* Currency Exchange

Indian Currency.

1 2 5 10 20 50 100 200 500 2000

Total: 5548

Min. no. of coins/notes to get the required cash.

	Notes/Coins.	Amount left
2000	2	1548
500	3	48
20	2	8
5	1	3
2	1	1
1	1	0

$\boxed{10}$

↑

Min # of notes/coins required to make a total 5548 rs.

* Why greedy works in Indian Currency ?

⇒ Every denomination is atleast greater than or equal to at least twice of its previous currency.

x Rs ⇒ $x > 500$

i) $x - 500 = y$ ⇒ 1 notes.

ii) $x - 2*200$ ⇒ 2 notes of 200 Rs.

⇒ Greedy Works in Indian Currency

* Currency : 1 10 18

Target : 20 Rs

Greedy

$$\underbrace{1 \times 18 + 2 \times 1}_{}$$

3 coins.

Correct

2×10 ⇒ 2 coins

⇒ 20

Q:

food items	Proteint content	Protein/kg
Tomato : 20 kg	200	10
Apple : 15 kg	180	12
Onion : 50 kg	250	5
Chicken : 10 kg	150	15
Potato : 25 kg	200	8
Mango : 12 kg	132	11
Seafood : 5 kg	100	20

→ We can pick max of 70 kg

→ We can pick [redacted] 1 kg from each item.

→ Pick the items s.t we get max protein.

⇒ Knapsack Problem

(Fractional Knapsack)

Idea 1:

Take the item based on Max Protein.

$$\underset{(250)}{\overset{50\text{ kg}}{\text{Onion}}} + \underset{(200)}{\overset{20\text{ kg}}{\text{Tomato}}} \Rightarrow 450 \text{ units Protein} \quad \times$$

⇒ Greedy based on Max protein content.

Idea 2 : Take items based on protein | kg.

	Seafood	Chicken	Apple	Mango	Tomato	Potato
item wt	5	10	15	12	20	8
Protein	100	150	180	132	200	64

826

⇒ Greed based on Max protein/kg.

⇒ Can we get more than 826 units of protein?

NO, because we are picking Max protein possible for each kg.

Properties of Greedy :

1. For optimization related problems.

⇓

Max profit | Min cost | Coins | .....

2. Based on what parameter we want to apply greedy.

3. Check for any counter examples where Greedy won't work.

* Ex. of Greedy Algorithms.

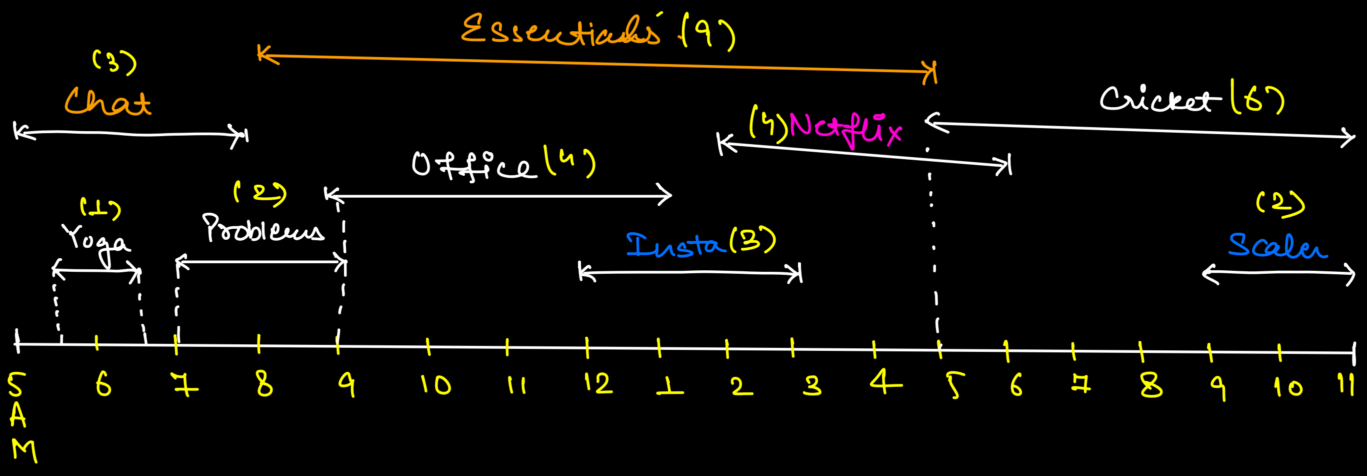
i) Prims & Kruskals Algo ⇒ MST

ii) Dijkstra's Algo.

iii) Huffman Coding

*

Activity Selection.



1. Once you start a task, we need to complete it.
2. At any given time, we can only perform one task.
3. Max. no. of tasks we can do.

ans Yoga
Problem Solving
Office
Netflix
Scaler.

* **Greedy**

1. Pick the tasks with MIN **duration**.

→ Yoga
→ Problem
→ Scaler
→ Insta

} **4**

2. Pick the tasks with MIN Start **time**.

↓ Starts **earlier**.

