- What is Greedy ?
- Free Cars
- Candy Distribution
- Maximum Jobs

Contest 4

LL, Queues

Reattempt -> 1 more

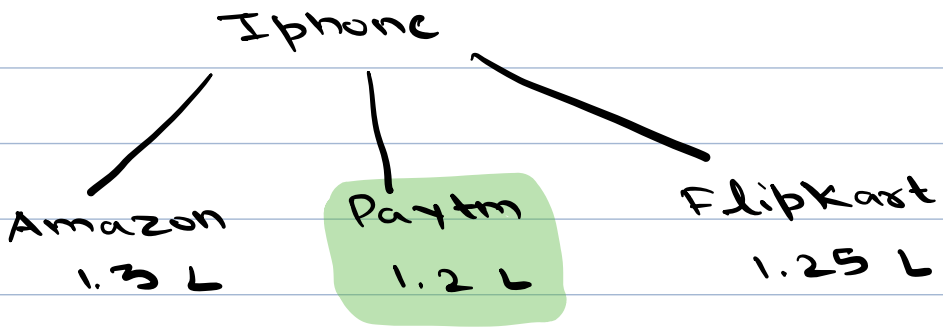
Contest -> 19 Jan

Trees, Heaps and Greedy

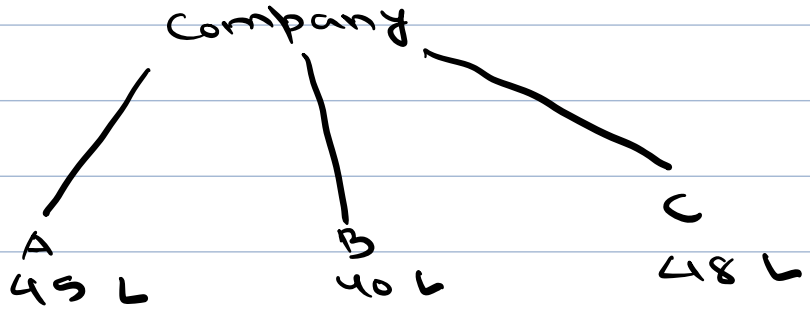
Greedy

↓

Maximize our profit and minimizing our loss

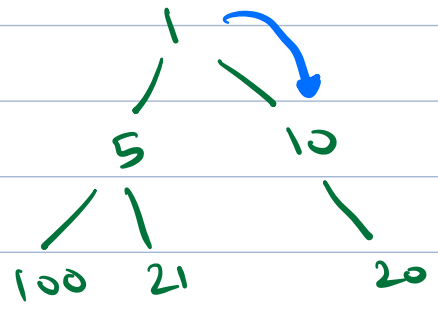


Considering → Price (Min)



→ Job is remote
→ Work Culture
→ Project
→ Timings

Greedy - It is an approach to solve optimisation problems by making locally optimal choices.



Max sum from root to leaf

1. Free Cars

There is a limited sale going on for toys.

A[i] → sale end time for ith toy

B[i] → happiness of ith toy

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

Buy toys such that sum of happiness is max.

sale end time →

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

↓ happiness

$t = 0 \times 2 \not< 3$

Toy → H

0 → 6

2 → 3

4 → 9

18

Idea : Pick toys in order of happiness

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

(1 and 3 marked X; 4 marked ↓)

