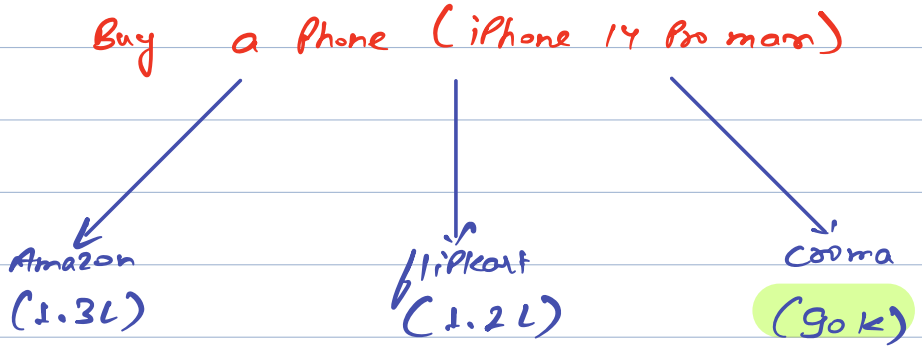
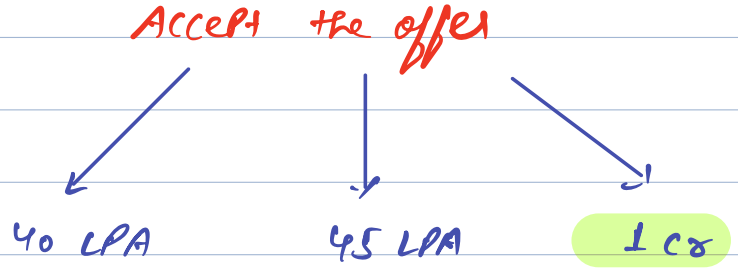
Today's Content

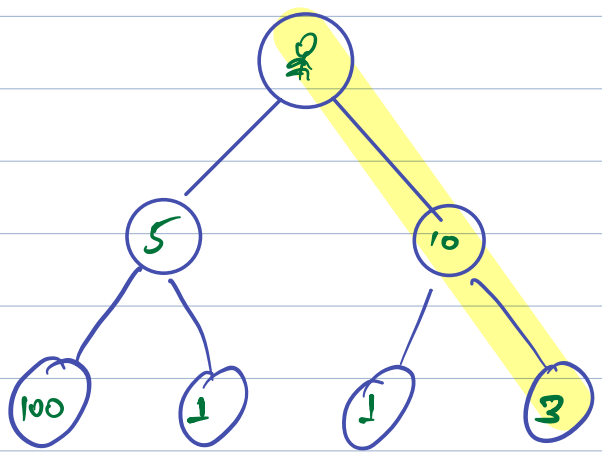
- Currency exchange
- fractional knapsack
- Greedy Properties
- Activity Selection
- job Scheduling

Greedy Algorithm





Is Greedy algo is an approach to solve Problems by making **locally optimal choices** at each step.



1 is always there.

curr[i] >= curr[i-1];

Indian currency: 1, 2, 5, 10, 20, 50, 100, 200, 500, 2000

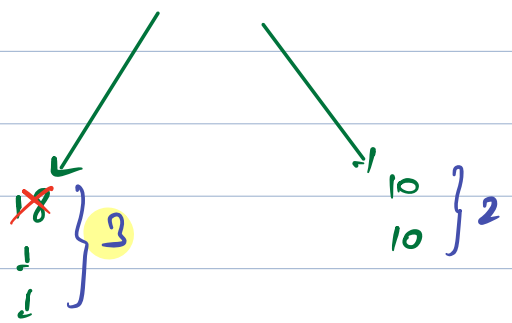
Cash: 5548 → min number of coins/notes to get required cash?

↳ multiples of same coins are allowed.

notes/Coins	Count	left amount.
2000	2	1548
500	3	48
20	2	8
5	1	3
2	1	1
1	1	1
	10	

idea always work?

Currency: 1 10 18 amount: 20



↳ Need to recheck this example.

1 10 21 amount: 51

↳ amount = 30

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 Protein gained	max Protein/kg
Tomato 0 → 20 kg	200 P → 10 P/kg * 20 = 200 P	
Apples 1 → 15 kg	180 P → 12 P/kg * 15 = 180 P	
Onion 2 → 50 kg	250 P → 5 P/kg	
Chicken 3 → 10 kg	150 P → 15 P/kg * 10 = 150 P	
Potato 4 → 25 kg	200 P → 8 P/kg * 8 = 64 P	
Mango 5 → 12 kg	132 P → 11 P/kg * 12 = 132 P	
Seafood 6 → 5 kg	100 P → 20 P/kg * 5 = 100 P	
	826 P	

eat on the basis of max Protein.

