GREEDY ALGOS

→

Example 1:

Trust

iPhone
- Apple store [1,00,000] → Free Sticker (worth 1 Rs)
- Amazon [90,000] → 10% off HDFC credit ← Amazon
  → 15% using Amazon
- Flipkart [95K] → 20% off using Citi Bank
- Paytm [93K] → 2K cashback

→ Option with min price

→ Greedy about the price

Example 2:

Company 1 — 23 L

Company 2 — 25 L

Company 3 — 30 L

→ CTC breakup
- → Base
- → shares / ESOP
- → PB / JP

→ Work-Life Balance

→ Role [Dev / Testing]

→ Location

→ Your Technical growth

→ Brand [Google India]

→ CTC is not the best

→ Being greedy on

optim-

Example 3:

→ 20 books in this year

→ I'll select around 100 books from my wishlist.

→ Thinner Books [less pages]

→ Books with less

We are greedy decision at every point

→

Example 4:

Master LLD

there are 100 resources

→ [Java / C++ / Language Agnostic]

→ [Beginner / Intermediate / Advanced]

→ Rating

→ If locally optimal solution leads to global optimal solution, then greedy can be used.

→ Greedy is a paradigm to optimally find solution to the whole problem by taking decisions that are optimal w.r.t the smaller problems, always being greedy repeating these optimal reach overall

and if we [illegible] decisions, we'll be able to [illegible] optimal solutions.

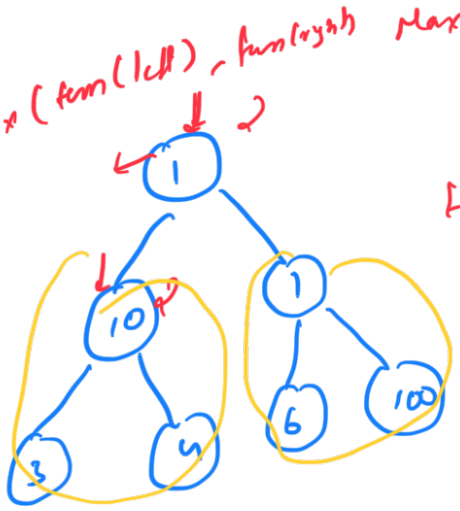
Ex: Joining ropes using minimum cost

A = 3 1 7 9 6

Strategy: At every point, join the 2 smallest ropes

Greedy Solution.

Ex 6:



1 + 10 + 4 = 15

Locally optimal solutions is not leading to global optimum.

→ Any problem which cannot be solved using greedy, we have explore all the possibilities and take the best option.

→ It is tough to prove that greedy solution will work

to prove greedy will not work, we [illegible] failing example.

can just give a

Question: Fractional knapsack Problem

→ Given N items each with a value & a weight

N = 3

	I0	I1	I2	I3
val	60	120	100	1500
weight	10	30	20	50

W = 200 (I3)

I0, I1, I2 = 10 + 30 + 20 = 60

Case 1: I0, I1 ⇒ (180, 40) + (100/20) × 10 = 230

Case 2: I0, I2 ⇒ (160, 30) 160 + 120/30 × 20 = 240

Case 3: I1, I2 ⇒ (220, 50)

W-max = 50

I0 = {60, 10}
I1 = {120, 30}
I2 = {100, 20}

→ Thief can choose a fraction of an item.

	I0	I1	I2
Val =	100	200	500
Weight =	50	60	80
Value Per Unit	2		

W = 30

→ Per kg

	I0	I1	I2
Val =	60	120	100
weight =	10	(30)	20

W_max = 50

Price Per unit

6 4 5

Val = 60 + 100 = 160 + 80 = 240

Cap = 10 + 20 = 30 + 20 = 50

(I2, 1020)

(I0, 6010)

W_max = 50

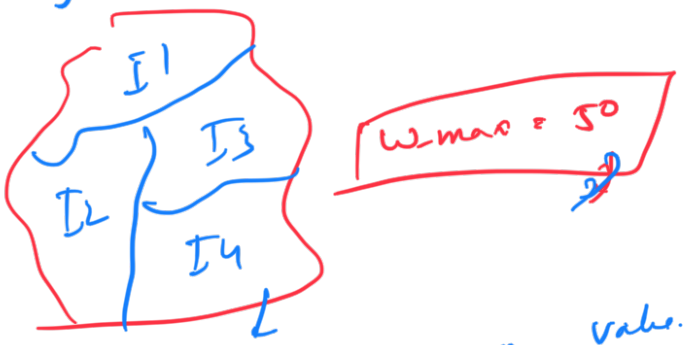
$\left(\frac{120}{30}\right) \times 20 = 80$

Strategy

1) Sort the items by (Val/wt) & (sort)

2) choose items in decreasing order of ratio while we have more capacity

Why is greedy working Here.