$t = 0 \times 2 \not< 3$

t → H

4 → 9

0 → 6

2 → 3

18

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

T → H
1 → 5
4 → 9
0 → 6
20

t = 0 1 2 3

	0	1
A[] :	1	2
B[] :	3	1500

t = 0/1

t = 0

Greed

t = 0 1

T → H
1 → 1500
1500

t = 0 1 2

T → H
0 → 3
1 → 1500
1503

Idea: Pick toys in order of time

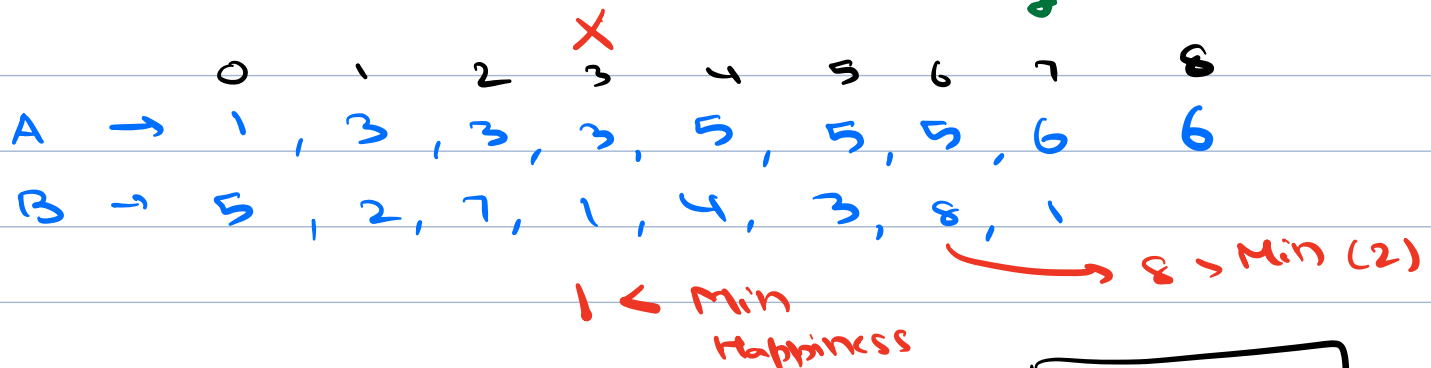
	0	1	2	3	4	5	6	7	8
A →	1	3	3	3	5	5	5	6	6
B →	5	2	7	1	4	3	8	1	x

t = 0 1 2 3 4 5 6

Unbuy 2 t = 4

Buy 8 t = 5

5	2	7
4	3	8
1		



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

5	~~2~~
7	4
3	8 1

Pseudocode

1. Sort toys in ascending order of time. ↓ $n \log n$

2. Minheap mh

```
t=0
for (i=0 ; i<n ; i++) <                    n log n
    if (t < A[i]) <
        mh.insert (B[i])
        t++
    >
    else <
        if (B[i] > mh.getMin()) <
            mh.extractMin()      // t--
            mh.insert (B[i])     // t++
        >
    >
>
```

3. Remove all elements from heap, add them and return sum. → N log N

TC : O (N log N) SC : O (N)

2. Candy Distribution

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

a) Every student should've atleast 1 candy

b) Students with more marks than any of his/her neighbours have more candies than them.

Find minimum candies to distribute.

	0	1	2	3
A :	1	5	2	1
	1	x_3	x_2	1

ans → 1 + 3 + 2 + 1 = 7

	0	1	2	3
A :	8	10	6	2
	1	x_3	x_2	1

ans → 7

A :	4	4	4	4	4
	1	1	1	1	1

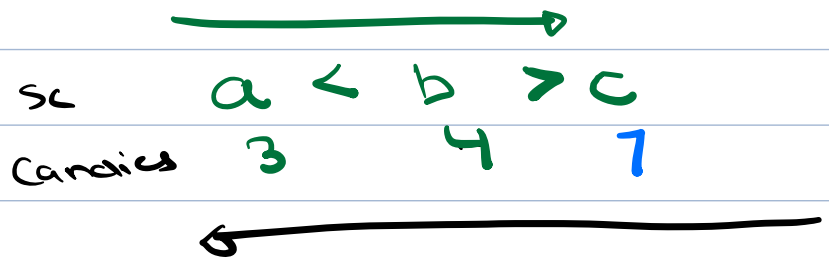
ans → 5

A :	1	6	3	1	10	12	20	5	2
	1	x_3	x_2	1	x_2	x_3	x_4	x_2	1

ans → 19

A : 1 6 3 1 10 12 20 5 2

1 x_3 x_2 1 x_2 x_3 x_4 x_2 1



① int c[n] = <1>

② for (i=1 ; i < n ; i++) <
    if (A[i] > A[i-1])
        c[i] = c[i-1] + 1
>

③ for (i=n-2 ; i ≥ 0 ; i--) <
    if (A[i] > A[i+1])
        c[i] = max(c[i], c[i+1]+1)
>

TC : O(N)