Chat
Essentials
Cricket

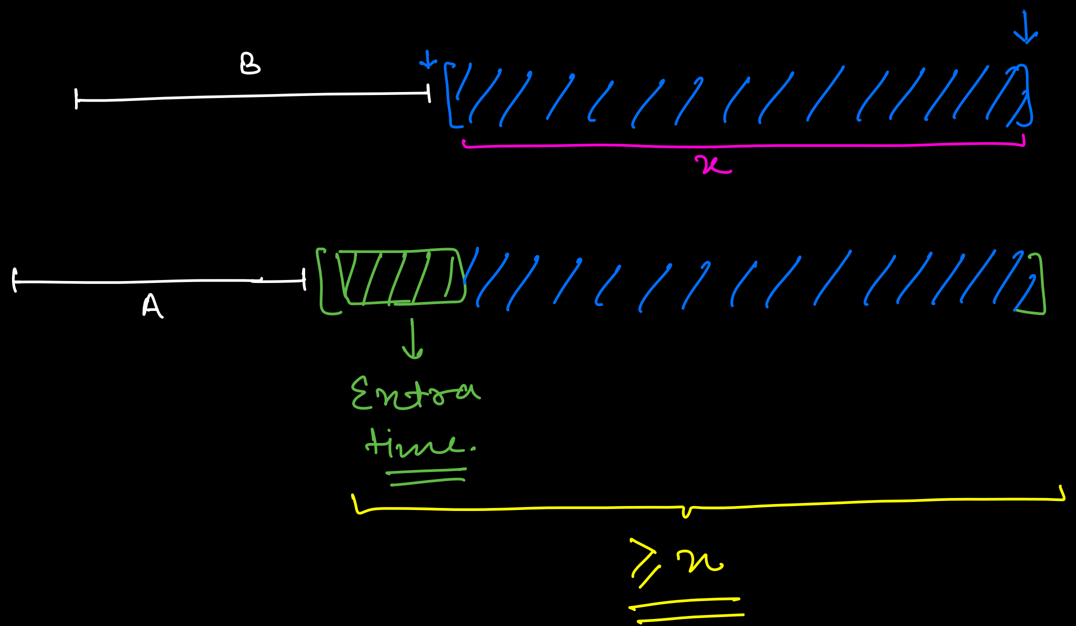
} X

3. Pick the task that ends **early**.

Yoga
PS
Office
Netflix
Scaler

} ✓ **5**

Correctness of logic



⇒ By picking the tasks that ends first we are leaving more slots | time to do more tasks.

Q

Job Scheduling

1) Given N tasks to complete.
2) Deadline assigned for each task, day on or before we can do the task.
3) Payment is assigned to each task.
4) On any given day we can perform only ① task & each task takes 1 day.
5) Find max payment we can get.

Ex

Task	Deadline (day)	Payment
a	3	100
b	1	19
c	2	27
d	1	25
e	3	30

d	e	a	⇒ 155
25	30	100	

a	c	e	⇒ 157
100	27	30	

a	e	c	]
✓	✓	✗	

↓

b	d	a
✓	✗	

d	c	a	⇒ 152
✓	✓	✓	

c e a ⇒ ✓

✓ ✓ ✓

e c a ⇒ ✓

* deadline = x^{th} day

⇒ We need to complete the task on <= x^{th} day

i.e on day 1 | 2 | 3 | - - - - | x

⇒ Greedy based on Deadline (Sort based on deadline)

Task :	b	d	c	a	e
deadline :	1	1	2	3	3
Payment :	19	25	27	100	30

Diary

~~19~~ ~~25~~ 27

100 30

→ 157

↳ insert

getMin()

deleteMin()

} Min Heap

Ex

Task :	1	2	3	4	5	6	7	8	9	10
Deadline :	2	1	1	1	4	5	4	5	5	2
Payment :	200	250	200	350	300	100	250	600	400	150

⇒ Sort based on deadline

⇓

Task :	2	3	4	1	10	5	7	6	8	9
Deadline :	1	1	1	2	2	4	4	5	5	5
Payment :	250	200	350	200	150	300	250	100	600	400

~~250~~ 350 ~~200~~
300 250
~~100~~ 600
400

⇒ 1900

						↓
Task :	2	3	4	1	10	6
Deadline :	1	1	1	2	2	5
Payment :	250	200	350	200	150	300

~~250~~ 350 200
300

⇒ ~~750~~ 850

```
int   maxPayment ( list< pair< int, int > > data){
                                   deadline  pay
        sort ( list ) // Based on the deadline
        MinHeap< int > minH;
        for( i= 0; i< N; i++ ) {
            pair<int, int >  x = data[i];
             dead = x.first;
             pay  = x.second;
              if(deadline > minH.size()){
                    // Empty slot is there
                    minH.insert(pay);
              }
              else if (pay > minH.getMin()){
                    minH.deleteMin();
                    minH.insert(pay);
              }
         }
         ans = 0
         while ( minH.size() > 0) {
                  ans += minH.getMin()
                  minH.deleteMin();
         }
         return ans;
}

         TC: O(NlogN)  SC: O(N)
```