Today's Agenda :-

1) Greedy (Properties of Greedy)

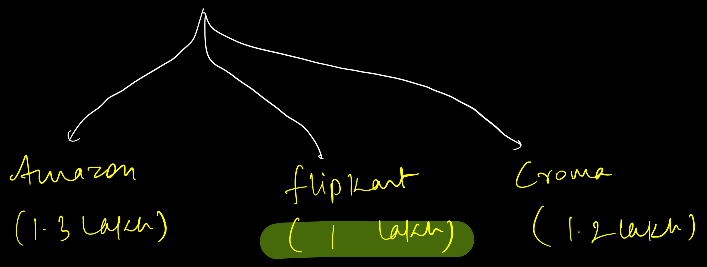
2) Fractional Knapsack

3) Activity Selection

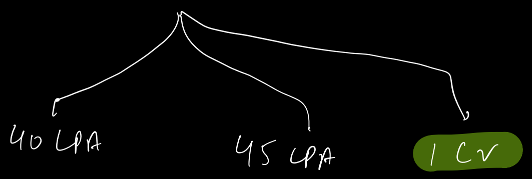
4) Job Scheduling

Greedy

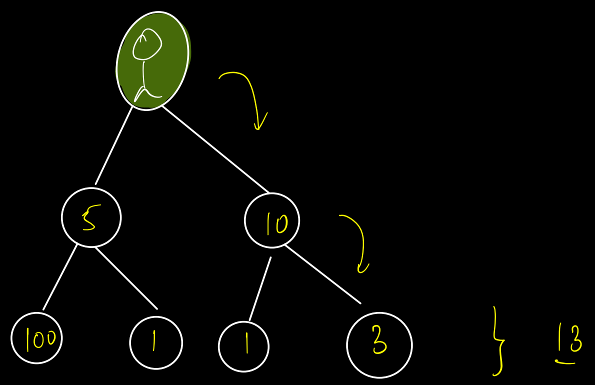
Buy an iphone (14 pro max)



Accept the offers



- Greedy algo is an approach to solve problems by making locally optimal choices at each step.



Q) Fractional Knapsack

You can consume K kg of vegetables, you can eat any integral amount of item. Max Protein you can get ?   K = 70 kg

Vegetables	Eating complete item gives you protein	P/kg	
Tomato 20kg	200	10	200
Apples 15kg	180	12	180
Onion 50kg	250	5	
Chicken 10kg	150	15	150
Potato 25kg	200	8	8 x 8 = 64
Mango 12kg	132	11	132
Seafood 5kg	100	20	100
			826 P

Eat on the basis of max Protein

Onion	250
Tomato	200
	450

wt:- [20, 15, 50, 10, 25, 12, 5]

P[]:- [200, 180, 250, 150, 200, 132, 100]

```
class Pair {
    int w, p;
    double ppkg;
    Pair ( w, p, ppkg) {
        ===
} }
```

```
int ar[];
ar[i] = x;
```

```
int fractionalKnapsack (int wt[N], int protein[N] , int K) {

    Pair items[ ] = new Pair [N];

    for(int i=0; i < N; i++) {
        Pair p = new Pair (w[i], p[i], p[i]/w[i]);
        items[i] =p
    }

    Arrays.sort(items); // based on PPkg

    double ans =0;

    for(int i = N-1; i >=0; i--) {
        if(K >= items[i].w) {
            ans += items[i].p;
            K= K- items[i].w
        }
        else {
            ans = ans + (K* items[i].ppkg)
            break;
        }
    }

    return ans;
}
```

(Custom Comparator)

TC: O(N logN)

SC: O(N)

wt :- [20, 15, 50, 10, 25, 12, 5]   K = 70

P[] :- [200, 180, 250, 150, 200, 132, 100]

items: [(20, 200, 10), (15, 180, 12), (50, 250, 5), (10, 150, 15), (25, 200, 8), (12, 132, 11), (5, 100, 20)]

After sorting

[(50, 250, 5) (25, 200, 8) (20, 200, 10) (12, 132, 11) (15, 180, 12) (10, 150, 15) (5, 100, 20)]

K = ~~70~~ − 5
~~65~~ − 10
55 − 15
40 − 12
28 − 20
8 − 8 = 0

ans = ~~0~~ ~~100~~ ~~250~~ ~~430~~ ~~562~~ ~~762~~ 826 Ans

Greedy Properties :-

Greedy
Binary search
DP

1) for min/max related problems.

2) based on what parameter, we want to apply greedy.

3) either prove it logically or discard it with counter example.

Break : 10 Min — 2 more problems

Q) Activity Selection

↳ Max count of task you can do ?

① Start a task, we need to complete

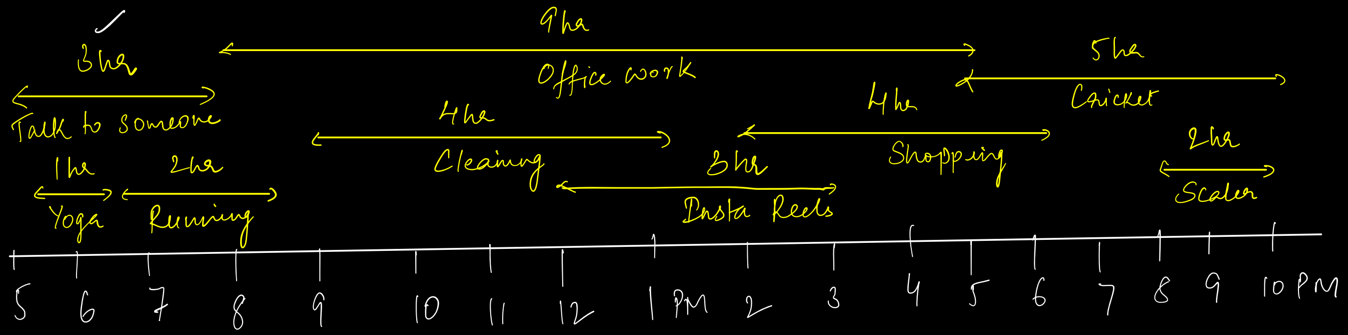
② At any point of time single task

(5, 8) (9, 13) (20, 22)

