Greedy

M T W T F S S | Page No.: | Date: | YOUVA

① LeetCode Q. No. (1005) { Maximize Sum of Array After K Negations. }

Need to Apply process 'k' times.
↳ Every time you must need to negate a one ele.
↳ -1 ⟶ +1
+1 ⟶ -1

Ex ①
nums = {4, 2, 3} k = 1
→ -4, 2, 3 → Ans = 1
→ 4, -2, 3 → Ans = 5 → Maxⁿ.
→ 4, 2, -3 → Ans = 3

You can chose same ele multiple time.
Need To return maxⁿ sum.

Ex ②
nums = {3, -1, 0, 2} k = 3 → Need to perform operation 3 times.
→ {3, 1, 0, 2} → {3, 1, -0, 2} → {3, 1, 0, 2}
Ans ⇒ 6 (maxⁿ)

Logic.

Ex ③ nums = {2, -3, -1, 5, -4} k = 4.
⇓ sort
{-4, -3, -1, 2, 5}
{4, -3, -1, 2, 5} k = 3
{4, 3, -1, 2, 5} k = 2
{4, 3, 1, 2, 5} k = 1 → Break loop. nums[i] != 0.

Do this if nums[i] < 0.

```
if (k%2 == 1) {
    sort(nums);
    nums[0] *= -1;
}
return sum;
```

② Q.fg. { Fractional Knapsack } → You can take fraction also.
Ex:- arr {{100, 20}, {60, 10}, {100, 50}, {200, 50}} C ⇒ 90 → Capacity.

{20, 10}
{200, 50}
(60, 10)
(100, 20)

C ⇒ 90 70 60 10.0
wt ⇒ 100 + 200 + 60 + 20.
⇒ 380 → maxⁿ.

(40, 10)
(100, 50)
(60, 10)
(100, 20)

C ⇒ 90 30 60 0.
wt ⇒ 40 + 100 + 60 + 100
⇒ 300. → Need to maximize

5 6 2 4 # use Comparator.

arr = {{100, 20}, {60, 10}, {100, 50}, {200, 50}}

↓ sort based on arr.ratio.

{{60, 10}, {100, 20}, {200, 50}, {100, 50}}

↓ curCap = 0, finalvalue = 0;

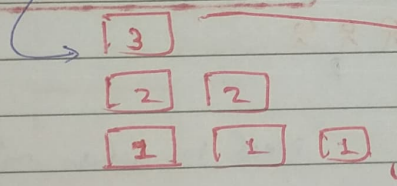
```
for(i = 0 to i<n){
    if(curCap + arr[i].wt <= c){
        curCap += arr[i].wt;
        finalvalue += arr[i].val;
    }
    else{
        // Calculate fractional value. And add to finalvalue.
        break;
    }
}
```

(3) Leetcode Q.No (1710) { Maximum Units on a Truck }

Ex :- (1).

boxTypes = {{1, 3}, {2, 2}, {3, 1}} (1 → No. of Boxes, 2 → Units per Box)

truckSize => 4,



3 + 2 + 2 + 1 => (1) → Ans.

Truck Size

Need to Return Max^m total No. of units.

Logic :-

Ex (2).

boxTypes = {{5, 10}, {2, 5}, {4, 7}, {8, 9}} truckSize => 10.

↓ Sort based on unit size

{{5, 10}, {3, 9}, {4, 7}, {2, 5}},

```
for(int i = 0 ; i < n ; i++){
    No.ofBox => boxTypes[i][0];
    NoofUnits = boxTypes[i][1];
    min = (trucksize , NoofBox);
    trucksize -= min;
    sum += min * No.of Units;
    if(trucksize == 0){
        return sum;
    }
}
return sum;
```

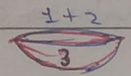
(4) LeetCode Q.No. (881) {Boats to Save Peoples},

Page No.:	YOUVA
Date:	

Ex (1) :-

people = { 1 , 2 } → weight   limit = { 3 } → wt. Limit of boat.

Need to return min^n boats require to carry given people



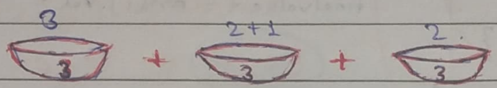
⇒ Ans ⇒ (1)

→ 1 boat can max^n carry 2 Peoples

⇒ people[i] <= limit

Ex (2)

people = { 3 , 2 , 2 , 1 } limit = { 3 }



Ans → (3).