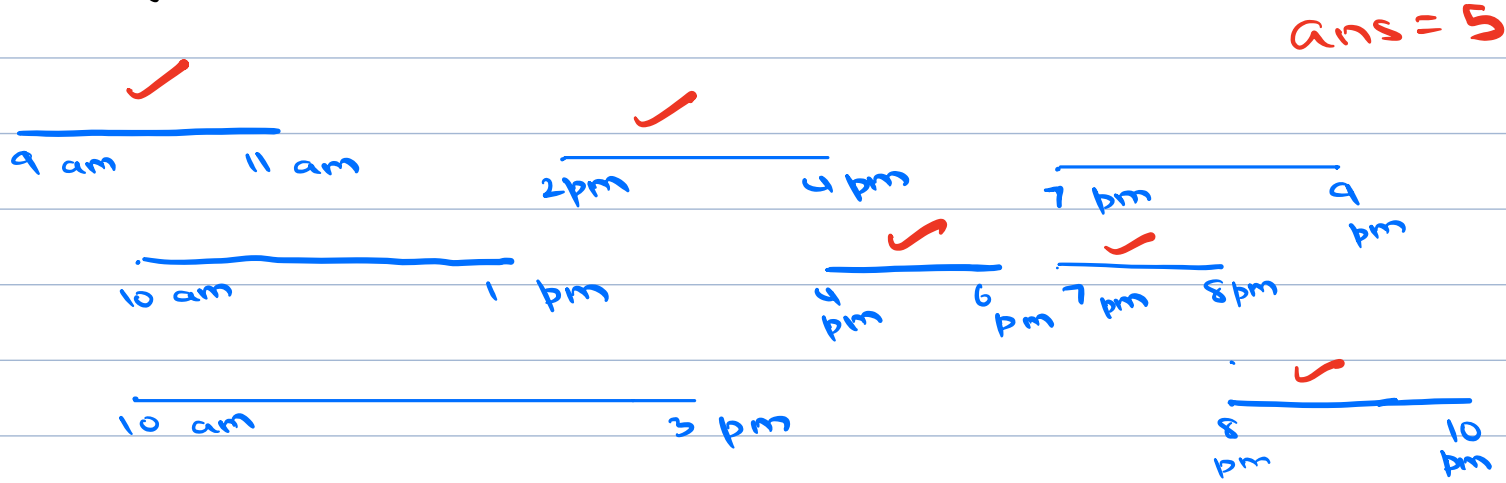
SC : O(N)

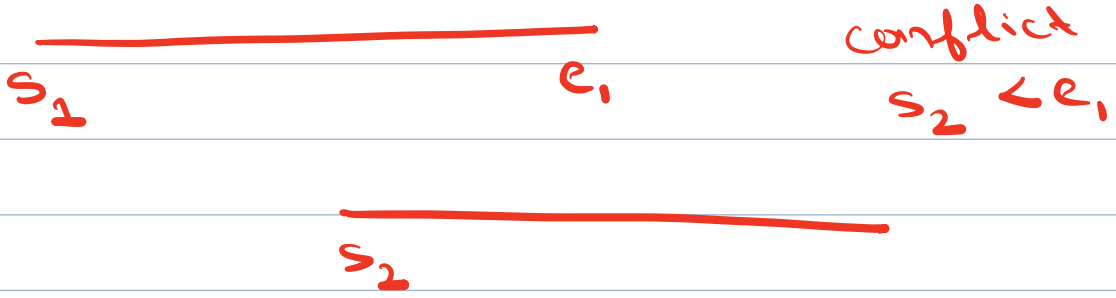
④ return sum(c[])

10:40

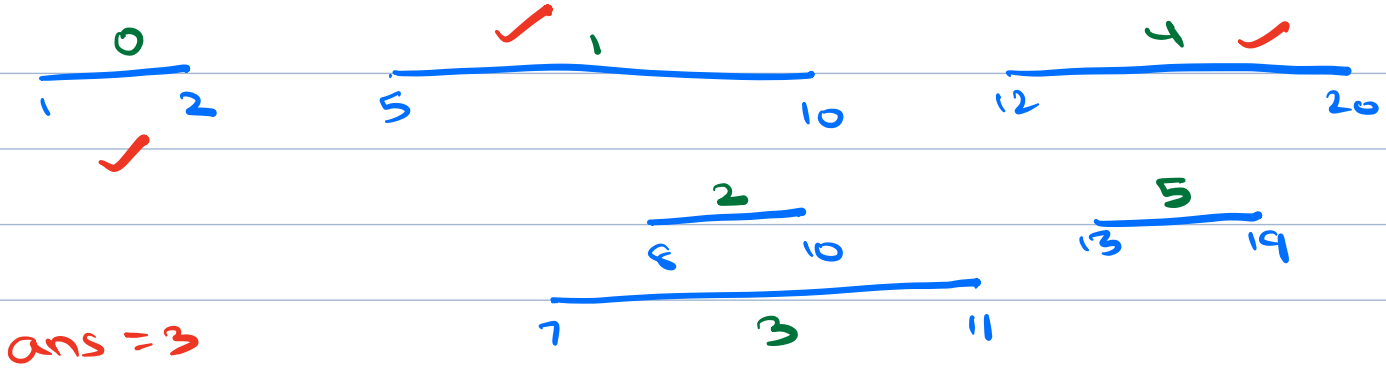
3. Maximum Jobs

Given N jobs with their start & end times. Find max no. of jobs that can be completed if only 1 job can be done at a time.

