Today's Content:

- → Currency exchange ✓
- → Fractional knapsack ✓
- → Greedy Properties ✓
- → Activity Selection
- → Job Scheduling
- → Min Choclates
- → Rats { If time permits }

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

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

	Notes/Coins	leftout
2000 →	2	1548
500 →	3	48
20 →	2	8
5 →	1	3
3 →	1	1
1 →	1	0
	10	

Why Indian Greedy ? Works

[Any denomination atleast >=2 more previous denomination]

Currency : 1 10 18 : Greedy is not working

↳ Money : 20 → Min coins required to get target money

As per greedy :

18
1
1 } 3

Min Coins

10
10 } 2 coins

Super Market : If needed we can eat a single kg from each item

max protien we can get

We can eat 70kg

Vegetables :		Eating Complete item protien gained	Eat based max total Protien	Eat based on protien/kg
Tomato	20 kg	200p	200p → 20kg	10p/kg - 20kg - 200
Apples	15 kg	180p		12p/kg - 15kg - 180
onion	50 kg	250p	250p → 50kg	5p/kg -
chicken	10 kg	150p	450p → 70kg	15p/kg - 10kg - 150
potato	25 kg	200p		8p/kg - 8kg - 64
Mango	12 kg	132p		11p/kg - 12kg - 132
Seafood	5 kg	100p		20p/kg - 5kg - 100
				Max protein : 826

For every kg we are taking max protien, we can get

Greedy Properties :

Min/Max

✓ a) For optimization related problems

✓ b) Based on what parameter we want to apply greedy

✓ c) By coming up with Counter examples

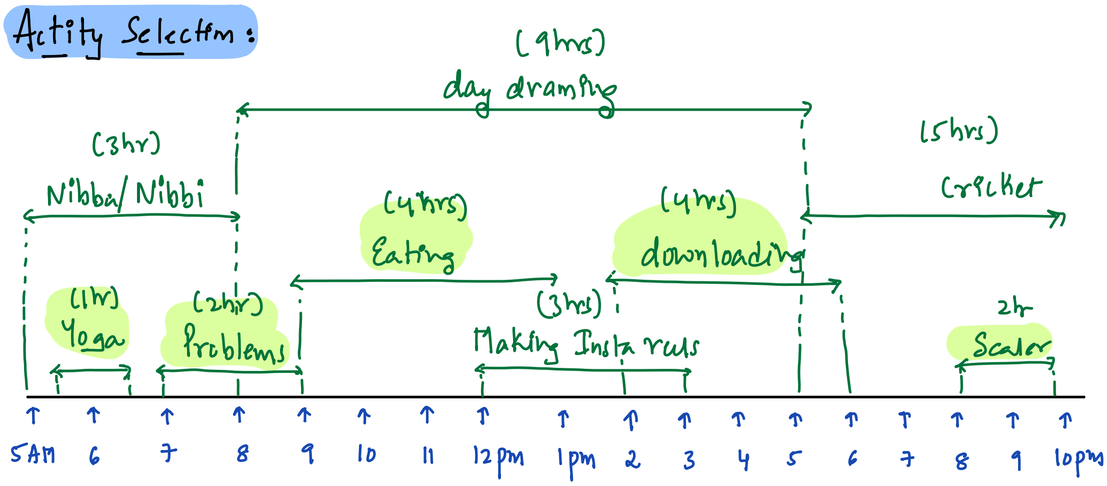
Real time algorithms :

a) Prims / kruskals algorithms →

b) Dijkstra's →

c) Haffman's Coding

Actity Selection:



→ Start a task we need to complete

→ At any given point single task

→ Max tasks which we can do

Tasks:
Yoga
Problems
Eating
downlading
Scaler

greedy:

1) Min duration *

Yoga
Problems
Scaler
Making Insta Reels

2) Pick with min Start time

Nibba/Nibbi
day dreaming
cricket

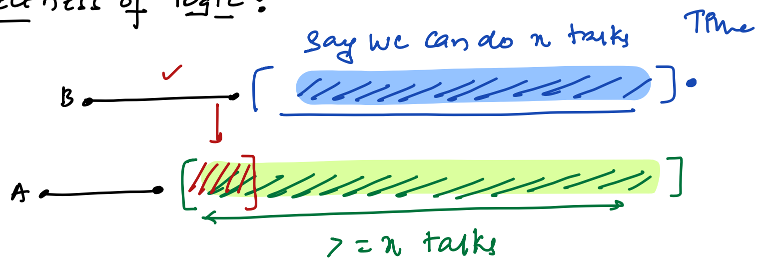
3) Pick task which ends earlier: ✓

Yoga
Problems
Eating
downlading
Scaler

→ Sort all tasks based on end times & iterate & get non-overlapping tasks

TC: N log N + N   SC: N

Correctness of logic:



→ By simply picking tasks which ends first, we are leaving more slots/time to do more tasks

Job Scheduling :

Given N Tasks to complete,

→ Deadline assigned for each task, day on or before we can do task

→ payement assigned to Each task

→ On any given day we can perform only 1 task & Each task take 1 day finish

→ Find max payement we can get

Ex1: $\boxed{\text{deadline } x \to \text{finish task on or before day } x}$

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

ans:

c e a → 157 : ans = 157

d c a → → 152

day1 day2 day3

a e d →

→ Greedy Deadline

→ 1 → 1 → 2 → 3 → 3

19 25 27 100 30

$\cancel{19}$ $\cancel{25}$, 27, 100

30

= 100 + 27 + 30

= 157

Ex2:

Task	Dead line	Reward
a	3	5
b	1	1
c	3	6
d	2	3
e	3	9

Sort based on deadline :

1 2 3 3 3

1 3 6 5 9

~~1~~ ~~3~~, 6

5 9

→ ans : 20

last example:

Tasks:	1	2	3	4	5	6	7	8	9	10
deadl:	2	1	1	1	4	5	4	5	5	2
Money:	200	250	200	350	300	100	250	600	400	150

Sort based on deadlines:

1	1	1	2	2	4	4	5	5	5
250	200	350	150	200	250	300	600	100	400
x	x	✓	x	✓	✓	✓	✓		

~~250~~, 350 250
~~150~~ ~~200~~, 400
300 600

ans = 1900

100 < min value in dabba
we won't insert

400 > min value in dabba

operations

→ size()
→ insert()
→ getMin()
→ deleteMin()

} min Heap

1	1	1	2	2	4
250	200	350	150	200	250
✓	*	✓	✓	✓	

~~250~~
350 ~~150~~
200
250

```
                                   deadline       payment
//  int maxCost( list< pair< int, int >> data) {
      int n = data.size()
      // data.sort( based on deadline)            TC: (NlogN + NlogN)
      Minheap<int> mh;                            SC: O(N)
      for(int i=0; i<n; i++){                     11:06 → 11:15pm
          pair<int, int> x = data[i]
          int day  = x.first
          int pay  = x.second

          if( day > mh.size()){
              // Empty slot is there
              mh.insert(pay)
          }
          else if( pay > mh.getMin() )// No Empty Slots
              mh.deleteMin()
              mh.insert(pay)
          }
      }
      int ans = 0
      while( mh.size()>0){
          ans = ans + mh.getMin()
          mh.deleteMin()
      }
      return ans;
}
```

Choclate distribution : → { HW }

Given N Student marks, assign choclate to all N Students in such a way that

→ Each Student should atleat get 1 choclate

→ If ar[i] > ar[i-1],

choclates assigned to i^{th} Student should be more $i-1^{th}$ Students Choclate

If ar[i] > ar[i+1]

choclates assigned to i^{th} Student should be more $i+1^{th}$ Students Choclate