Logic. Sort

{ 1 , 2 , 2 , 3 } (i) at 1, (j) at 3

1 → Lightes person. 3 → heaviest person

```
while (i <= j) {
    if (p[i] + p[j] <= limit) {
        i++;
        j--;
    }
    else {
        j--;
    }
    count++;
}
```

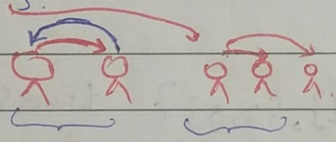
(5) LeetCode Q.No. (781) { Rabbits In Forest. },

Ex (1) ⇒ answers = { 1 , 1 , 2 } (indices 0, 1, 2)

R R R (R)   R R (G)   R R (B)

Need to return min^n No. of Rabbits in Forest

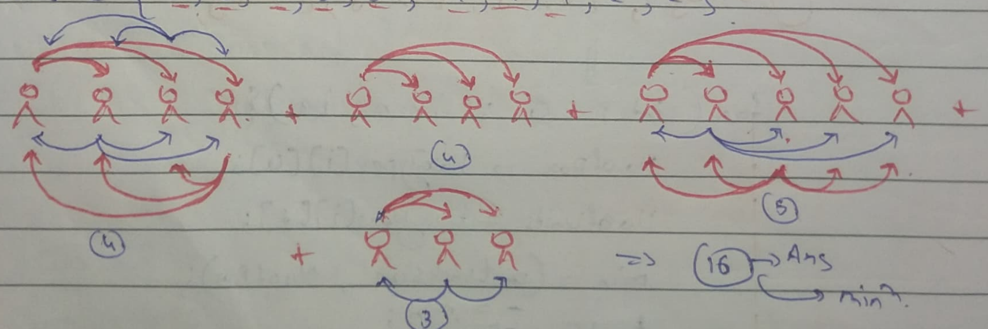
{ 1 , 1 , 2 }



2 + 3 ⇒ (5) → Ans. → min^n.

Ex (2) :-

answers = { 3 , 3 , 3 , 3 , 3 , 4 , 4 , 4 , 2 , 2 } (indices 0 1 2 3 4 5 6 7 8 9)



⇒ (16) → Ans → min^n.

Logic → Create frequency map.

arr = {3, 3, 3, 3, 3, 4, 4, 4, 2, 2}

↳ key → arr[i]+1 (freq)

4 → 5

5 → 3

3 → 2

q => Value/key

=> 5/4 = (1)

No. of group required

r = Value % key

↳ 5%4 => 1

if (r != 0) { → means ek aur group ki zaroorat hai

ans += key

}

ans += q * key

arr = {3, 3, 3, 3}

↳ key → (Value) frequency

4 → 4

q = 4/4 => (1) → No. of group required

r = 4%4 = 0

Ans += q * key

= 4 * 1 = 4

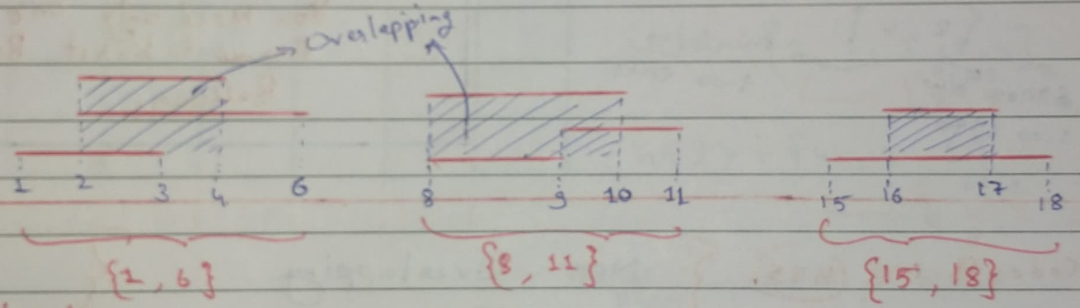
Ans => (4)4

---

(6) LeetCode Q.No (56) { Merge Intervals. }4

Ex(1).