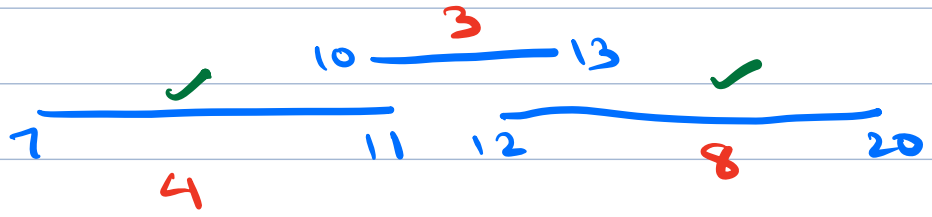




	0	1	2	3	4	5
S →	1	5	8	7	12	13
E →	2	10	10	11	20	19



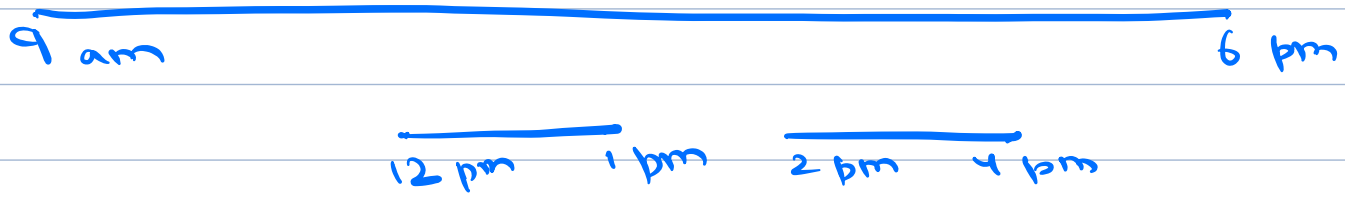
Idea 1 : Pick jobs based on duration



Greedy ans = 1 X

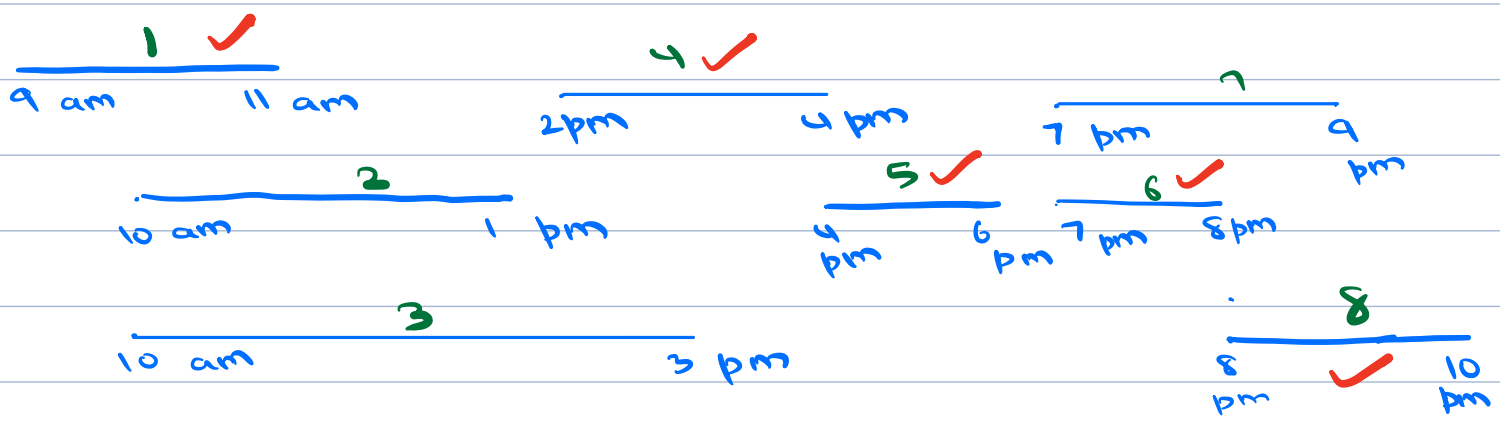
Optimal ans = 2

Idea 2 : Pick jobs based on start early

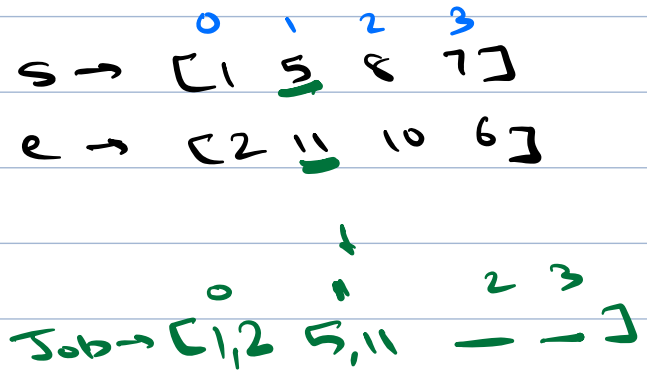


Idea 3 : Pick jobs which end early

start early
Our 2
Idea



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




```
int solve(int[] s, int[] e){
    Pair job[s.len]
    for (i=0; i<s.len; i++){ → n
        job[i] = new Pair(s[i], e[i])
    }
    Arrays.sort(jobs, compare) → NlogN

    ans = 1
    prevjobend = jobs[0].e
    for (i=1; i<jobs.len; i++){ → N
        if (jobs[i].s ≥ prevjobend){
            ans++
            prevjobend = jobs[i].e
        }
    }
    return ans
}
```

```
bool compare(pair u, pair v ) {
    if (u.e < v.e)
        return true      u comes 1st
