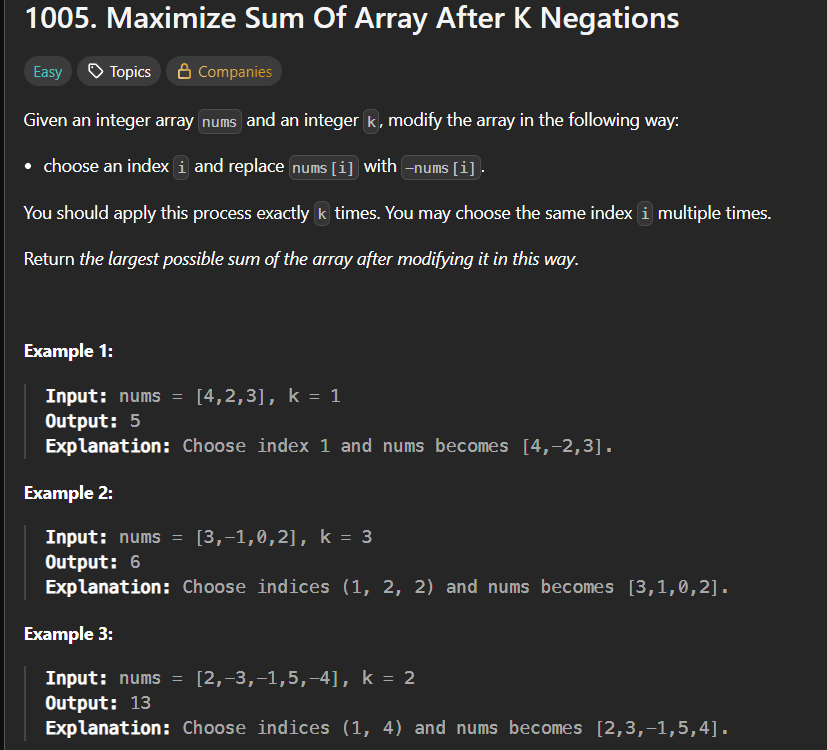
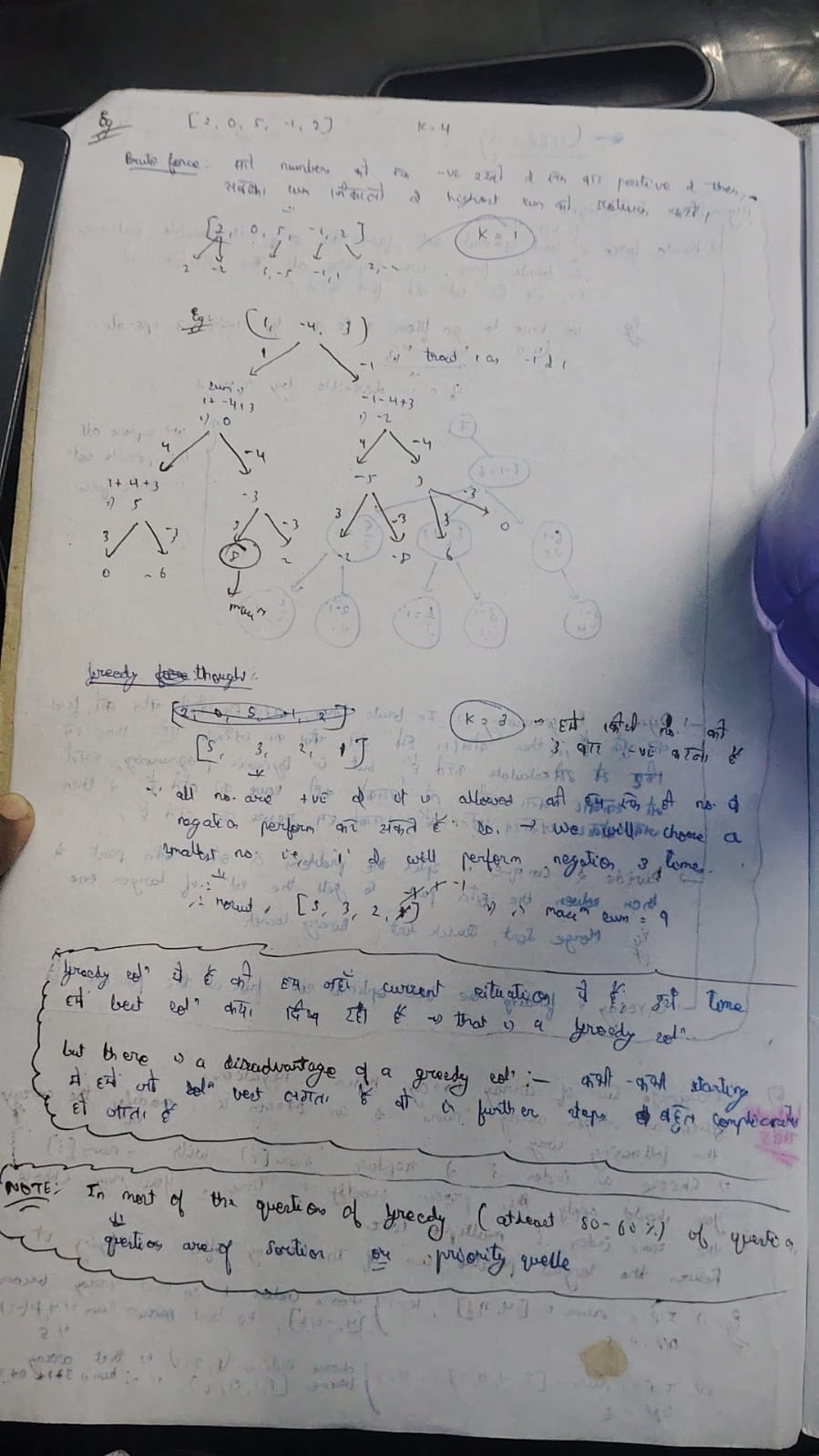
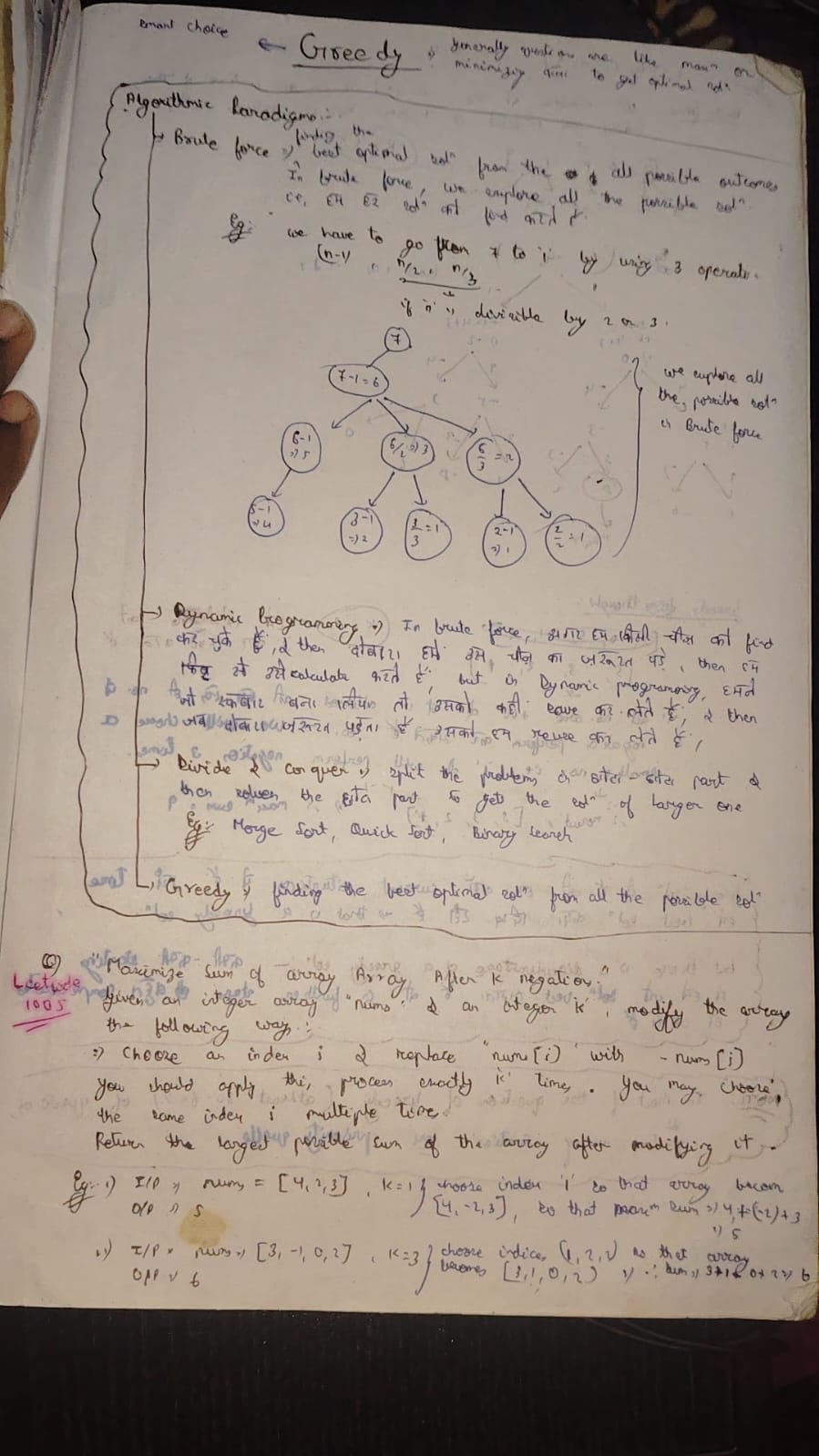
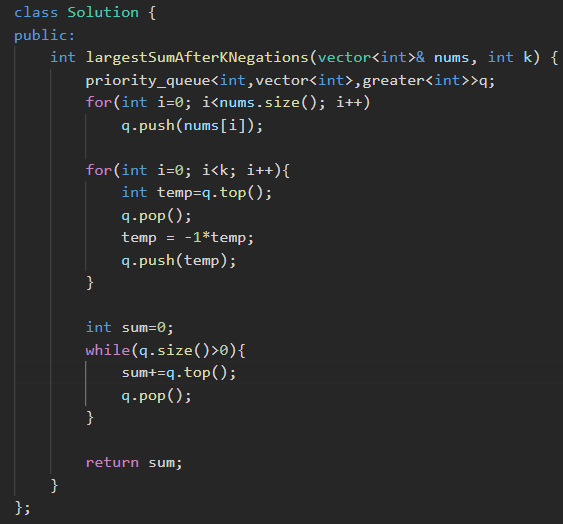
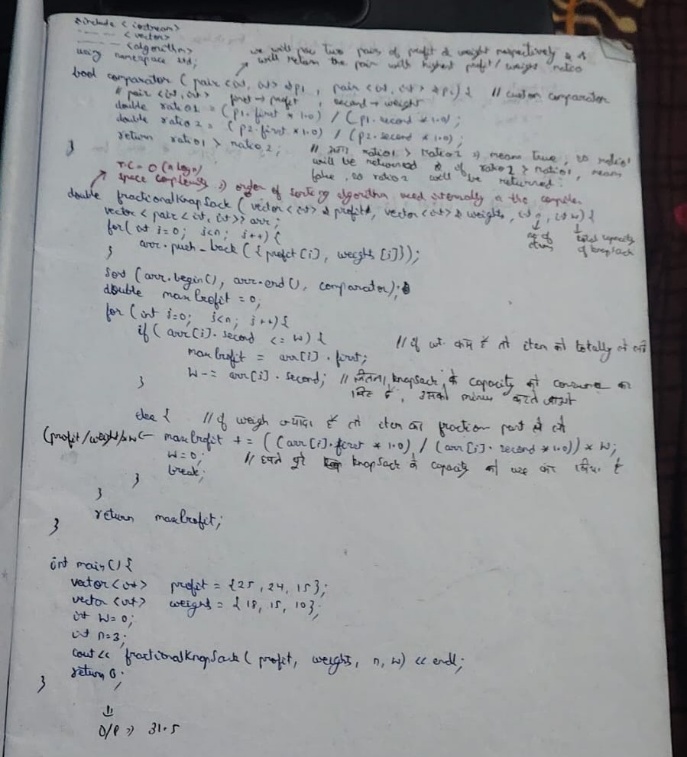
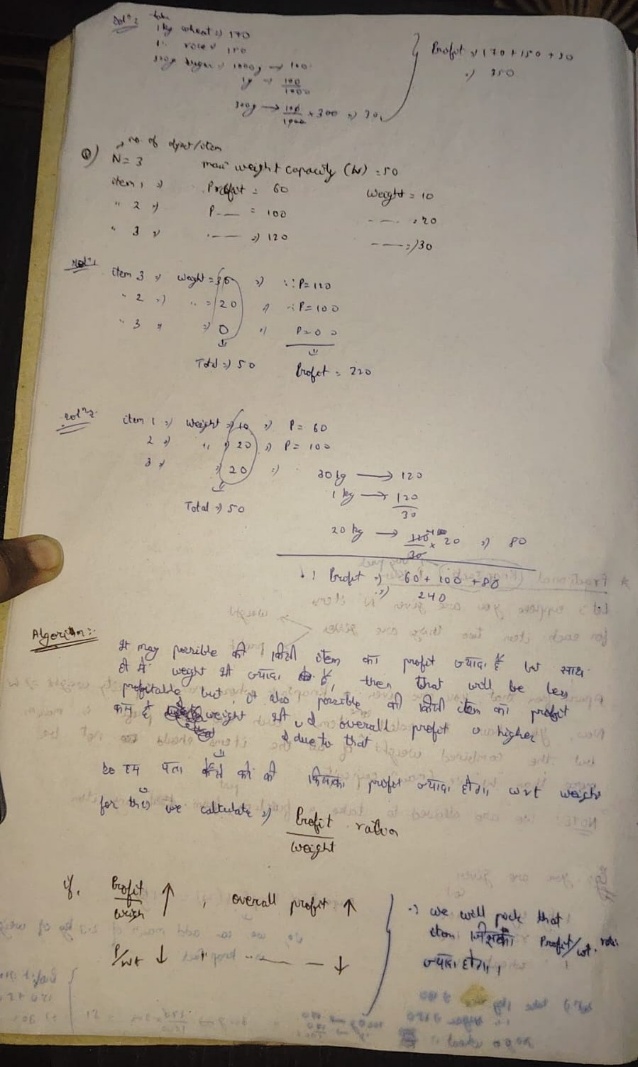
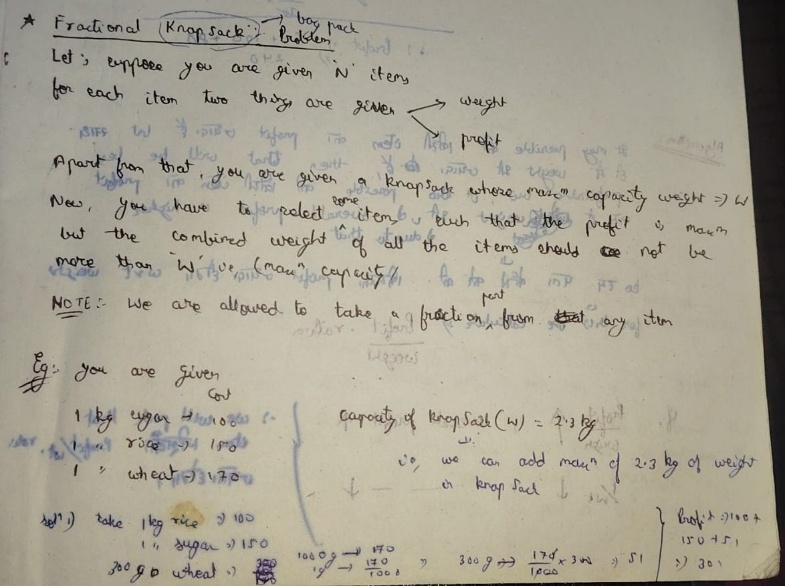
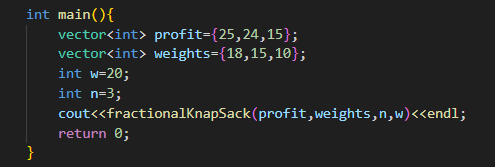
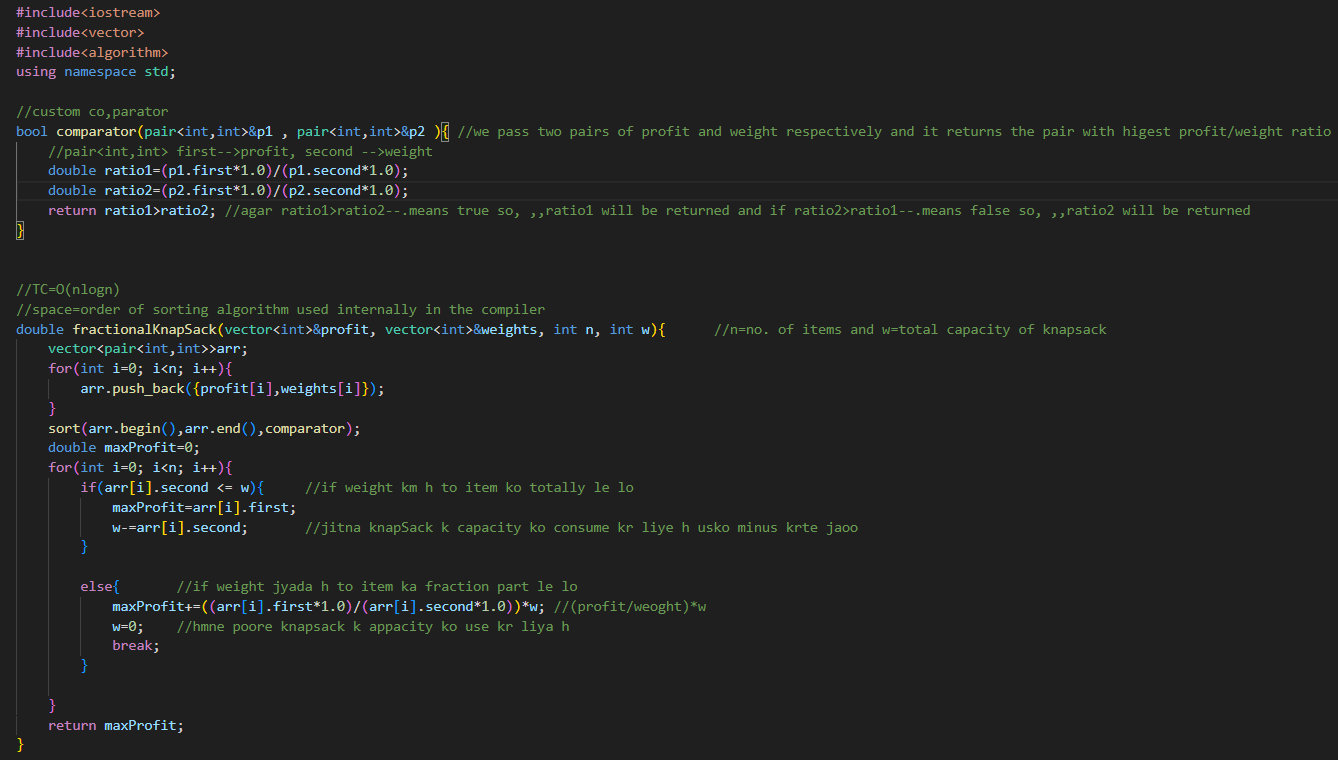
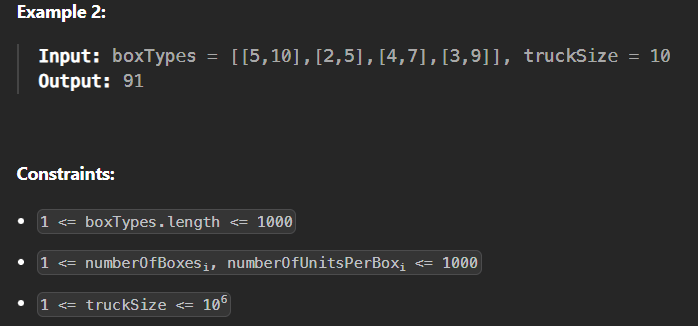
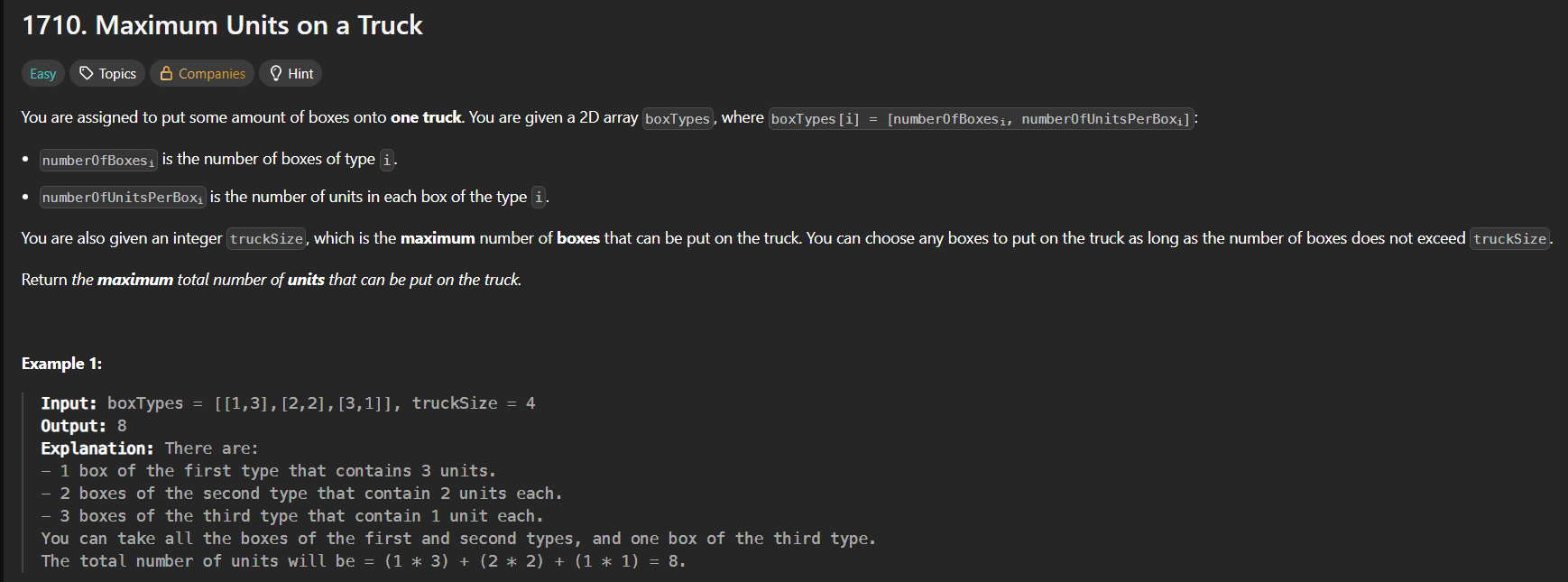
**Greedy Algorithms**