(5, 6) (12, 15)

(6, 8) (14, 18)

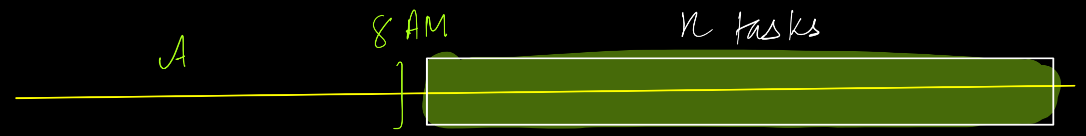
(8, 17) (17, 22)

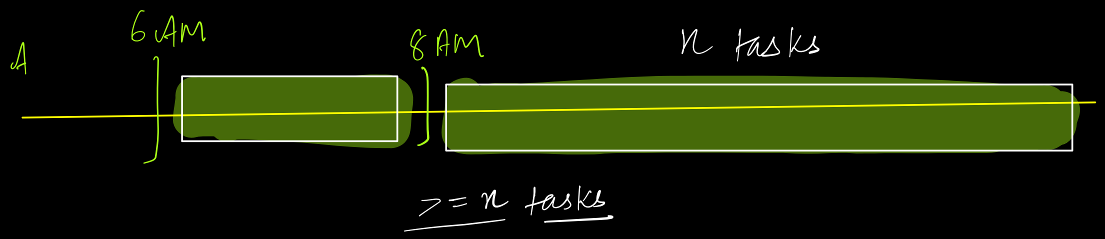


① Min duration task first Yoga, Running, Scaler, Insta Reels, } 4

② end early : Yoga, Running, Cleaning, Shopping, Scaler } 5

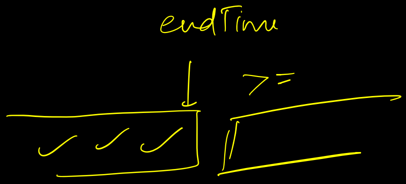
Correctness :





```
class Pair{ int s, int e }

int ActivitySelection (int start[N], int end[N]) {
    Pair [] arr = new Pair[N];
    for(int i=0; i< N; i++){
        Pair p = new Pair ( start[i], end[i])
        arr[i] = p
    }
                                          → Comparator
    Arrays.sort(arr) //based on end time
    int ans = 1;
    int endTime = arr[0].e;
    for(int i = 1; i < N; i++){
        if(ar[i].s >= endTime) {
            ans ++
            endTime = ar[i].e
        }
    }
    return ans
}
```



[

endTime = ~~13~~ ~~18~~ 22   ans = ~~3~~ ~~4~~ 5

[(5,6) (5, 8) (6,8) (9, 13) (12, 15) (8, 17) (14, 18) (17, 22) (20, 22)]

Job Scheduling :-

- Given N tasks to complete
- Deadline for each task, day on or before we can do this
- Payment assigned for every task
- Single task in single day.
- find max payment we can get

Job	deadline date	Payment
a	3	100
b	1	19
c	2	27
d	1	25
e	3	30

a	c	e
3	1	2
100	27	30

157

Exr

Job	Deadline	Payment
a	3	5
b	1	1
c	3	6
d	2	3
e	3	9

} Sort based on deadline.

Jobs	b	d	a	c	e
Deadline	1	2	3	3	3
Payment	1	3	5	6	9

Task at greater index can replace task at lower index.

	✓	✓	✓	✓	✓
Jobs	b	d	a	c	e
Deadline	1	2	3	3	3
Payment	9	3	5	6	9

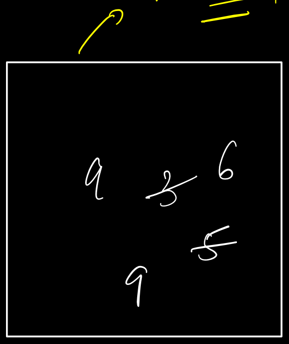
c e

b ~~d~~ ~~a~~

9 3 5

9+6+9 ✓

Min Heap 0



6+5+9

20

```
Pseudo Code :        class Pair { int d, p; }

int jobScheduling (int deadline[N], int payment[ ]) {
    Pair[] arr = new Pair[N];
    for(int i=0; i<N; i++) {
        Pair p = new Pair(deadline[i], payment[i]);
        arr[i] = p
    }
    Arrays.sort(arr) // based on deadline
    minheap<int> mh;
    for(int i=0; i<N; i++){
        if(arr[i].deadline > mh.size()){
            mh.insert(arr[i].payment)
        }
        else {
                                                    min value in
                                                         mh
            if (ar[i].payment > mh.peek()) {
                mh.remove()
                mh.insert(ar[i].paymnt)
            }
        }
    }
    //ans = sum of all elements in heap
}
```