Greedy

↓

Maximise profit / Minimise the loss

Iphone 15 pro max →
- Flipkart [1.6 L]
- Amazon [1.5 L]
- Paytm [1.39 L]

Company.

- 22 LPA
- 25 LPA
- 28 LPA

Q) There is a limited time sale going on for toys.

A[i] → sale end time for i^{th} toy.

B[i] → Beauty of i^{th} toy

Time starts with T = 0 & it takes 1 unit of time to buy one toy & toy can only be bought if $T < A[i]$.

Buy toys such that sum of beauty of toys is maximized.

A[] → [3 1 3 2 3]
(0 1 2 3 4)

B[] → [6 5 3 1 9]

T = ~~0~~ 1 2 3

beauty = 0 + 9 + 6 + 3 = 18. ✗

ans = 20

A[] → [1, 2]
(0 1)

B[] → [3, 1500]

ans = 1503.

idea → Buy everything → ascending order of time.

A[] → [1 3 3 3 5 5 5 8]
(0 1 2 3 4 5 6 7)

B[] → [5 2 7 1 4 3 8 1]

T = ~~0~~ 1 2 ~~3~~ 4 ~~5~~ 6

ans = 0 + 5 + 7 + 4 + 3 + 8 + 1 = 28.

idea → min PQ / min Heap

A[] → [3 (0) 1 (1) 3 (2) 2 (3) 3 (4)]

B[] → [6 5 3 1 9]

sort ↓

sale-end time →	1	2	3	3	3
beauty →	5	1	3	6	9

3, 2

T = 0

5	6
9	

ans = 20

#code. →

① Sort them on the basis of sale end time in increasing order.

② minPriorityQueue pq;

T = 0

③ for(i = 0; i < N; i++){

if(T < A[i]){

pq.insert(B[i]);

T++;

}

else{

if(pq.getMin() < B[i]){

pq.removeMin();

pq.insert(B[i]);

}

}

}

T.C → O(NlogN)

S.C → O(N)

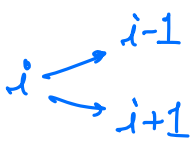
correcting an incorrect decision taken in the past.

④ Remove all elements from pq & keep on adding them to get ans.

Q: There are N students with their marks.
Teacher has to give them candies such that

a) Every student should have atleast one - candy.
b) Student with more marks than any of his/her neighbours have more candies than them.

Find minimum candies to distribute.



A[] → [1 5 2 1]
0 1 2 3

ans = 7.

[1 3 2 1]

A[] → [4 4 4 4 4]
0 1 2 3 4

[1 1 1 1 1]

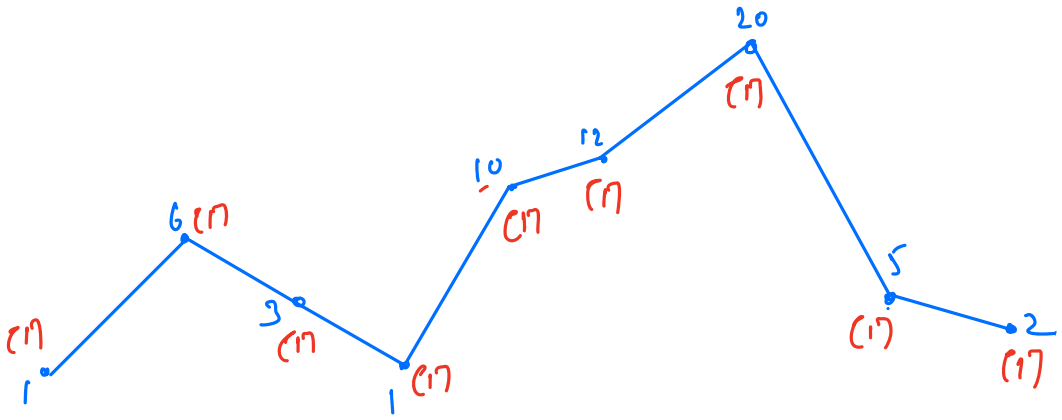
ans = 5.