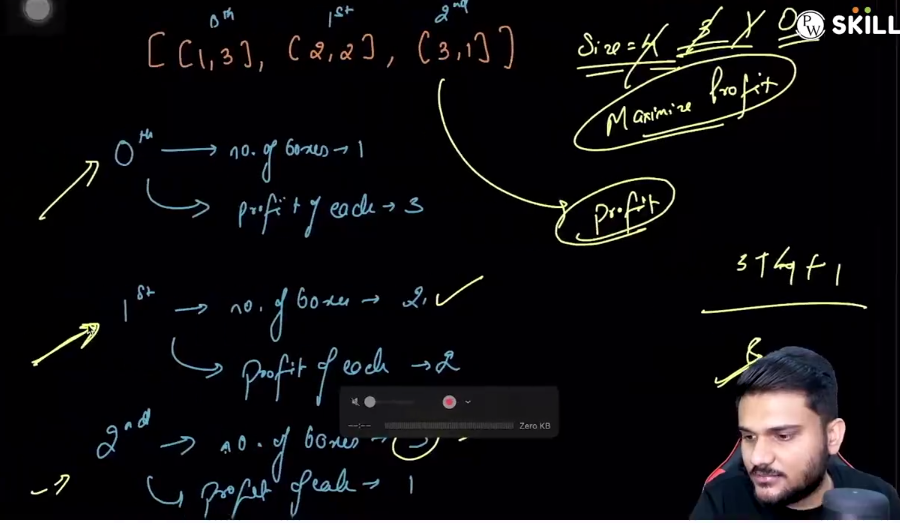
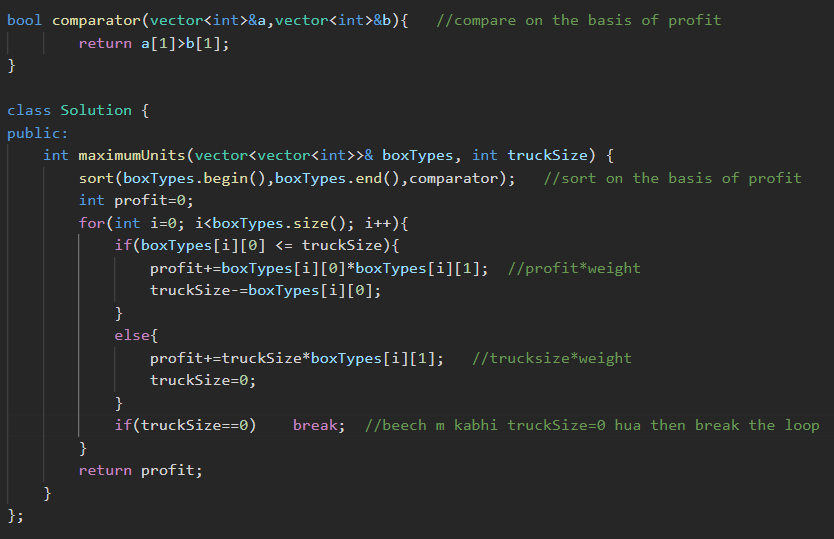
🡪explanation in above notes