arr = { {1, 3}, {2, 6}, {8, 9}, {9, 11}, {8, 10}, {2, 4}, {15, 18}, {16, 17} }



Sort based on arr[i][0]

arr { {1, 3}, {2, 6}, {2, 4}, {8, 9}, {8, 10}, {9, 11}, {15, 18}, {16, 17} }4

Traverse

ans = { {1, 6}, {8, 11}, {15, 18} } => Ans

```
if (ans.size() == 0 || arr[i][0] > ans.get(size()-1)[1]) {
    ans.add(arr[i][0], arr[i][1]);
}
else {
    ans.get(size()-1).set(1, Math.max(ans.get(size()-1).get(1),
                                      arr[i][1]));
}
```

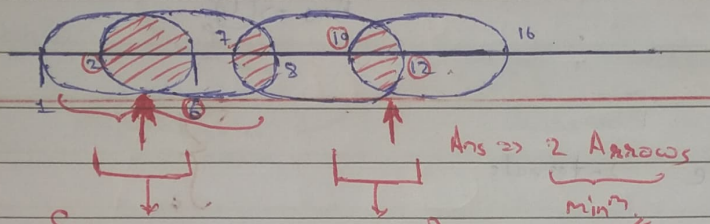
⑦ LeetCode Q. No. (452) { Minimum No. Of Arrows to Burst Balloons.

Ex① :- X start, X end.

points = { {10, 16}, {2, 8}, {1, 6}, {7, 12} }

Balloons.

⇓ Sort based on X start.



Ans => 2 Arrows (Min^m)

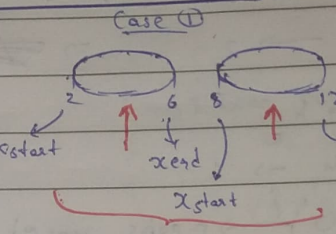
Ans => { {2, 6}, {10, 12} } → We will store this

→ return ans.size();

{2, 6} → Min b/w two ends.

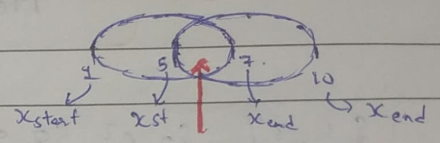
& Max b/w two starts

Case ①



You Need Two Arrows to Burst Both the Balloons

Case ②.



You Need only One Arrow to burst Both Balloons

---

⑧ LeetCode Q. No. (435) { Non - Overlapping Intervals.

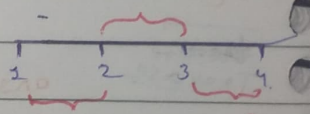
Followup to previous one

Ex :-

intervals := { {1, 2}, {2, 3}, {3, 4}, {1, 3} }

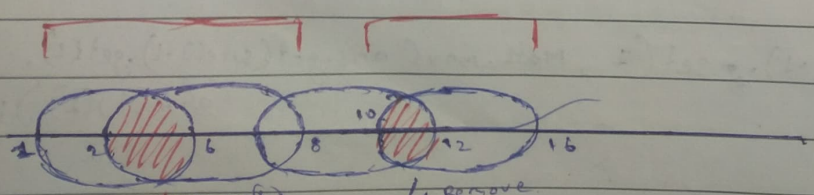
=> Count Non-Overlapping intervals, same as previous Question.

=> Then Return intervals.size() - ans.size();

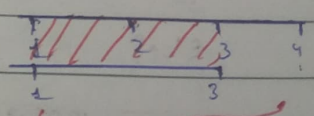


These are Non Overlapping

ans = { {10, 16}, {2, 8}, {1, 6}, {7, 12} } => size => (4)



Ans => { {2, 6}, {10, 12} } → size => (2)



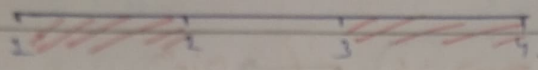
Overlapping we Need to remove (1, 3).

Ans => 4 - 2 => (2) { Min^m No. of intervals need to Remove

(9) LeetCode Q.No. (646) { Maximum Length of Pair Chain }

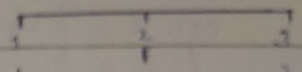
Ex (1):

pairs = { {1,2}, {2,3}, {3,4} }.