A[] → [8 10 6 2]
0 1 2 3

ans = 7

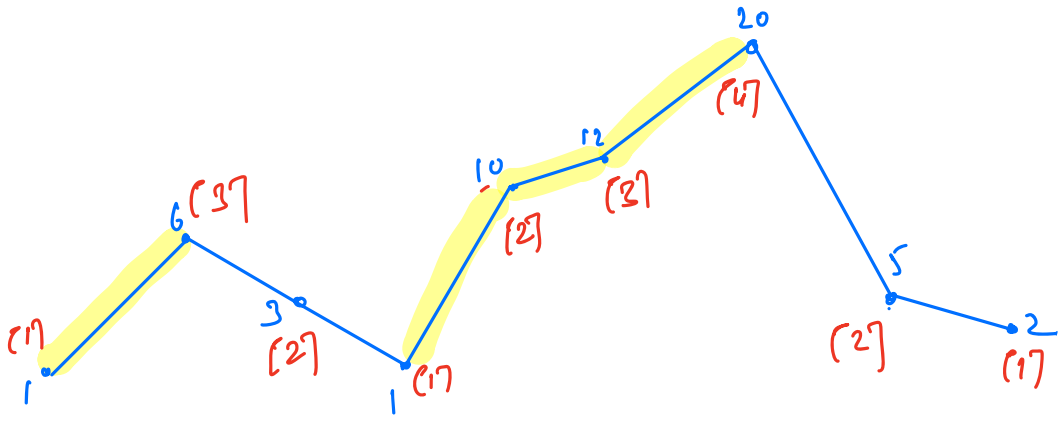
[1 3 2 1]

Arr → [1 6 3 1 10 12 20 5 2]
0 1 2 3 4 5 6 7 8



B.f. → for every element, find no. of continuous decreasing elements on l.h.s and continuous decreasing elements on r.h.s.

T.C → O(N²).



ans = 19.

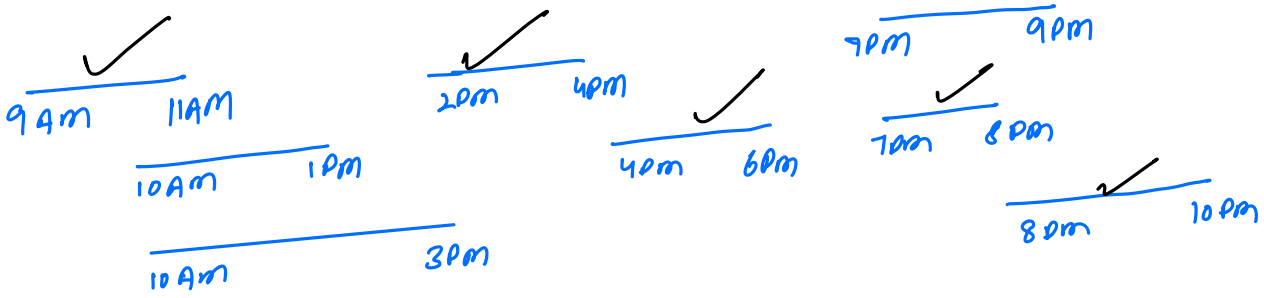
code...

```
int c[N];    ∀i, c[i]=1;

for(i=1; i<N; i++){
    if(arr[i] > arr[i-1]){
        c[i] = c[i-1]+1;
    }
}

for(i=N-2; i>=0; i--){
    if(arr[i] > arr[i+1] && c[i] < c[i+1]+1){
        c[i] = c[i+1]+1;
    }
}

ans=0;
for(i=0; i<N; i++){
    ans += c[i];
}
return ans;
```

T.C → O(N)
S.C → O(N)

Q1 Given N jobs with their start & end-time. Flipkart

find the max jobs that can be completed if only one job can be done at a time.