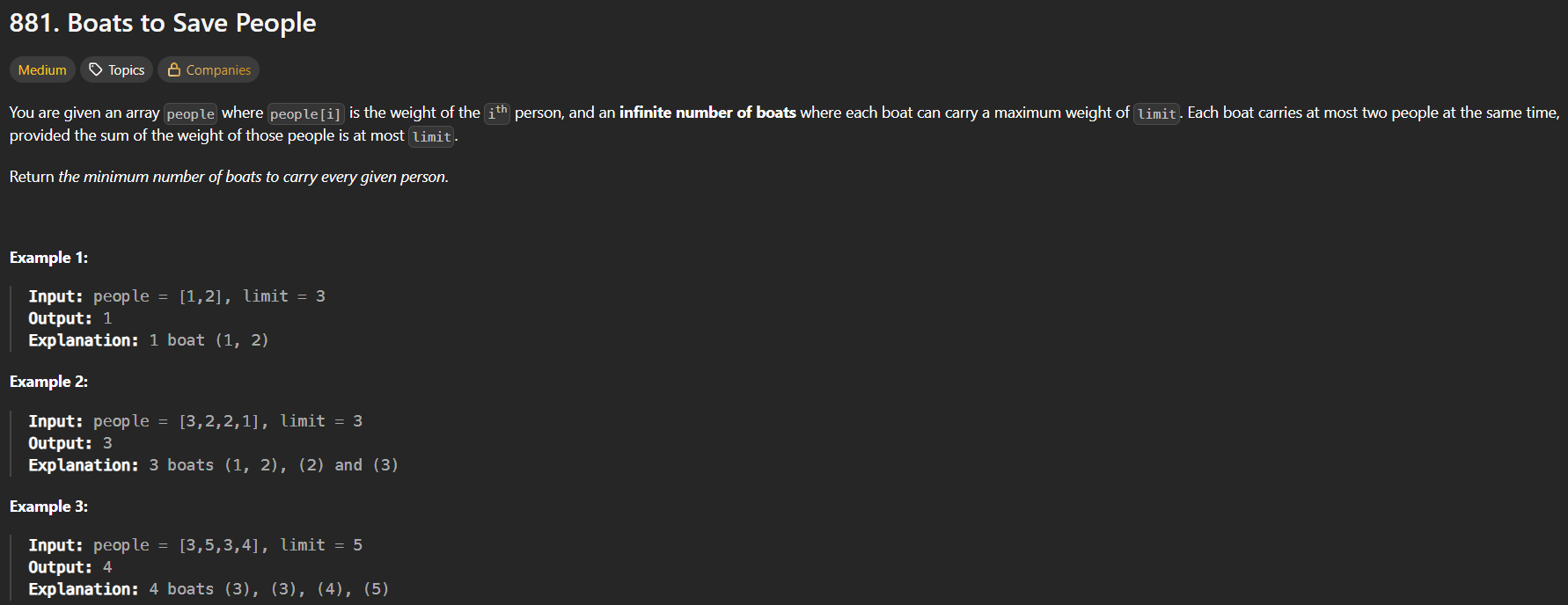
* Algorithm 🡪 ek minheap bnaao, and saare elements ko usme daal ko, and then har baar sbse Chhota element choose kro for k times, and then usko negate krte jaoo, lastly saare elements ka sum return kr do. 