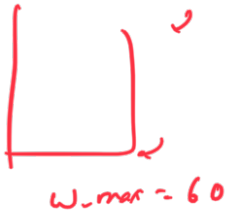
Give the max value.

T.C: O(n log n)

S.C: O(n)

Question: 0-1 knapsack

	I0	I1	I2	I3
val	100	280	120	10
wt	10	40	20	5
		4	2	1

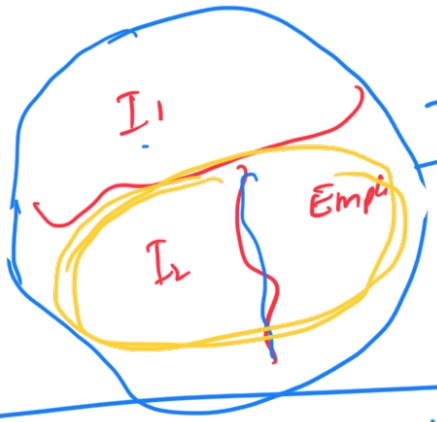


val/wt : 10 7

$(I_1, I_2) \Rightarrow (400, 60)$

val = 100 + 280 of 10 = 390

wt = 10 + 40 + 5 = 55



Greedy

I_3

D.P

I_2 +

Greedy will not work.

Greedy + Seq

Question: Activity Selection

N activities each with a start & end time.

N = 6

	A0	A1	A2	A3	A4	A5
start =	1	3	0	8	5	5
end =	2	4	6	9	7	9

→ We can perform atmost activity at any time instant.

Ans = [A0 A1 A4 A3]

Ans = 4

Brute force: all Possibility

Consider

→ Consider all subsequences) Backtracking

Wrong Approach 1:

Choose the jobs with minimum duration

A0 A1 A2

→ (0, 4) (3, 6) (5, 10)

Duration 4, 3, 5

end = start

[A0, A2]

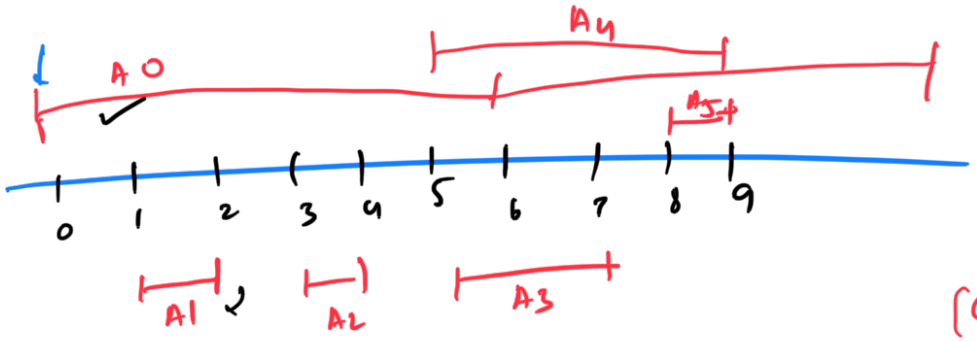
[A0, A2]

Ans = 1

A1: (3, 6)

Wrong Approach 2: Sort by start time.

	A0	A1	A2	A3	A4	A5
start =	0	1	3	5	5	8
end =	6	2	4	7	9	9



Approach 2: sorted based on End time (Comparator)

(sort) start, end

	A0	A1	A2	A3	A4	A5
	1	3	0	5	5	8
	2	4	6	7	9	9
	↑	✓	X	↑	↑	↑

Ans = A0, A1, A3, A5..

= Ans 4

end(A0) ≤ start(A1)

Merge Sort → n log n

T.C: O(n log n)

S.C: O(n) → [Array of pairs]

Approach:

Sort by start time

	A0	A1	A2	A3	A4	A5
start :	0	1	3	5	5	8
end :	6	2	4	7	6	9

Activity : A1, A2, A4, A5.

Question : Free Cars

Given 2 Arrays

t=1 | t=1,2 | t=1,2 | t=1,2,3 | t=1,2,3

	C0	C1	C2	C3	C4
deadline :	1	3	2	3	3
profit :	5	6	1	5	9

t = 2

deadline[i] = the time instant before which we can buy the car for free

profit[i] = the profit we get by buying a car

→ Buy the cars to get maximum profit.