    else
        return false     v comes 1st
}
```

TC: O(N log N) SC: O(N)

↓

sorting + jobs algo []

Merge N Sorted Arrays

0 - [2, 3, 11, 15, 20]
1 - [1, 5, 7, 9]
2 - [0, 2, 4]
3 - [-2, 5, 10, 20]

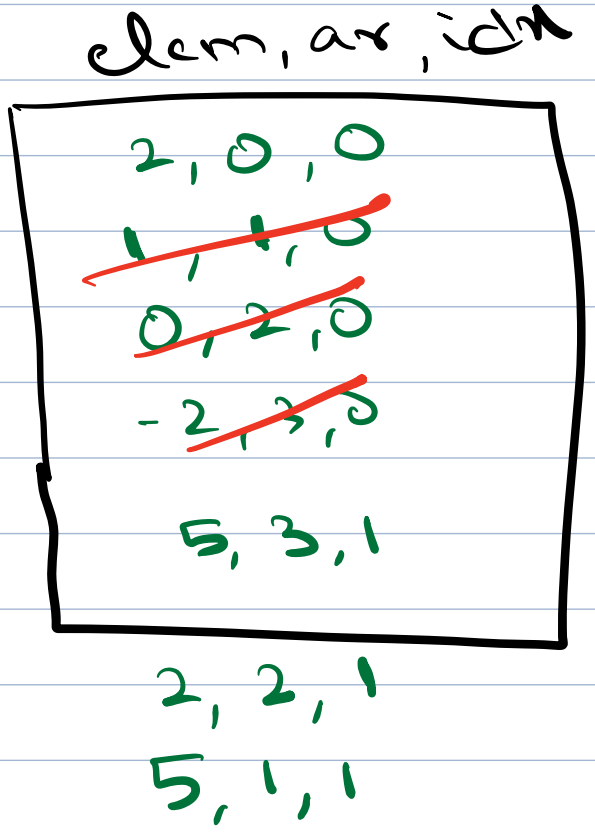
We've to merge these sorted arrays.

Idea:

- If we want to merge 2 sorted arrays, then we need 2 pointers.
- If we want to merge 3 sorted arrays, then we need 3 pointers.
- If we want to merge n sorted arrays, then we need n pointers. ⇒ complexity becomes very high we need to keep track of N pointers.

Optimized:

New: [-2, 0, 1 . . . .]



1. MinHeap of Point

```
class Point {
    int elem
    int arno
    int idx
}
```

2. Insert every ar's 0 idx element in min heap

```
list<int> l
```

3. while (mh.size()>0) {

```
    Point p= mh.extractmin()
    l.add(p.elem)
    if (p.idx+1 < p.arno.size()){
        mh.insert(<arno[elem], arno, idx+1>)
    }
}
```