Lecture :- Greedy Algorithms (Approaches)

Agenda

- Introduction
- Toys
- Distribute candies
- finish max jobs.

Qu! There is a limited time sale going on:

A[i] = sale end time of ith toy.

B[i] = beauty of ith toy.

Time starts from t=0, and it takes 1 unit of time to buy each toy and toy can be bought only if T < A[i].

Buy toys such that beauty is maximised. [Medium - Hard]

Eg:

sale end A[] = 0: 3, 1: 1, 2: 3, 3: 2, 4: 3

beauty B[] = 6, 5, 3, 1, 9

t = ∅ ✗ ✗ 3

toy	beauty
0	6
2	3
4	9
	18

Idea: Buy most expensive toy first [Wrong]

A[] = 0: 3, 1: 1, 2: 3, 3: 2, 4: 3

B[] = 6, 5, 3, 1, 9

t = ∅ ✗ ✗ 3

toy	beauty
4	9
0	6
2	3
	18

Wrong ans

Contradiction

A[] = [0: 1, 1: 2]

B[] = [3, 1500]

t = ∅ 1

0	3
1	1500
	1503

What about this?

	0	1	2	3	4
A[] =	3	1	3	2	3
B[] =	6	5	3	1	9

t = ~~0~~ ~~1~~ ~~2~~ 3

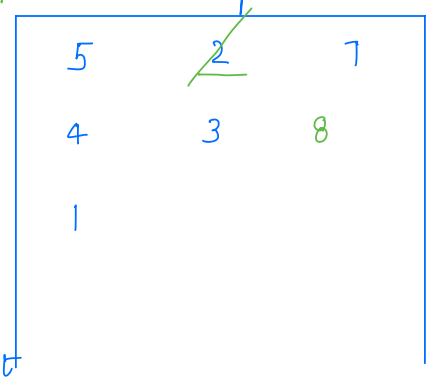
toy	beauty
1	5
4	9
0	6
	20

Idea: Buy everything ⟶ asc order of time

	0	1	2	3	4	5	6	7
A[] =	1	3	3	3	5	5	5	8
B[] =	5	2	7	1	4	3	8	1

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

Min heap



ans = 5 + 7 + 4 + 3 + 8 + 1 = 28 Ans

Correcting an incorrect item bought in past

↳ Min heap

	0	1	2	3
A[] =	3	3	3	3
B[] =	4	8	12	16

t = ~~0~~ ~~1~~ ~~2~~ 3

toy	beauty
0	4
2	12
1	8

Correct :- 16, 12, 8

Code:

```
class Pair {
    int e;
    int beauty;
    Pair (x, y) {
        e = x;
        beauty = y;
    }
}

int maximiseBeauty ( A[], B[] ) {
    n = A.length;
    // Sort in asc order of time
    Pair[] arr = new Pair[n];
O(n) —— for(i=0; i<n; i++) {
        arr[i] = new Pair( A[i], B[i] );
    }
O(nlogn) — Arrays.sort( arr, (u, v) → u.e - v.e) );
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    int t = 0;
O(nlogn) —— for(i=0; i<n; i++) {
        int e = arr[i].e;
        int b = arr[i].beauty;
```

```
        if( t < e) {
            pq.add(b);
            t++;
        } else {
            if( b <= pq.peek()) {
                continue;
            } else {
                pq.poll(); — O(log n)
                pq.add(b);
            }
        }
        int ans=0;
        while( !pq.isEmpty()) { — (n log n)
            ans += pq.poll();
        }
    return ans;
}
```

TC: O(n log n)

SC: O(n)

Qu. There are n students with marks.

Teacher has to distribute candies such that —

a.) Every student should have atleast one candy.

b) Student with more marks than neighbours should have more candies.

find min no. of candies to distribute. [Medium]

A[] = [1 5 2 1]

1 ~~1~~ 2 3 ~~1~~ 2 1

ans = 1 + 3 + 2 + 1 = 7

A[] = [8 10 6 2]

1 ~~1~~ ~~2~~ 3 ~~1~~ 2 1

ans = 1 + 3 + 2 + 1 = 7 Ans

A[] = [4 4 4 4 4]

1 1 1 1 1

ans = 5

A[] = [1 6 3 1 10 12 20 5 2]

1 ~~1~~ ~~2~~ 3 ~~1~~ 2 1 ~~1~~ 2 ~~1~~ 3 ~~1~~ 4 ~~1~~ 2 1

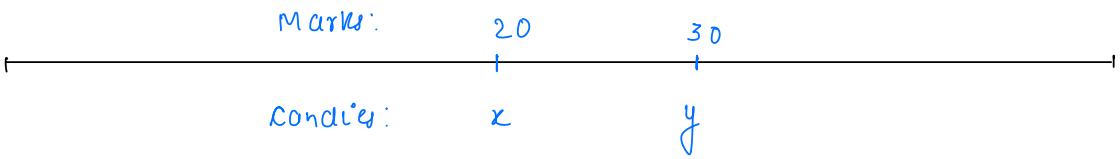
ans = 1 + 3 + 2 + 1 + 2 + 3 + 4 + 2 + 1 = 19 Ans

Logic:

A[] = [1 6 3 1 10 12 20 5 2]

	1	~~6~~	1	1	~~10~~	~~12~~	~~20~~	1	1
L → R	1	~~2~~	~~3~~	1	2	3	4	~~5~~	1
R → L	1	3	2	1	2	3	4	2	1

Why this works?