→ You can buy only car at a single [redacted] time instant.

t=1

deadline

{ deadline
  profit

[c2 c1 c4]

t=1

Strategy1: t=1 ~~2~~ ~~3~~ 4

c0, c1, c3 = 5 + 6 + 3 = 14

[strategy2:

t=1, c0
t=2, c1
t=3, c4

=> 5+6+9 = 20

Strategy3: Always buy car with the highest profit

t=1 => c4 => 9
t=2 => c1 => 6
t=3 => c3 => 3

9+6+3 = 18

-> We have to use the deadline information

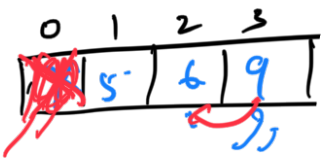
Approach1:

t=3

deadline: 3 3 1 3 2

profit: 9 6 5 3 1

Array=1



=> 5+6+9 = 20

| 5 | 9 | 6 |

T.C:

1) Sorting: O(nlogn)

2) O(t × n) = O(n log n + t n)
↓ (Max value of deadline array)

S.C: O(t)

Approache:

	c0	c1	c2	c3	c4
deadline =	1	2	3	3	3
profit =	5	1	3	6	9

t = ~~3~~ 4

[c0, 1, 5]
~~[c1, 2, 1]~~ [c3, 3, 6]
~~[c2, 3, 3]~~
[c4, 3, 9]

profit = 5 ~~+ 1~~ ~~+ 3~~ ~~+ 6~~

5 + 6 + 9 = 20

→ if profit[i] > min(DS)
→ Remove min(DS)
→ insert

DS?
Min Heap

t = Max time

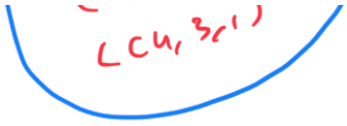
	c0	c1	c2	c3	c4
deadline	1	1	2	2	3
profit :	5	4	3	9	1

t = 1 ~~2~~ ~~3~~ 3

[c0, 1, 5]
~~[c2, 2, 3]~~
[c3, 2, 9]

min(Heap) > curr
] Cr Noa

profit = 5 + 9 + 1 = 15



Min Heap

T.C:

1) Sort by deadline : o(nlogn)

2) Min Heap & Iterate. o(nlogn)

T.C: o(nlogn)
S.C: o(n)

	C0	C1	C2	1000 space
deadline:	1000	1001	1003	
profit:	10	20	30	

t=3

(maxtime - mintime)

(1003 - 1000)

t=1, C0

t=2, C1

t=3 C2

Question: Distributing N candies

1) Given ratings of N students

2) Every student should get atleast one candy

3) Students with higher rating than neighbour should get more candies

Task: What is min no. of candies needed to distribute to the students

Ex1

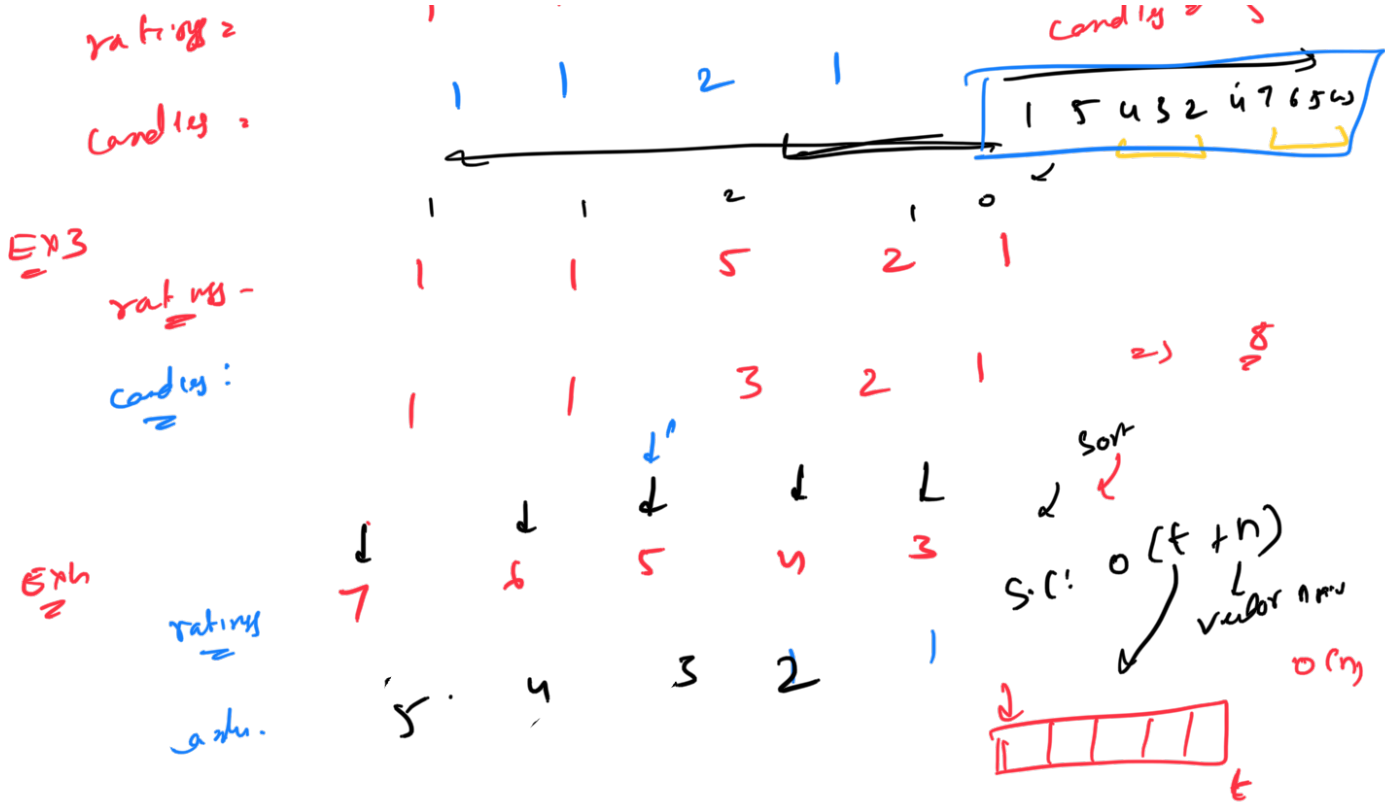
	s0	s1	s2	s3
ratings =	1	5	2	1
candies =	1	3	2	1

