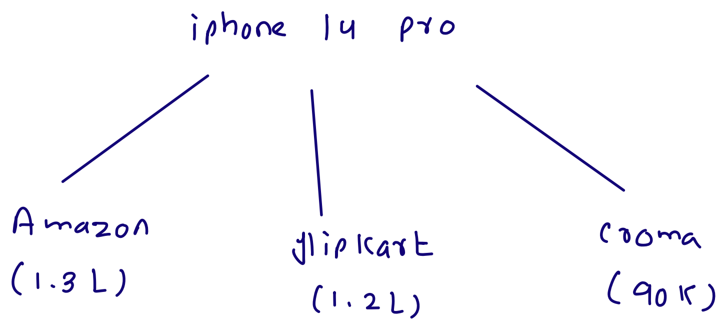
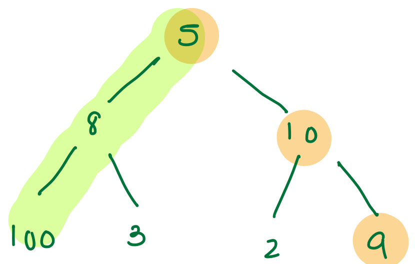
Agenda

1) Introduction
2) Fractional knapsack
3) Activity Selection
4) Job scheduling

Introduction

Greedy algo : Choosing the local best every time.





Q.1 Fractional Knapsack

we can consume K kg of food item. Find max protein we can get.

Note: Eating any integral amount of an item is allowed.

K = 70 kg

Food item	Eating complete item protein gained	(PPK) → protein per kg
Tomato → 20 kg	200	→ 10 u
Apples → 15 kg	180	→ 12 u
Onion → 50 kg	250	→ 5 u
chicken → 10 kg	150	→ 15 u
Potato → 25 kg	200	→ 8 u
Mango → 12 kg	132	→ 11 u
seafood → 5 kg	100	→ 20 u

(50kg + 20kg)

✗ max protein = 250 + 200

= 450 u

K = 70 kg

	Food item	Eating complete item protein gained	
0	~~Tomato → 20kg~~	~~200~~	→ ~~10 u~~
1	~~Apples → 15kg~~	~~180~~	→ ~~12 u~~
2	Onion → 50kg	250	→ 5 u
3	~~chicken → 10kg~~	~~150~~	→ ~~15u~~
4	~~Potato → 25kg~~ (8kg)	~~200~~	→ ~~8u~~
5	~~Mango → 12 kg~~	~~132~~	→ ~~11 u~~
6	~~seafood → 5kg~~	~~100~~	→ ~~20 u~~

ans = 100 + 150 + 180 + 132 +
+ 200 + 64

K = ~~70~~ (seafood)
~~65~~ (chicken)
~~55~~ (apples)
~~40~~ (mango)
~~28~~ (tomato)
~~8~~ (potato : 8kg)
0

```
class Pair {
    int wt;
    int protein;
    double ppK;
}
```

```
int solve(int[] wt, int[] protein, int k){
    int n= wt.length;
    Pair[] arr = new Pair[n];

    for(int i=0; i<n; i++) {
        // Create pair with the help of ith item
        Pair p= new Pair(wt[i], Protein[i], protein[i]*1.0/wt[i]);
        arr[i] = p;
    }
                                  → sort arr on the basis of
    Arrays.sort(arr, —);            ppk (protein per kg)
                                        {ascending order}

    double ans = 0.0;

    for (int i= n-1; i>=0; i--) {
        Pair ji = arr[i];
        if (ji.wt <= k) {
            // take ji completely
            ans += ji.protein;
            k = k - ji.wt;
        }
        else {
            // take k kg's of ji
                ans += k * ji.ppk;
                break;
        }
    }

    return ans;
}
```

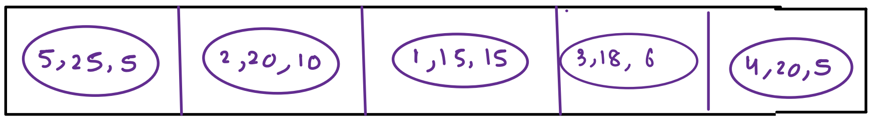
TC: O(nlogn)