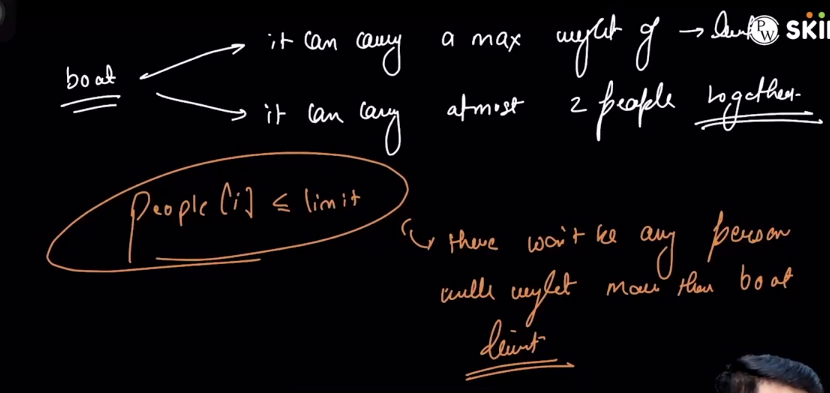
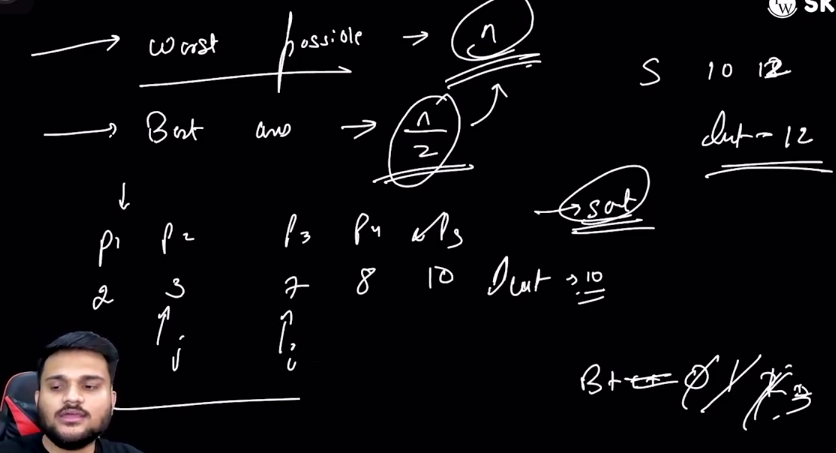
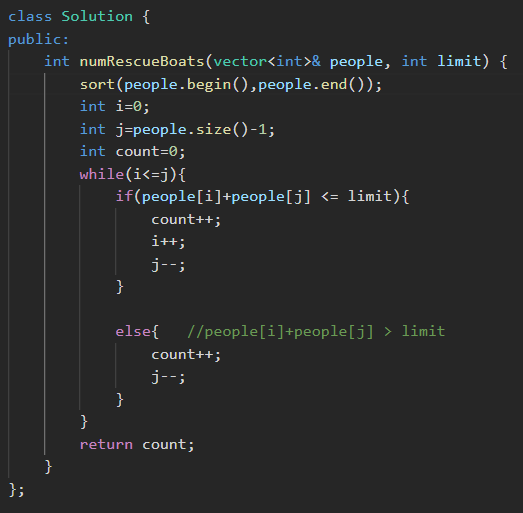


