; j++){
            if (arr[j] > arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

i<=n-2

i=0

ei

j -> {0, 6}

{ 5 7 5 4 10 -2 6 3 }
0 1 2 3 4 5 6 7

5 7 4 7 -2 6 6 10
3

{ 5 5 4 7 -2 6 3 10 }

i=1

j -> {0, 5}

{ 5 5 4 7 -2 6 3 10 }
0 1 2 3 4 5 6 7

4 5 -2 6 3 7

{ 5 4 5 -2 6 3 7 10 }