Today's Content:

- → Fractional knapsack
- → Greedy Properties
- → Activity Selection
- → Job Scheduling
- → Min Choclates

Indian curreny :

1	2	5	10	20	50	100	200	500	2000
✓	✓	✓	✗	✓	✓	✗	✗	✓	✓

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

	Amount / Count	Rem
5548 →	2000 / 2	1548
1548 →	500 / 3	48
48 →	20 / 2	8
8 →	5 / 1	3
3 →	2 / 1	1
1 →	1 / 1	0

: ans = 10, min coins/notes required = 10

Obs: For Indian curreny greedy always works?

? Every denomination atleast >= 2 previous deminotion

$x > 500$ & $x < 1000$:

$x - 500$: 1 coins :

$x - 2*200$: 2 coins

Curreny: 1 10 18

amount: 20

	Amount / Count	Rem
20 →	18 / 1 →	2
2 →	1 / 2 →	0

as per greedy = 3 coins ✗

Expected coins = 2

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

We can eat 70kg, What's the max protein we can get

Vegetables :		P/W	Eating Complete item protien gained
Tomato	20 kg	10	200p .
Apples	15 kg	12	180p
onion	50 kg	5	250p
chicken	10 kg	15	150p
potato	25 kg	8	200p .
Mango	12 kg	11	132p
Seafood	5 kg	20	100p

Greedy ?

max protien

50kg onion - 250

20kg Tomato - 200

Total - 450

protien/weight ratio

Seafood	5kg	- 100
chicken	10kg	- 150
apples	15kg	- 180
mango	12kg	- 132
Tomato	20kg	- 200
Potato	8kg	- 64
Total		→ 826

Greedy Properties :

a) For optimization min/max related problems → (1)

b) Based on what parameter we want to apply gredy

c) By coming up with Counter examples

Real time algorithms :

a) Prims / kruskals algorithms } graphs

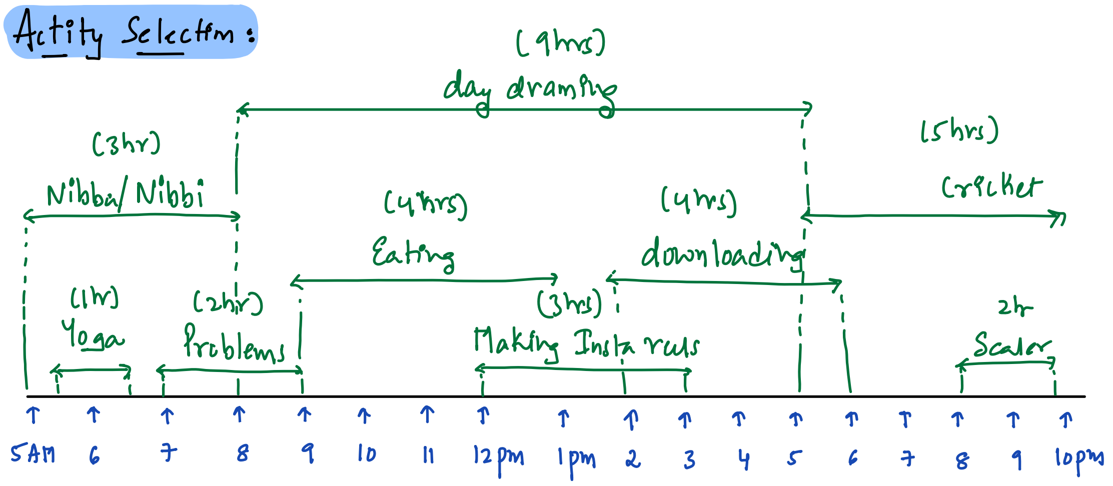
b) Dijkstra's → google maps

c) Haffman's Coding

Greedy : Technique used to solve optimization problems by making "locally optimal choices" at each step so that we can get final optimization

greedy choice

Actity Selection :



→ Start a task we need to complete

→ At any given point single task

→ Max tasks which we can do

c) Tasks with early end time

- → yoga
- → Problems
- → Eating
- → downloading
- → Scaler

Tasks:

Yoga

Problems

Eating

downloa

Scaler

greedy: parameter:

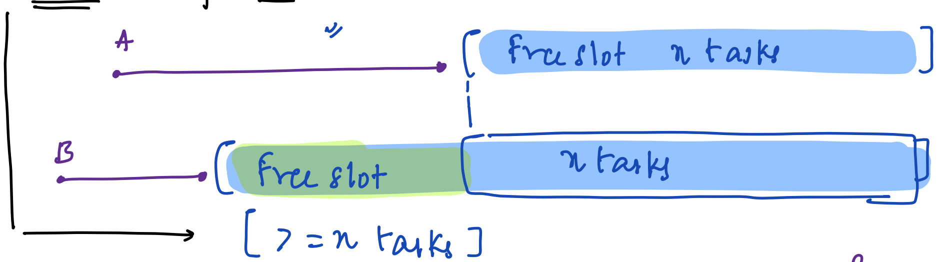
a) Tasks with less duration *

- → yoga
- → problems
- → Scaler
- → Making Insta

b) Tasks with early start time *

- → nibba/nibbi

Correctness of logic :



Choose task, ends early, it will give us more free slot hence we can do more tasks.