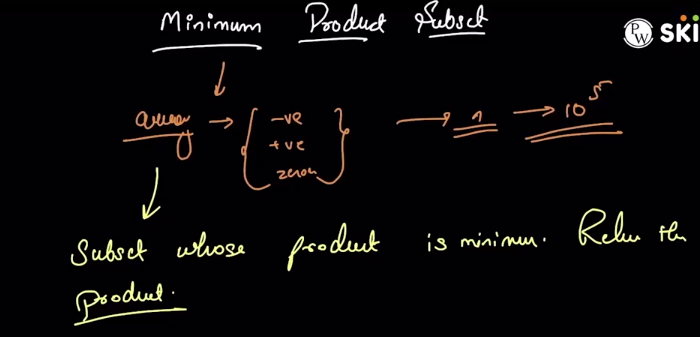
* 

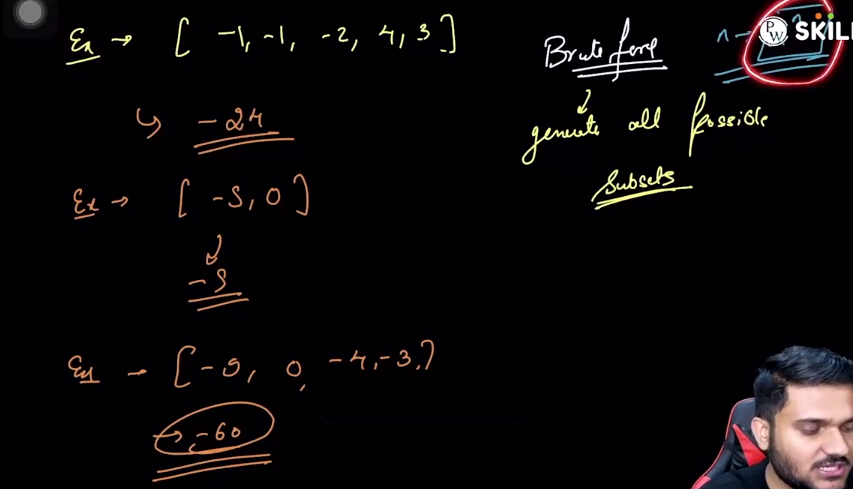
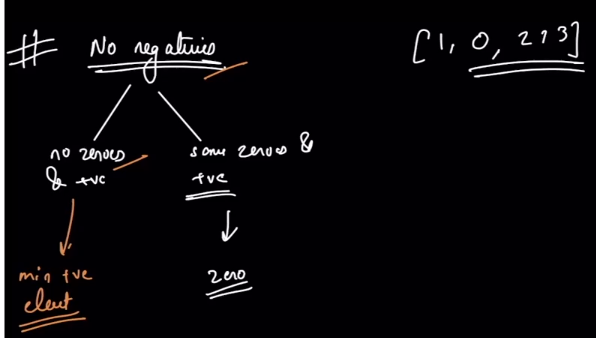
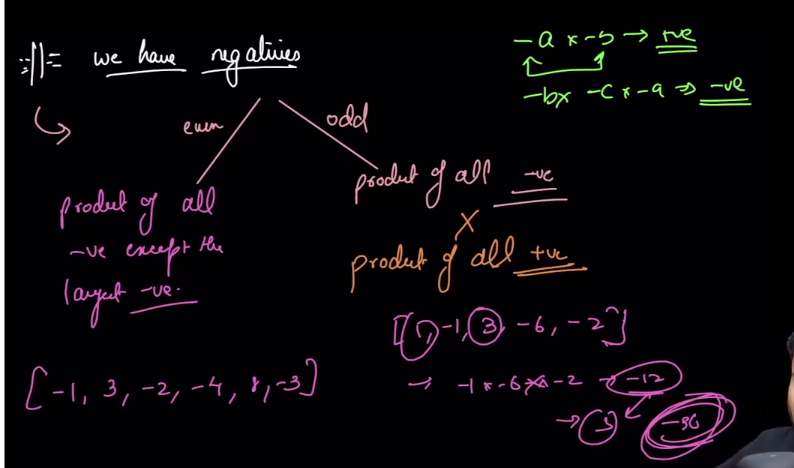
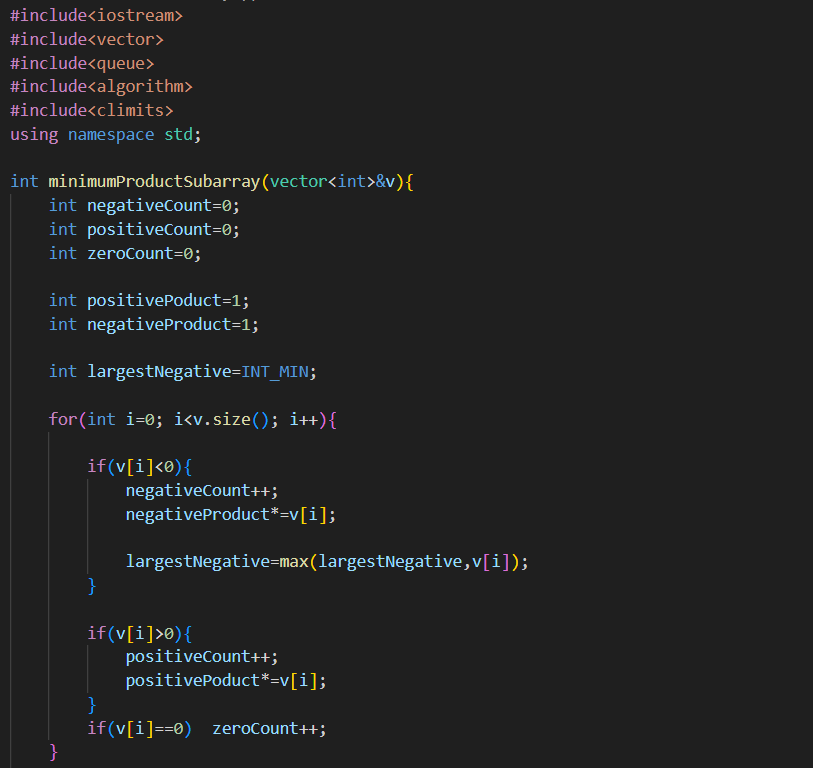
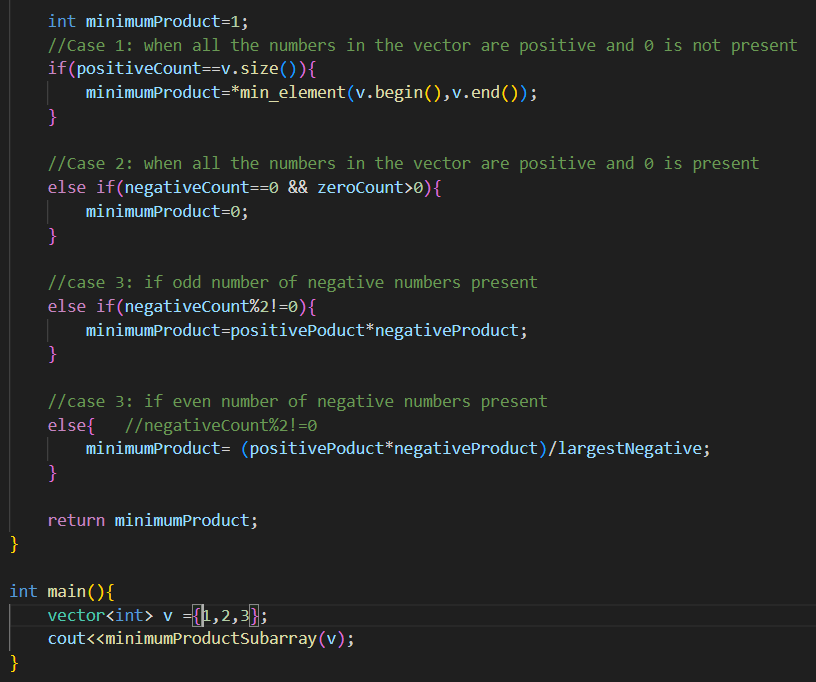
🡪 question is of fractional knapsack

