Ajanta
Page No.
Date 29-06-25

GREEDY

Concepts via Questions

- Simple and easy with good time complexity but alot of times will not get global optimum.

* **Concept-1 : Activity Selection** - (selecting disjoints) or (max length chain of pairs)

→ * There are n activities with start & end times.
* Select max. no. of activities performed by one person.
* assume work on single activity at a time.

↳ ∴ disjoint

→ e.g. start = [10, 12, 20] } already sorted (must) ans. $A_0, A_2 = 2$
end = [20, 25, 30]

end times for respective start times

↳ unhi activities ko select kr skte jo end time se disjoint ho + maximize one

→ Approach : * Always select first activity (A_0) → Just simple obs. not global optimum like DP
* for next activity : start >= last chosen end time
* count ++

→ Easy code :

```
psum {
① int[] start = {...};
   "  "  end  = " ;
② ArrayList<> list = new ArrayList<>();
③ list.add(0); → pehla to lena hi hai
   int maxAct = 1;
④ int lastEnd = end[0];
   for (int i = 1 to end.length){
    if (start[i] >= lastEnd){
     maxAct ++;
     list.add(i);
     lastEnd = end[i];
    }
   }
```

print (maxAct); ✓

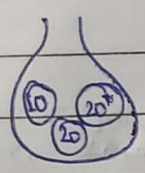
better → print ("A" + ~~ans~~ list.get(i) + " ");
inside for (int i=0 to ans.size())

* Concept-2 : Fractional knapsack

→ * There are weights and values of 'n' items
* Put these values in sack of weight 'W'
* In such a way that we get the max^m. total value.
* Here we can take fractional values also ∴ 0-1 knapsack & unbounded knapsack are subsets of fractional knapsack.

→ e.g. val = [60, 100, 120]
wt = [10, 20, 30]
W = 50

→ * "obviously" koi ek item ek baar hi jaa skta hai



60+100+80 = 240 ∵ 1. Pehla poora daal diye (10kg) → 60
2. Dusra " " " + (20 kg) → 100
30 kg
∴ 20 kg left
3. Teesra wale # se 20 kg jitna daal diye (50 full) exact → 120/30 x 20 = 80

→ Approach: * weight ↓ (to put more items), value ↑ (more profit) } favourable situation = val/wt ↑↑

* ∴ Select that one jiska (val/wt) ratio max. ho (∵ wt kam chahiye tha aur val jada)

* If space of last > rem. wt. ∴ val/wt x space left

→ * from above eg. v/w = [6, 5, 4] 1. ∴ if we choose 0th index first capacity left = 40
1st 2nd 3rd
2. ∴ then " " 1st " ∴ capacity = 20 left

* 3. ∴ finally if last space > rem. wt.
∴ Fraction used → (left highest ratio) x (space left)
= 4 x 20
= 80 ✓
∴ capacity full

Page No.
Date 30-06-25

import java.util.*;

→ Code :- psvm {

int[] val = {---};

int[] wt = {----};

int W = 50;

int n = val.length;

double[] ratio = new double[n];

for (i=0 to n){

ratio[i] = (double) (val[i]/wt[i]);

}

data type used to store fractional values

general

** Descending sort → Arrays.sort(items, (a,b) → Double.compare(b.ratio, a.ratio));

* ascending sort → Arrays.sort(items, (a,b) → Double.compare(a.ratio, b.ratio));

↳ this compares a.ratio

why this?
→ If you sort ratio directly you'll lose track of which val & wt it belongs to

→ Integer not int because comparator lambda req. object array like Integer[]

Integer[] index = new Integer[n];

for (i=0 to n){

index[i] = i;

}

Arrays.sort(index, (a,b) → Double.compare(ratio[b], ratio[a]));

eg. val = {60, 20, 100}
wt = {10, 20, 30}
ratio = {6, 1, 3.33}

Direct sort → ratio = {6, 3.33, 1}

Now you don't know 6 is from val 60 & wt 10

∴ we created index array = {0,1,2} which keeps index of val, wt, ratio in order

Then we sorted index array based on ratio ∴ index = {0,2,1} ∴ first pick index 0 then 2 then 1.

double totalval = 0;

int remCapacity = W;

for (i=0 to n){

int idx = index[i];

if (remCapacity >= wt[idx]){

totalval += val[idx];

remCapacity -= wt[idx];

}

else{

totalval += ratio[idx] * remCapacity;

break; → sack gets full here

}

}

print(totalval); ✓

Descending sort 1 : Arrays.sort(index,(a,b)→ Double.compare(ratio[b],ratio[a]));
(sort after comparing with other)

Descending sort 2 : Arrays.sort(coins, Comparator.reverseOrder());
(direct)

Ajanta
Page No.
Date 30-06-25

Concept-3 : Min. Abs. Diff. Pairs

→ * Find the pairs from two arrays such that their absolute difference is minimum.