SC: O(n)

```
int solve(int[] wt, int[] protein, int k){
    int  = wt.length;
    Pair[] arr = new Pair[n];
    for(int i=0; i<n; i++) {
        // create pair with the help of ith item
        Pair p= new Pair(wt[i], Protein[i], protein[i]*1.0/wt[i]);
        arr[i] = p;
    }
    Arrays.sort(arr, — );          → sort arr on the basis of
                                     ppk (protein per kg)
    double ans = 0.0;
    for (int i= n-1; i>=0; i--) {
        Pair fi = arr[i];
        if ( fi.wt <= k) {
            // take fi completely
            ans+= fi.protein;
            k = k- fi.wt;
        }
        else {
            // take k kg's of fi
            ans+= k* fi.ppk;
            break;
        }
    }
    return ans;
}
```

$K = 7$

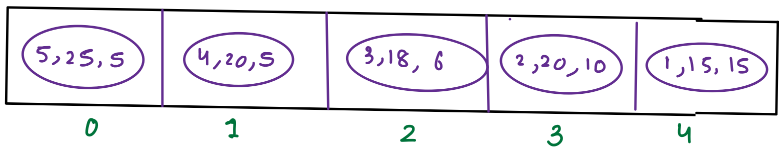
	0	1	2	3	4
wt:	5	2	1	3	4
protein:	25	20	15	18	20



dry run

```
for (int i= n-1; i>=0; i--) {
    Pair fi = arr[i];
    if ( fi.wt <= k) {
        // take fi completely
        ans+= fi.protein;
        k = k- fi.wt;
    }
    else {
        // take k kg's of fi
        ans+= k* fi.ppk;
        break;
    }
}
```

after sorting



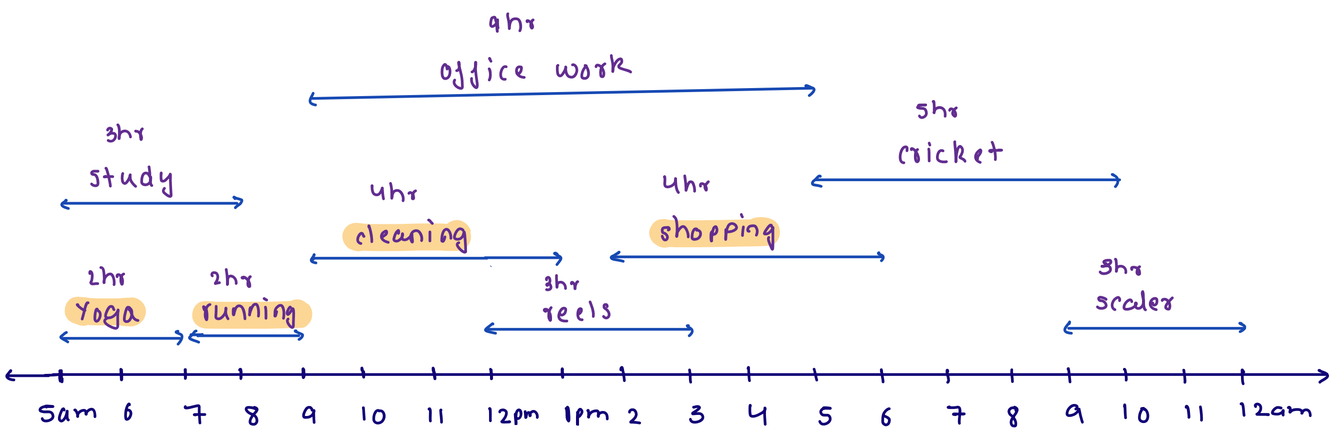
k = ~~7~~	ans = ~~0~~
~~6~~	~~15~~
~~4~~	~~35~~
~~1~~	~~53~~
break	58

Q.2 Activity Selection

Find max no. of task we can do.