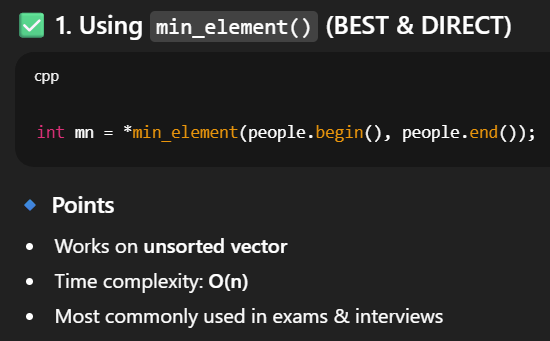
* 
* 

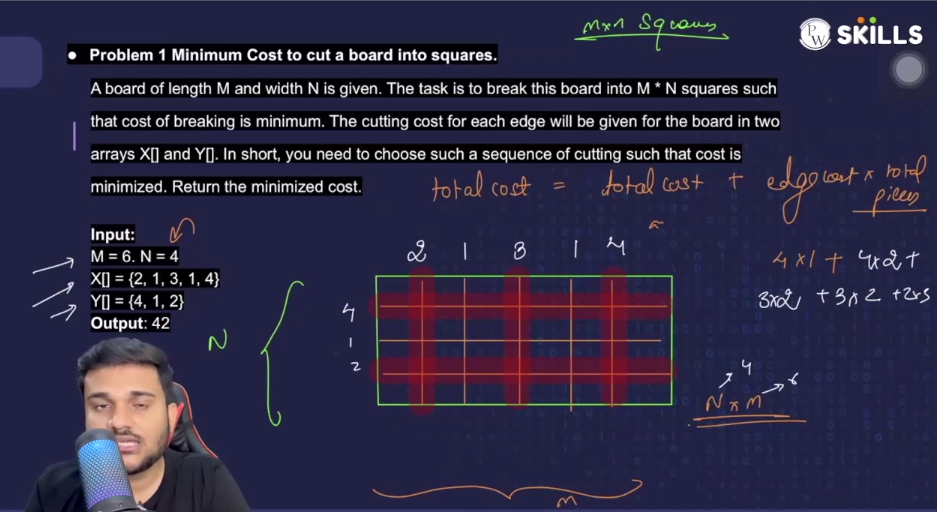


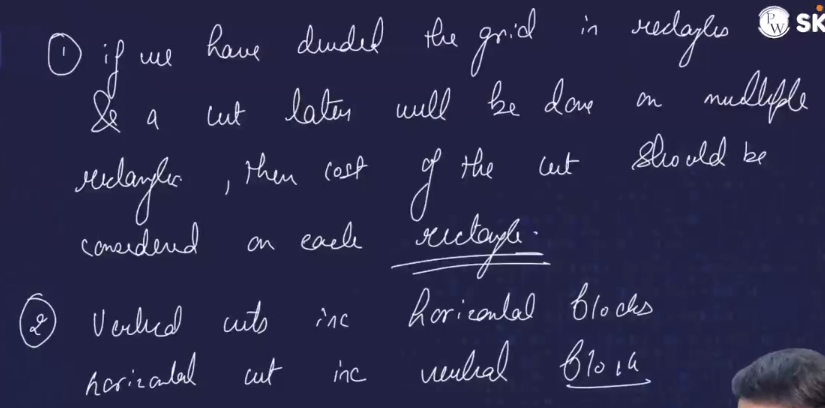
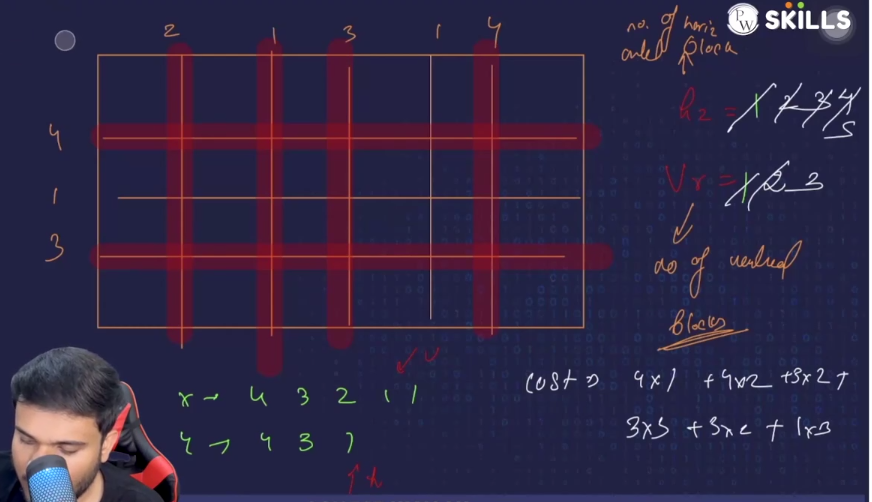
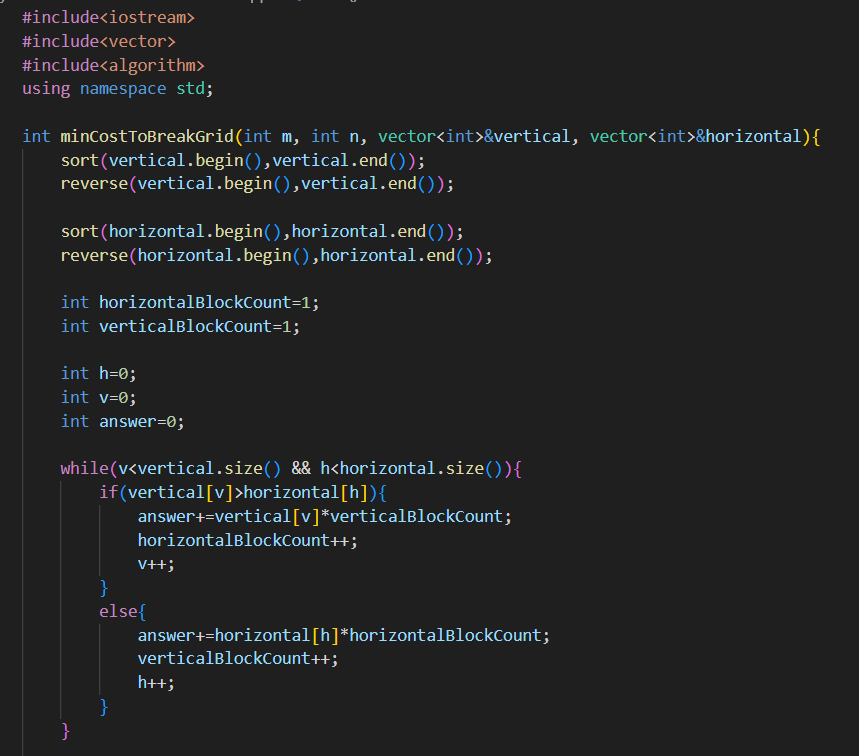
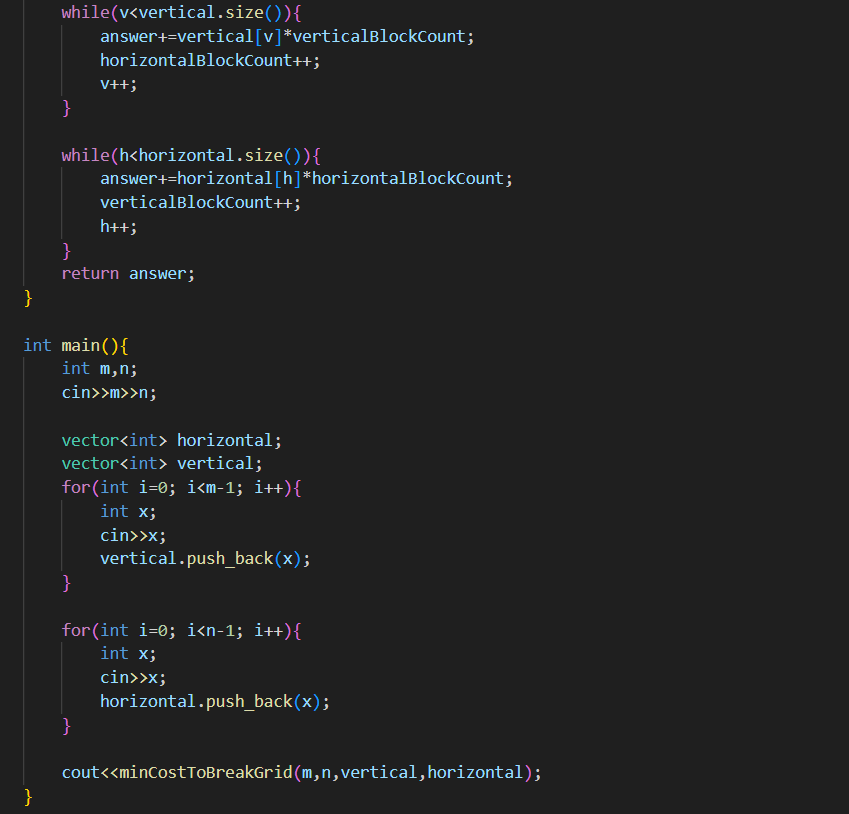
* 
* 🡪algorithm 🡪 1st sort the vector and then use two pointer algorithm🡪 **i** ko starting m rakho and **j** ko end m, check krlo ki sbse bada and sbse Chhota saath jaa skta h ki nhi, if jaa skta h then count ko badha do and i ko ek aage krdo and j ko peeche , and if dono saath m nhi jaa skte then, sirf bade waale ko saath lekar jaoo, and j ko km krdo and count ko badha do. 🡪T.C 🡪 O(nlogn) & S.C 🡪O(sorting)