N=4

candies = 7

Ex2

	1	1	5	2



Brute Force:

T.C: $O(n^2)$ S.C:

Observation

1) All students will get only 1 candy with minimum mock rating

2) Students with max rating will get max candies?

ratings: 1 10 1 2 3 4

Candies: 1 2 1 2 3 4

Approach 2:

-) Can we simply sort the ratings array? We'll lose the info our neighbour

<rating, index>

→ have a pair

	s0	s1	s2	s3	s4	s5	s6	s7
rat-	1	2	5	6	3	1	9	2
ans	1	2	3	4	2	1	2	1

sort: [<1,0>, <1,5>, <2,1>, <2,7>, <3,4>, <5,2>, <6,3> <9,6>]

max(left, right)+1

Equal

Equal

	s0	s1	s2	s3
rating	1	3	3	2
candies	1	2	2	1

<1,0> <2,3> <3,1> <3,2>

rating

T.C: O(nlogn + n)
~ O(nlogn)

S.C: O(n)

Approach:

rating

left:

right:

	s0	s1	s2	s3	s4	s5	s6	s7
	1	2	5	6	3	1	9	2

→ candies : 1 2 3 4 2 1 2 1

candies[i] = max(left[i], right[i])

T.C : O(n)

S.C : O(n)

→ Candy of i^{th} student will depend on left & Right Neighbour

→ Find min no. of candies by only considering left

→ Find min no. of candies by only considering right

map [left[i], right[i]]

if neighbour has h.

→ Proofs will involve paths

→

```
int candy(vector<int> & ratings) {
            int candiesCnt = 0, curCandy, pos;

            vector<pair<int, int> > valueWithPos;

            for (int i = 0; i < ratings.size(); i++) {
                valueWithPos.push_back(make_pair(ratings[i], i));
            }

            sort(valueWithPos.begin(), valueWithPos.end());

            vector<int> candies (ratings.size(), 0);

            for (int i = 0; i < valueWithPos.size(); i++) {
                pos = valueWithPos[i].second;
                curCandy = 0;

                if (pos > 0 && ratings[pos - 1] < ratings[pos]) {
                    curCandy = candies[pos - 1];
                }

                if (pos < ratings.size() - 1 && ratings[pos + 1] <
ratings[pos]) {
                    curCandy = max(curCandy, candies[pos + 1]);
                }
                candies[pos] = curCandy + 1;
                candiesCnt += candies[pos];
            }
            return candiesCnt;
        }
};
```

```
int maxProfit(vector<int> A, vector<int> B){
      for(int i = 0; i < A.size(); i++)
            v.push_back(<A[i], B[i]>);
      sort(v);
      minHeap pq;
      timer = 0;
      for(int i = 0; i < A.size(); i++){
            if(v[i].time > timer){
                  pq.push(v[i].profit);
                  timer++;
            }
            else{
                  if(pq.top().profit < v[i].profit){
                        pq.pop();
                        pq.push(v[i].profit);
                  }
            }
      }
      return (sum of heap)
}
```