Note : i) on starting a task, we need to complete it

ii) At any point of time do only a single task

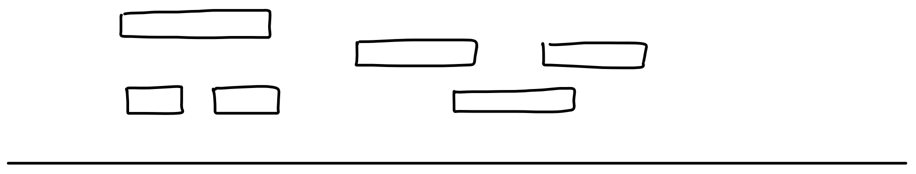


i) short duration tasks (x)

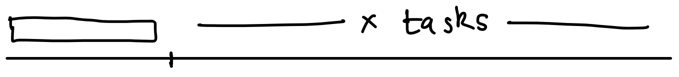
→ Yoga, running, reels, scaler   ans = 4

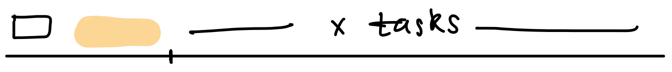
ii) Ending early (✓)

→ yoga, running, cleaning, shopping, scaler   ans = 5



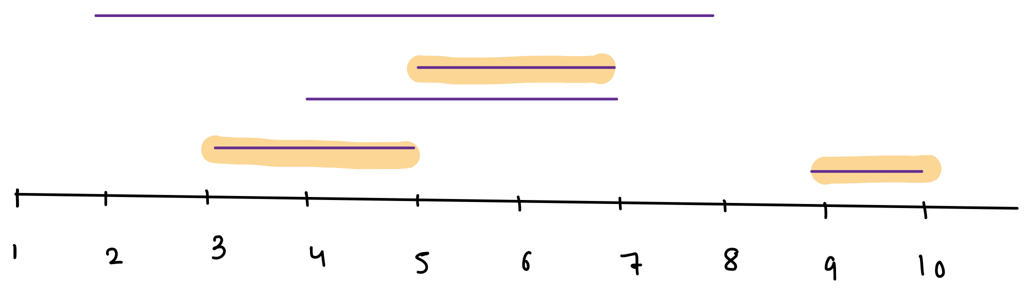
Correction





>= x tasks

	0	1	2	3	4
st :	4	5	3	9	2
et :	7	7	5	10	8



```
int solve( int[] st, int[] et) {
    int n= st.length;
    Pair[] arr= new Pair[n];

    for(int i= 0; i<n; i++) {
        Pair p= new Pair(st[i], et[i]);
        arr[i]=p;
    }

    Arrays.sort(arr, ___);     sort the arr based on ending
                               time of tasks [ascending order]

    // pick non-overlapping tasks
    int ans= 1;
    int ltet= arr[0].et;  // last task's ending time

    for(int i=1; i<n; i++) {
        // can I do ith task
        if(arr[i].st >= ltet) {
            ans++;
            ltet= arr[i].et;
        }
    }

    return ans;
}
```

```
class Pair {
    int st;
    int et;
}
```

TC: O(nlogn)

SC: O(n)

	0	1	2	3	4
st :	4	5	3	9	2
et :	7	7	5	10	8

4,7	5,7	3, 5	9, 10	2, 8
0	1	2	3	4

After sorting : (based on ending : ascending)

3, 5	5,7	4,7	2, 8	9, 10
0	1	2	3	4

```
// pick non- overlapping tasks
int ans= 1;
int dtet= arr[0].et; // last task's ending time

for (int i=1; i<n; i++) {
        // can i do ith task
        if (arr[i].st >= dtet) {
            ans++;
            dtet= arr[i].et;
        }
}

return ans;
```

ans = ~~1~~ ~~2~~ 3

dtet = ~~5~~ ~~7~~ 10

Q.3 Job scheduling

Given N tasks to complete

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

→ Payment assigned to each task

→ on any given day we can perform only 1 task and each task take 1 day to finish.

→ find max payment we can get.

duration of each task → 1 day

eg1

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

sort on the basis of deadline

dd :	1	1	2	2	3
pay :	25	19	30	27	100

eg2

Job	deadline	Payment
a	3	5
b	1	1
c	3	6
d	2	3
e	3	9

The step after sorting pair array :

	0	1	2	3	4
dl	1	1	2	2	3
pay	25	19	30	27	100

27
30
100

minPQ (Integer)

	0	1	2	3	4	5	6	7	8	9
dl	1	1	1	2	2	3	3	3	4	4
pay	10	15	12	18	10	25	15	30	8	16

```
if (arr[i].dl > pq.size() {
|       pq.add(arr[i].pay);
}
else {
|       // replace if you want
|       if (pq.peek() < arr[i].pay) {
|           pq.remove();
|           pq.add(arr[i].pay);
|       }
}
```

30
18
25
16

min PQ

code : todo
(refer to ide)