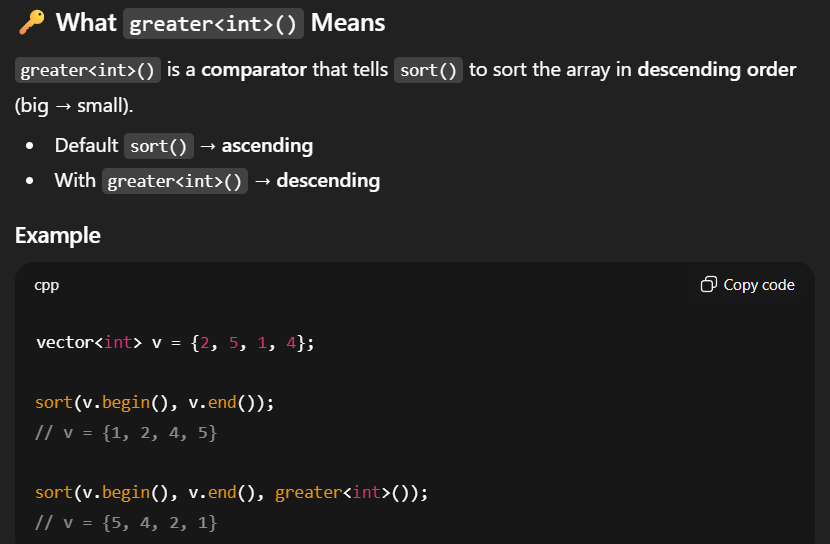
🡪Question says🡪hme ek array given hoga that can contain +ve, -ve and 0 all. We have to find the subset from the array that will give the minimum product.

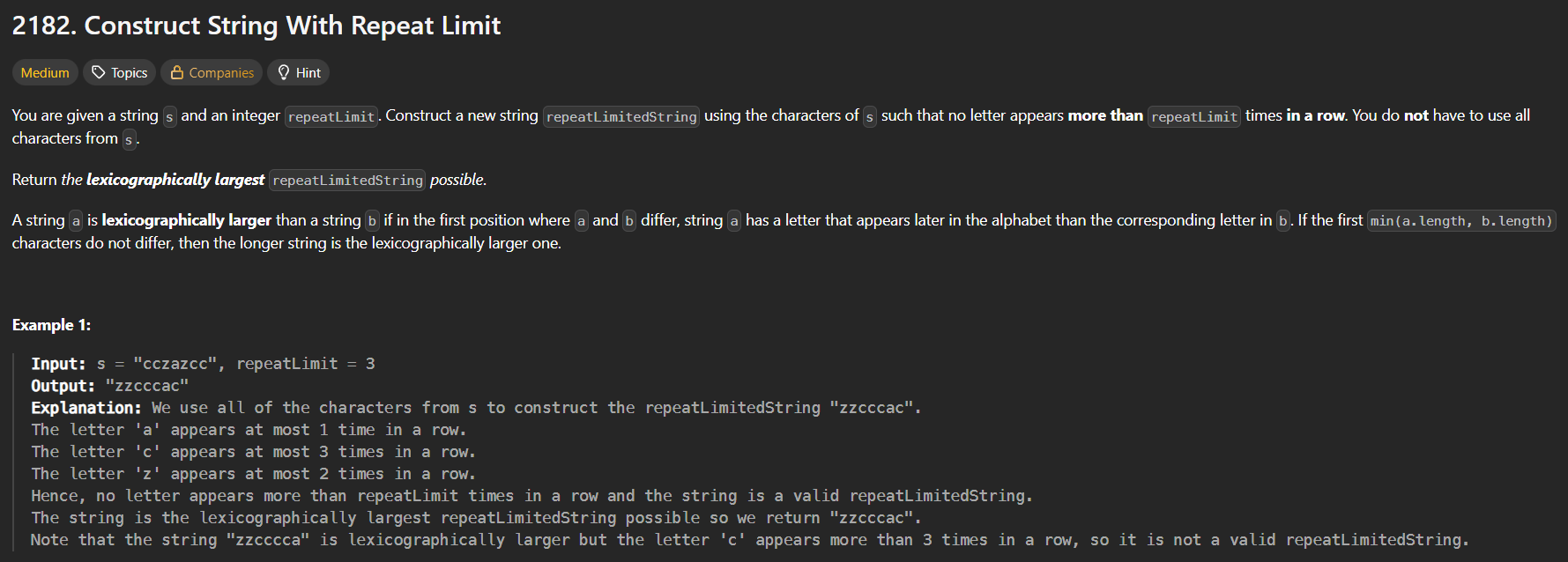
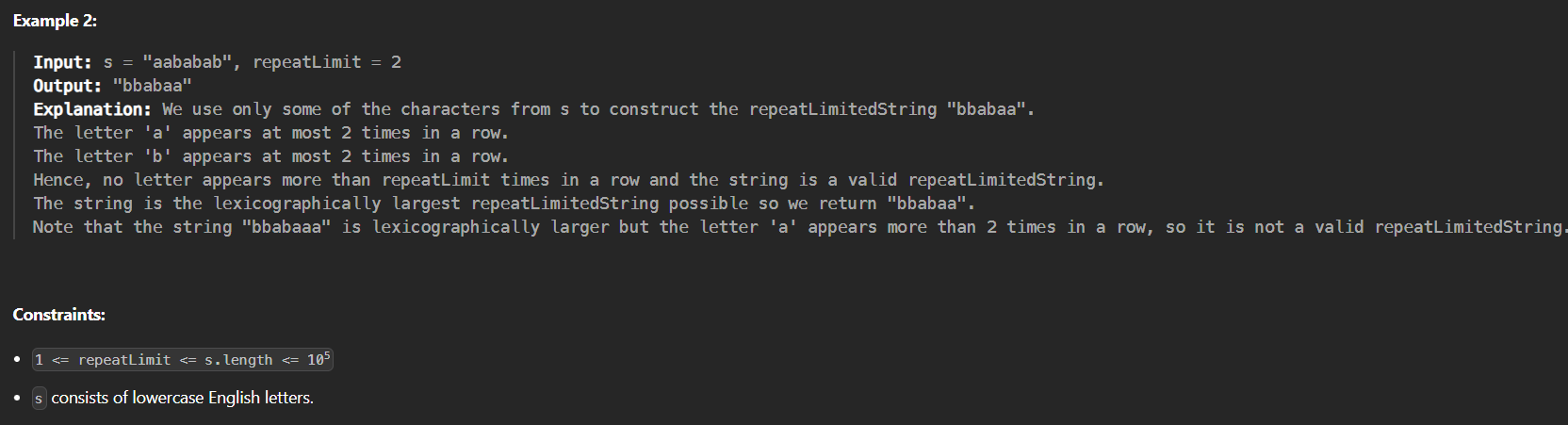
* Method 1🡪 brute force🡪generate all possible products and find the minimum one.
* Method 2🡪 greedy
  + Case 1: if we have no negative numbers in the array
    - If 0 is not present 🡪 then the minimum element is the minimum product.
    - If 0 is present 🡪 then 0 is the minimum product.
    - 
  + Case 2: if we have negative numbers in the array
    - If odd numbers of negative numbers present then 🡪 (product of all +ve nos) \* (product of all -ve nos)🡪will give the minimum product
    - If even numbers of negative numbers present then 🡪 (product of all +ve nos) \* (product of all -ve nos except the maximum -ve number)🡪will give the minimum product
    - 
  +  

**Note: we can directly fetch the minimum element from the vector using min element** **** **🡪similarly max\_element also exists in vector**