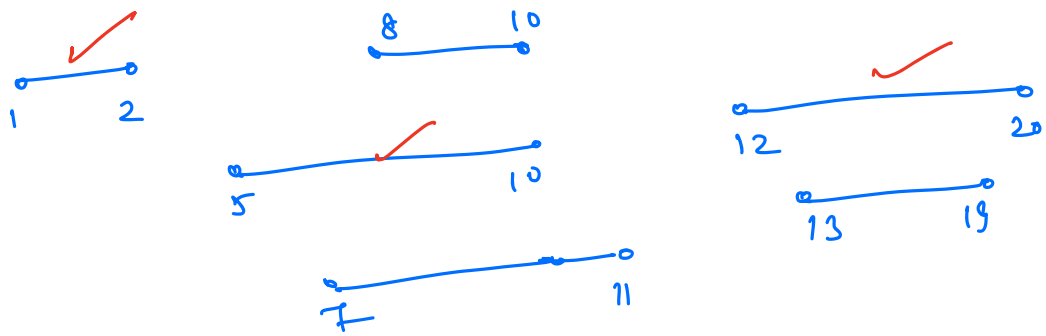


ans = 5

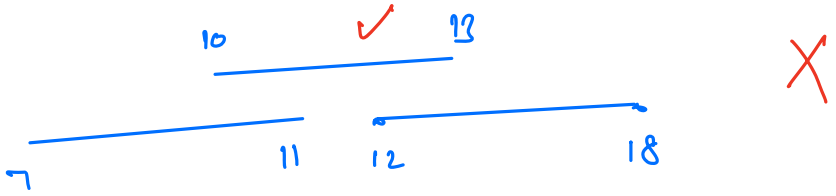
s[] = [1 5 8 7 12 13]

e[] = [2 10 10 11 20 19]

ans = 3

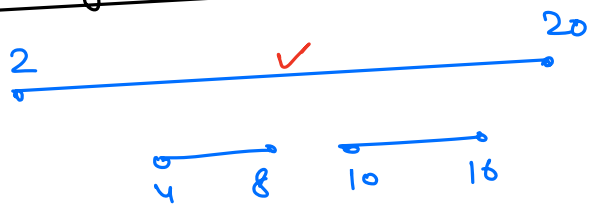


① Sort on basis of duration



✓ ✗

② sort on the basis of start-time →



✓ ✗

start early + minimum duration = end early.

③ Sort on the basis of end-time →

s[] → [1 5 8 7 12 13]

e[] → [2 10 10 11 20 19]

↓ Sort

s[] → [1 5 8 7 13 12]

e[] → [2 10 10 11 19 20]

ans = 3.

s[7] → [10 7 12]

e[7] → [13 11 18]

↓ Sort on the basis of end-time

s[7] → [7 10 12]

e[7] → [11 13 18]

ans = 2.

#code →

① Sort on the basis of end-time in ascending order.

② count = 1, lastEndTime = e[0];

③ for(i = 1; i < N; i++){

```
    if( s[i] ≥ lastEndTime) {
        count++;
        lastEndTime = e[i];
    }
}
```

④ return count;

[T.C → O(N log N)
S.C → O(N)]