L → R: $x < y$

R → L $x < y + \underline{delta}$

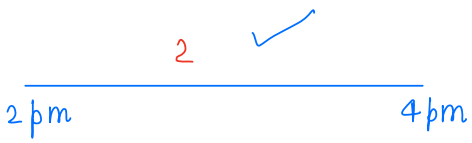
Code:

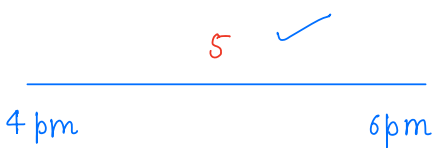
```
int minimiseCandies(A[]) {
    n = A.length;
    int[] res = new int[n];

    for(i=0; i<n; i++){
        res[i] = 1;
    }

    L → R

    for(i=1; i<n; i++) {
        if(A[i] > A[i-1]) {
            res[i] = res[i-1] + 1;
        }
    }

    R → L

    for(i=n-2; i>=0; i--) {
        if(A[i] > A[i+1]) {
            if(res[i] <= res[i+1]) {
                res[i] = res[i+1] + 1;
            }
        }
    }

    int ans = 0;
    for(int el: res) {
        ans += el;
    }
    return ans;
}
```

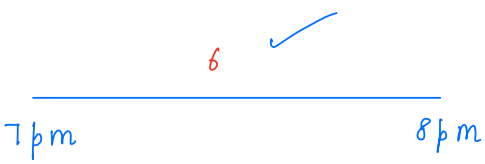
TC: O(n)
SC: O(n)

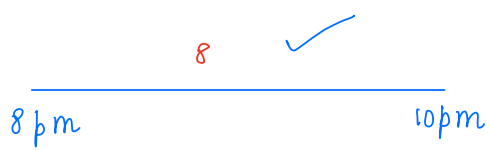
Break: 8:17 – 8:28 AM

Qu: Given n jobs with their start and end time. find max jobs that can be completed if only one job can be done at a time. [Medium]









ans = 5

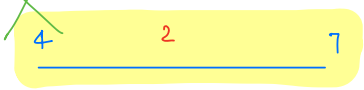
ip format

	0	1	2	3	4	5
s[] =	1	5	8	7	12	13
e[] =	2	10	10	11	10	19

Greedy ideas

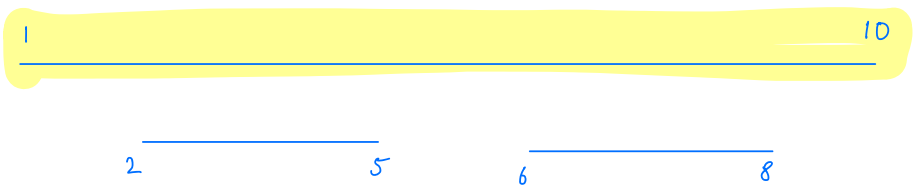
1. Shortest duration
2. Shortest start time
3. " end time

1.) Shortest duration. ✗

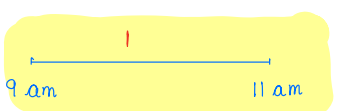


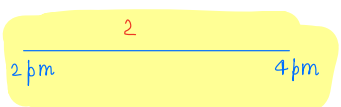
ans = 1 [wrong]

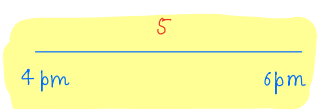
2.) Early start time ✗

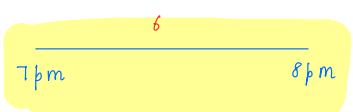


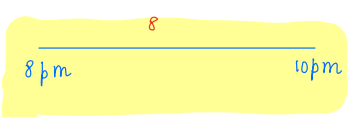
3.) Early end time :- (start time + duration) less











ans = 5

	0	1	2	3	4	5
s[] =	1	5	8	7	12	13
e[] =	2	10	10	11	13	19

lastjob Ended = ~~2~~ ~~10~~ ~~8~~ 19

ans = ~~1~~ ~~2~~ ~~3~~ 4

Code:

```
class Pair {
    int s;
    int e;
    Pair(x, y) {
        s = x;
        e = y;
    }
}

int maximiseJobs(s[], e[]) {
    Pair[] arr = new Pair[n];
    for(i=0; i<n; i++) {          // O(n)
        arr[i] = new Pair(s[i], e[i]);
    }
    Arrays.sort(arr, (u, v) -> u.e - v.e));   // O(nlogn)
```

```
        ans = 1;
        lastjobEnded = arr[0].e;
O(n) ——— for(i=1; i<n; i++){
            pair p = arr[i];
            if(p.s >= lastjobEnded){
                ans += 1;
                lastjobEnded = p.e;
            }
        }
        return ans;
}
```

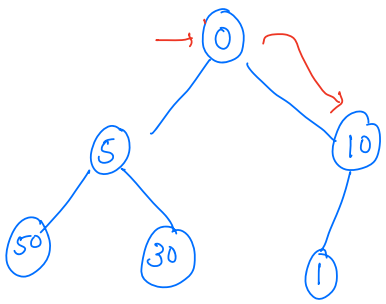
TC: O(nlogn)

SC: O(n)

Greedy : लालची

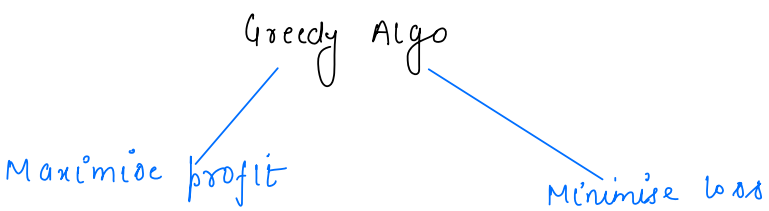
↳ Approach to solve optimisation problems by making greedy | optimal choices

Ex:



Max sum from root to leaf

Greedy does not work.



Thankyou :)