onion → 50 kg → 250 P

tomato → 20 kg → 200 P

450 P

zero K = 8

25 <= K

```
Class Pair {
    int w;
    int p;
    double ppu;
}
```

//Pseudo code

```
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(wt[i], Protein[i], Protein[i]/wt[i]*1.0);
        items[i] = P;
    }

    Arrays.sort(items);   // on the basis of PPU (Cimllement comparator)

    double ans=0

    for (int i=N-1; i>=0; i--){
        Pair xem = item[i];   // xem.w  xem.P  xem.PPU
        if (xem.w <= K){
            ans = ans + xem.P;
            K = K - xem.w;
        }
        else {
            ans = ans + (K * xem.PPU);
            break;
        }
    }

    return ans;
}
```

T.C: O(NlogN)

S.C: O(N)

Tracing

```
double ans:0
for (int i=N-1; i>=0; i--){
    Pair rem = item[i];
    if (rem.w <= k){
        ans = ans + rem.P;
        k = k - rem.w;
    }
    else {
        ans = ans + (k * rem.PPU);
        break;
    }
}
return ans;
```

rem.w
rem.P
rem.PPU

	0	1	2	3	4	5
W[]:	20	15	50	10	25	12	5
P[]:	200	180	250	150	200	132	100
PPU:	10	12	5	15	8	11	20

k = 70 - 5 = 65 - 10 = 55 - 15
= 40 - 12 = 28 - 20 = 8

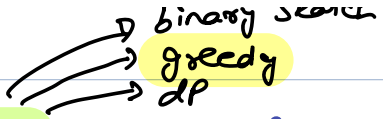
Pair rem: item[6]

ans = 0 + 100 + 150 + 180 + 132 + 200 + 64

i

	0	1	2	3	4	5	6
item[]:	{50, 250}	{25, 200}	{20, 200, 10}	{12, 132, 11}	{15, 180, 12}	{10, 150, 15}	{5, 100, 20}
	5	8					

* Greedy Properties:



(i) for min/max related problems.

(ii) based on what parameter we want to apply greedy.

(iii) either prove it logically or discard it with one counter example.

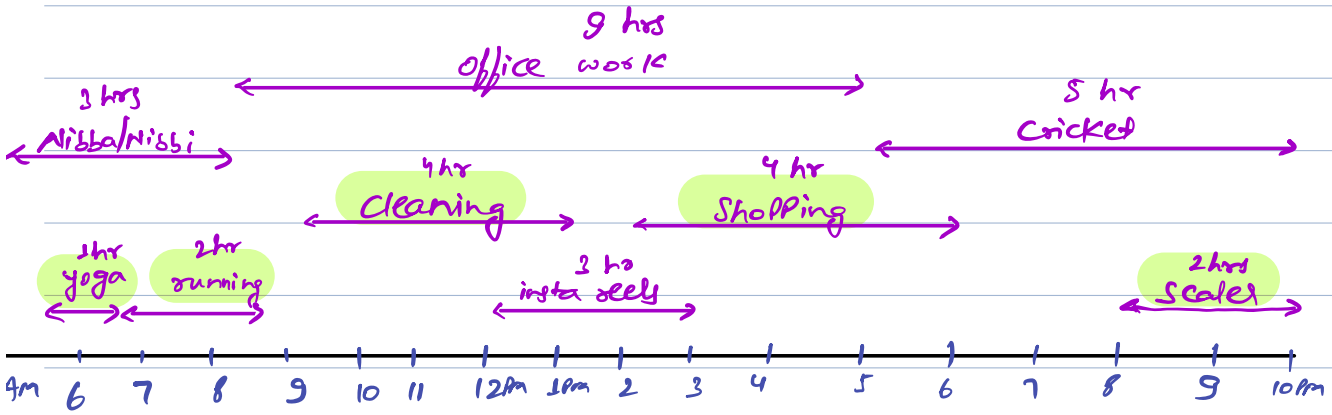
Break till 10:25 Pm

Q) Activity Selection

↳ Max Count of task you can do.

Note: (i) Start a task, we need to complete.

(ii) At any Point of time, single task.



(i) min duration task first ✗✗

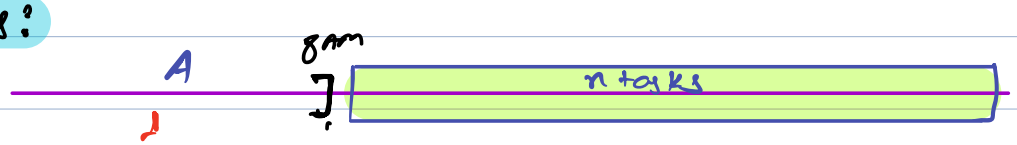
↳ yoga, running, Scaler, insta reels

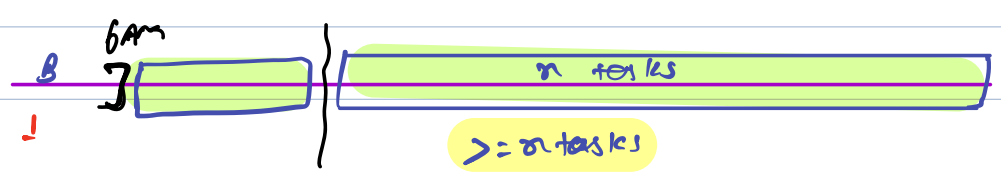
(ii) ending early

↳ yoga, running, Cleaning, Shopping, Scaler

↳ ans: 5

Correctness:





//Pseudo code

```
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
    }

    Arrays.sort(arr);   // sort on the basis of e. (end time)

    // Pick all the Non-overlapping tasks.
    int ans = 1;
    int endtime = arr[0].e;
    for(int i=1; i<N; i++){
        if(arr[i].s >= endtime){
            ans++;
            endtime = arr[i].e;
        }
    }

    return ans;
}
```

T.C: O(NlogN)

S.C: O(N)

ans = 1 2 3 4

yoga

(5,7) (5,8) (7,9) (9,13) (12,15) (8,17) (14,18)

nilo/nissi

ei

Job Scheduling

- Given N tasks to complete
  - → Deadline assigned for each task, day on or before we can do task.
  - → Payment assigned to each task.
  - → on any given day we can perform only 1 task and Each task take 1 day to finish.
  - → find max Payment we can get.

Ex1:

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

Parameter

(i) Payment

(ii) deadline

(iii) Payment + deadline

Ex2:

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

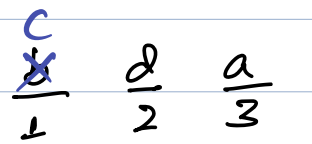
→ Sort on the basis of deadline

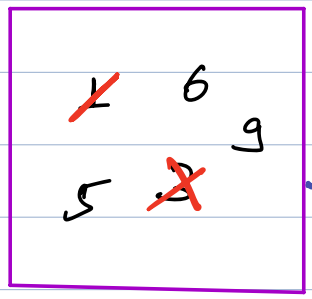
Jobs:	b	d	a	c	e
deadline:	1	2	3	3	3
Payment:	1	3	5	6	9

task at greater index can replace task at lower index.

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

Payment locked





Ans: 20

//Pseudo code

```
Class Pair {
    int d;
    int P;
}

int jobScheduling (int deadline[N], int Payment[N]) {

    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);   // on the basis of deadline.

    minheap<int> mh;
```

```
for (int i=0; i<N; i++){
    Pair sem = arr[i];
    if (sem.d > mh.size()){
        mh.insert(sem.P);
    }
    else{
        if (sem.P > mh.peek()){
            mh.remove();
            mh.insert(sem.P);
        }
    }
}
// Sum all the elements of the heap
}
```

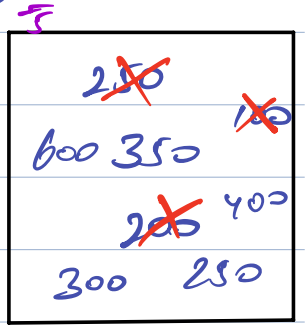
Tracing

i	0	1	2	3	4	5	6	7	8	9
Lead:	1	1	1	2	2	4	4	5	5	5
Payment:	250	200	350	200	150	300	250	100	600	400

```
for (int i=0; i<N; i++){
    Pair sem = arr[i];
    if (sem.d > mh.size()){
        mh.insert(sem.P);
    }
    else{
        if (sem.P > mh.peek()){
            mh.remove();
            mh.insert(sem.P);
        }
    }
}
// Sum all the elements of the heap
```

350 200 300 250 600

1 2 3 4 5



i	0	1	2	3	4	5
Wt(i):	20	15	50	10	25	12	5
Profit:	200	180	250	150	200	132	100

```
void bubbleSort (int arr[], int N){
    for(int i=0;  i<N-1; i++){ //no. of iter.
        for(int j=0;  j<=N-2-i;  j++){      // j < N-1-i  or
            if(arr[j] > arr[j+1] >      ){
                swap(arr[j], arr[j+1]);
            }
        }
    }
    Pair[] wt = new Pair[N];
    Arrays.sort(arr) Comparator{
        Quick Sort ← Arrays.sort(arr) Comparator{
    }
}
```

Pair { int val1; int val2; }

```
0  1  2  3  4  5  6
```

wt[]: (20,2) (15,7) (50,5) (10,0) (25,3) (12,1) (5,6)

Is sorting on basis of val1 or val2??

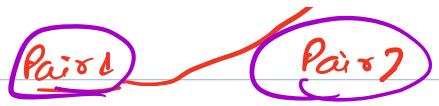
i:0→

arr[j] > arr[j+1]

arr[j] - arr[j+1] > 0

Comparator (arr[j], arr[j+1]) > 0

1 → swap; 0 → ; -1 →



Pair1 Pair2

this.val1 − o.val1 >=0 → 1

→ 0

→ -1

Pair1.val1 - Pair2.val1

↳ → 1 ⇒ increasing order

log1 + log2 + log3 . . . — — log N

= N log N

1 + 2 + 3 . — — —— + N ≃ N*(N+1)/2 ≃ O(N²)

return (P1.val1 + P1.val2) − (P2.val1 + P2.val2)

log(a) + log(b) = log(a*b)

log(1) + log(2) — — log(N)

log(1*2* . . . N)

N log N