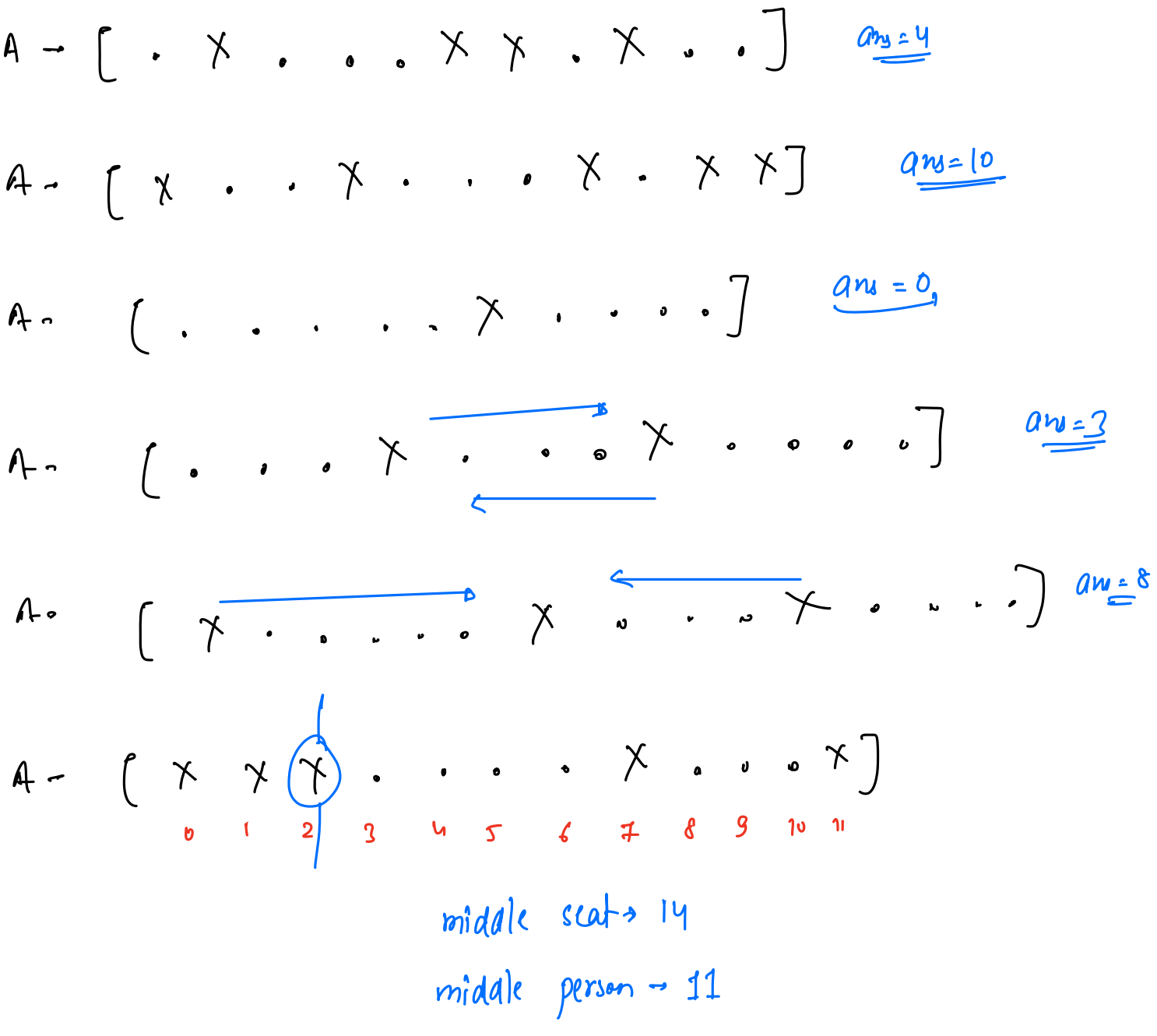
Seats -

There is a row of seats represented by string A. Assume that it contains N seats adjacent to each other. There is a group of people who are already seated in that row randomly. i.e. some are sitting together & some are scattered.

An occupied seat is marked with a character 'x' and an unoccupied seat is marked with a dot ('.')

Now your target is to make the whole group sit together i.e. next to each other, without having any vacant seat between them in such a way that the total number of hops or jumps to move them should be minimum.

In one jump a person can move to the adjacent seat (if available).



idea → Choose the middle person.

ans = 4

middle → 7

code → #todo

Greedy → min, max, sort, exploring just one path.

```
Pair {
    int st;
    int et;
}

Pair[] arr = new Pair[N];
for(i=0; i<N; i++){
    arr[i] = new Pair(s[i], e[i]);
}
```

```
Arrays.sort( arr , new Comparator<Pair>(){

    public int compare( Pair one, Pair two){
        return one.end - two.end;
    }

});
```

[Backtracking - Revise Recursion]