{1,2} --> {3,4}.

Max Length => (2) => Maximum.



This will not form Chain they are Over lapping

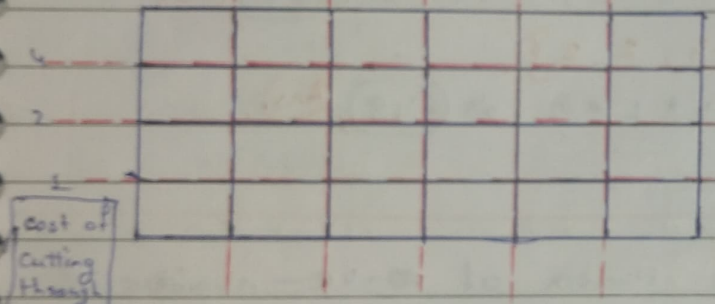
=> Code:

```
for(int[] pair : pairs){
    if(prend < pair[0]){
        count++;
        prend = pair[1];
    }
}
```

(10) gfg { Minimum Cost to Cut A Board into Squares }

{2, 1, 3, 1, 4} Costs of Cutting through these edge

* Cost => edge Cost x No. of Pieces

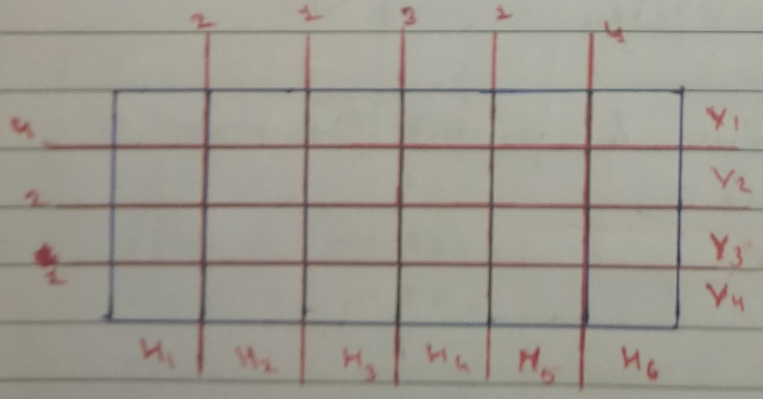


int[] x = {2, 1, 3, 1, 4}
0 1 2 3 4
→ {4, 3, 2, 1, 1}
1

int[] y = {4, 2, 1}
0 1 2
→ {4, 2, 1}
0

Vertical Pieces => ~~1 2 3 4 5 6~~

Horizontal Pieces = ~~1 2 3 4~~



Horizontal Cut => Cost * Vertical Pieces

Cost = 0 + (4×1) + (4×2) + (3×2) + (2×3) + (2×3) + (1×4) + (1×4) + (1×4)

Cost = (42) => Ans.

M T W T F S S | Page No.: | Date: | YOUVA

(11) Minimum Product Subset of An Array.

Ex (1) :

arr = { -1, -1, -2, 4, 3 }.
→ Ans => (-24) => (-2 * -1 * -1 * 4 * 3) = -24

Ex (2) :-

arr = { -1, 0 } → -1 → (-1) } → min^m.
→ -1 * 0 → 0
→ Ans => (-1)

Case : (1)
→ If there are even No. of Negative No.'s & [No Zeroes] the result is the [Product of all] except the largest valued negative No.
→ arr { (-1), -2, -3, -4 } [largest -ve No.]
→ -1 * -2 * -3 * -4 => 24
→ -2 * -3 * -4 => (-24) (Ans)
→ arr { -1, -2, -3, -4, 1, 2 }.
→ -2 * -3 * -4 * 1 * 2 => (-48) (Ans).

Case : (2).
→ If There are An odd Number of -ve numbers & No zeroes, the result is simply product.
→ arr = { -1, -2, -1, 4, 2 }
→ (-24) Ans

Case : (3)
→ If there are zeroes & positive, No -ve the result is 0.
→ arr { 0, 1, 2, 3 }
→ Ans 0

Exceptional case:-
→ when there is No negative & All other elements is +ve then result is first min^m +ve number.
→ arr = { 1, 2, 3, 4 } (1 = min^m)
→ Ans → (1)