Idea: Sort tasks based on endtime, while selecting tasks make sure, no overlap. TC : $n\log n + n$

8:42 → 8:52 am

Job Scheduling :

Given N Tasks to complete, payement assigned to Each task

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

On any given day we can perform only {1 task} & Each tasks take {1 day finish}

Find max payement we can get

Ex1: deadline x → 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

day1	day2	day3	
d	b	a	*
d	c	a	: 152
c	e	a	: 157
a	e	c	: * payement

Parameter → payement / deadline / deadline + payement /

↳ // Sort based on inc of deadline

↳ If 2 items have same deadline, any item come first

→ Sort above example Based on deadline

	b	d	c	a	e
Dealine	1	1	2	3	3
amount	19	25	27	100	30

inc order deadline ✓

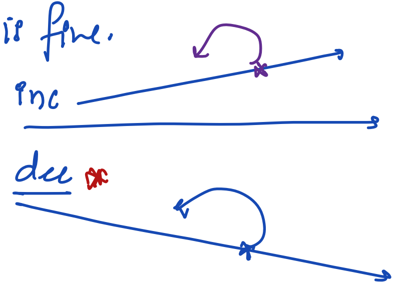
~~19~~ ~~25~~ 30
27 100

Doubts: 3 → 3
4 → 4
↕ → 25
→ 35

Ex2:

Task	Deadline	Reward
a	3	5
b	1	1
c	3	6
d	2	4
e	3	9

Obs: Replace lower deadline with higher deadline task is fine.



Sort based on deadline:

	b	d	a	c	e
deadline	1	2	3	3	3
amount	1	4	5	6	9

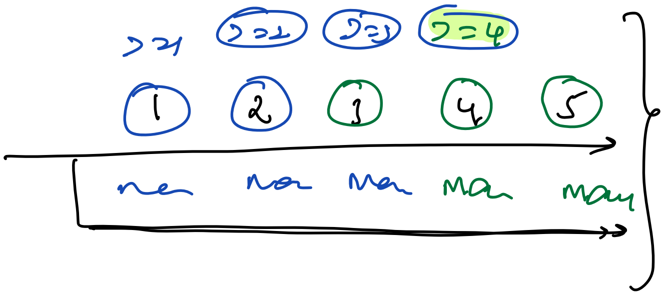
□ with a deadline

$\cancel{1}$ $\cancel{4}$ 5

6 9

ans = 20

3



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:

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

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

ans = 1900

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

} minheap

```
Idea: 1) Sort all Tasks based on dealine in inc        { inserts/deletemin() }
      minheap<int> mh;        TC: nlogn + n{ logn }        SC: O(n)
      Iterate on all tasks:
      | : Say we have task A: with deadline d & amount A
      |   if( mh.size() < d) { mh.insert(A)}
      |   else // when can we replace
      |   |   if( A > mh.getmin()){
      |   |   |   mh.deleteMin()
      |   |   }   mh.insert(A)
      }   }
      2) Calculate sum of all elements of minheap.
```

Choclate distributin :

Given N Student marks, assign choclates to all N Students in such a way that. Calulate min no: of choclate, to assign all N Students

→ Each Student should atleat get 1 choclate

If ar[i] > ar[i-1], only neighbours

choclates assigned to i^{th} Studen should be more $i\text{-}1^{th}$ Students Cholate

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

choclates assigned to i^{th} Studen should be more $i{+}1^{th}$ Students Cholate

Ex1:

	0	1	2	3	4	
ar[] :	1	5	2	1	6	
choclate :	1	1	1	1	1	
	1	5	2	1	6	→ Choclates = 15
	1	3	2	1	2	→ Choclates = 9

Ex2:

ar[] :	2	6	3	1	10	1	2	20	5	2	
	1	2	1	1	2	1	2	3	1	>=1	: leftside
	1	3	2	1	2	1	2	3	2	1	: ans
	1	3	2	1	2	1	1	3	2	>=1	: rightside

Only 1 cond

If ar[i] > ar[i-1]
Eq

Only 1 cond

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