→ e.g. * A=[1, 2, 3] B=[2, 1, 3] → |1-2| + |2-1| + |3-3| = 2 ✗
|1-1| + |2-2| + |3-3| = 0 ✓

→ Approach : * nos. that are in pair if they're close to each other then uska abs. diff. utna hi kam hoga.

* Sort the arrays & get diff. at indices 0 to n.

→ Easy code :
```
psvm {
[] A = ✓
[] B = ✓
Arrays.sort(A);
   ,,     ,,  (B);
int minDiff = 0;
for(i=0 to A.length) {
  minDiff += Math.abs(A[i] - B[i]);
}
print(minDiff);
```

Concept-4 : Indian Coins Change (Khulla)

→ * If you have coins of [1,2,5,10,20,50,100,500,2000].
* Find min. no. of coins used to make change of value V.

→ e.g. V=551 → ans. 3 (∵ 500,50,1)
V=590 → ans. 4 (∵ 500,50,20,20)

→ easy code :
```
import java.util.*;
psvm {
Integer[] coins = {1,2,5,.....,2000};
Arrays.sort(coins, reverseOrder());
}
```

(understand approach from code)

Integer[] → object array

reverseOrder() → for a comparator

Arrays → Collections.

→ Descending sorted ∵ pehla bada coin lo for min. change

```
ArrayList<Integer> ans = new ArrayList<>();
int count = 0;
int amt = 590;
```

Concept → (Debugging way to sochlo if not clicked)

```
for (i=0 to coins.length){
    if (coins[i] <= amt){
        while (coins[i] <= amt){
            count++;
            ans.add(coins[i]);
            amt -= coins[i];
        }
    }
print(count);
```

to print coins used →

```
for (i=0 to ans.size){
    print(ans.get(i) + " ");
}
```

Read!

Concept-5 : Job Sequencing

* → Given Jobs with deadline and profit.
→ Every Job can be done one at a time.
→ (min. possible deadline for any Job is 1)
→ maximize total profit.

Deadline ↓ Profit ↓

* eg. Job A = {4, 20} → matlb Job A krna main 4 hrs lagega with profit 20
Job B = {1, 10} → " " B " " 1 hr " " " 10
Job C = {1, 40}
Job D = {1, 30}

Choices we can select → 1. If we select A (first), then it will take 4 hrs so we can't do any other job (∵ baaki ki 1 hr main khatam)
∴ Only A → ₹20
2. BA → can take both ∵ A ki deadline 4 hai aur B 1 hr me done ho jayega → BA → ₹30
3. CA → ₹60 → Max. ans.
4. DA → ₹50
5. BC → Not possible → ∵ Koi ek karne jaye to dusra ki deadline over.

Ajanta
Page No.
Date

* Approach → • sort jobs based on profit (descending)
∴ Job C, D, A, B.

• time = 0
for (i=0 to jobs) {
if (job(deadline) > time) {
add job in ans
} time ++

* code → import java.util.*;

public class JobSeq. {

static class Job {
∵ no need for declaring objects wala jhanjhat
int id;
int profit;
int deadline;

Constr of all (id, deadline, profit)
}

psvm {

int[][] jobInfo = {{4,20},{1,10},{1,40},{1,30}};
ArrayList<Job> jobs = new ArrayList<>();
for (i=0 to jobInfo.length) {
jobs.add(new Job(i, jobInfo[i][0], jobInfo[0][i])
}
deadline profit

Descending order sort for profit → Collections.sort(jobs, (obj1, obj2) → obj2.profit - obj1.profit);

Socho (easy hai bahut)
Jiska profit high hai wo select ho gya + kisi aur ka deadline bacha hai to add it based on profit sorting

ArrayList<Integer> seq = new ArrayList<>();
int time = 0;
for (i=0 to jobs.size()) {
Job current = jobs.get(i);
if (current.deadline > time) {
seq.add(current.id);
time++;
}}

print(seq.size() + " = max Jobs");
for (i=0 to seq.size()) {
print(seq.get(i));
}