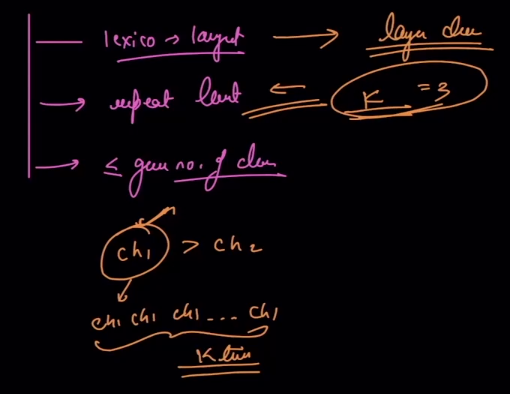
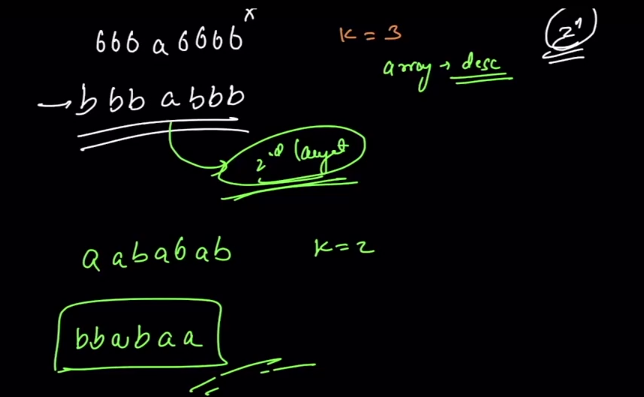
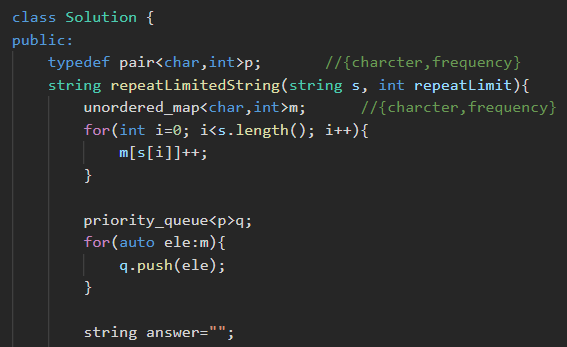


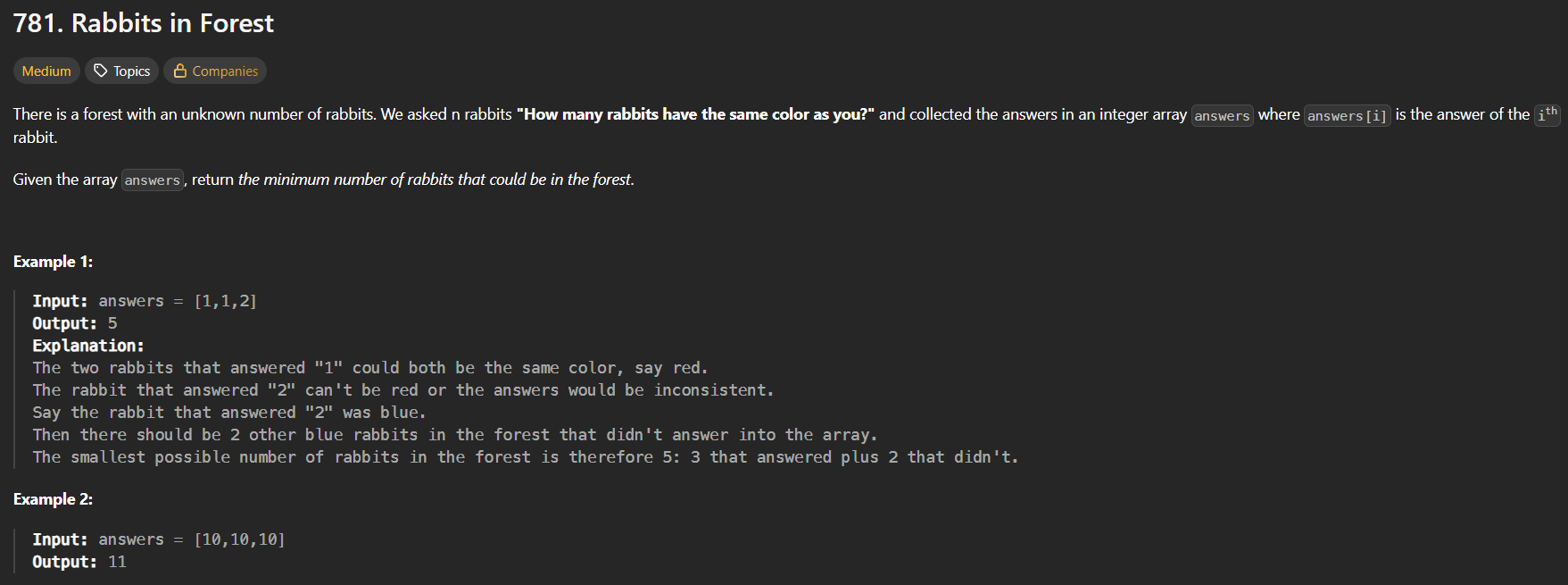
* 
* 
*   

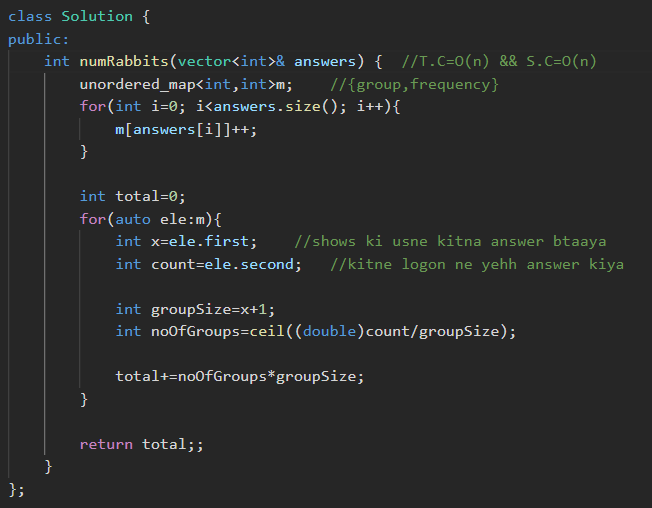
**Note:** 

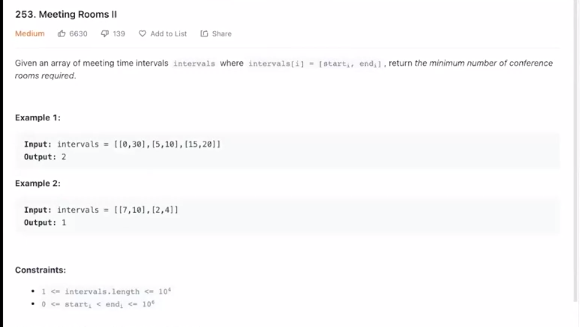
 🡪we have to find lexographically largest element (by considering the repeatLimit i.e element cannot be repeated more than this limit at once)

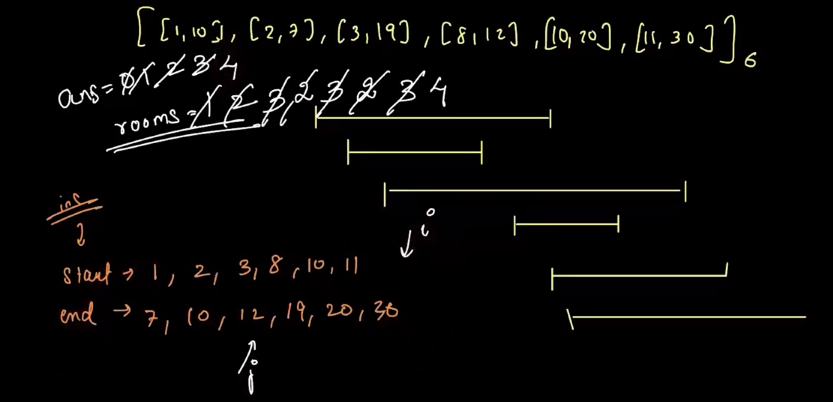
🡪lexographically largest element means🡪bada element pehle aana chahiye i.e b is larger than a.

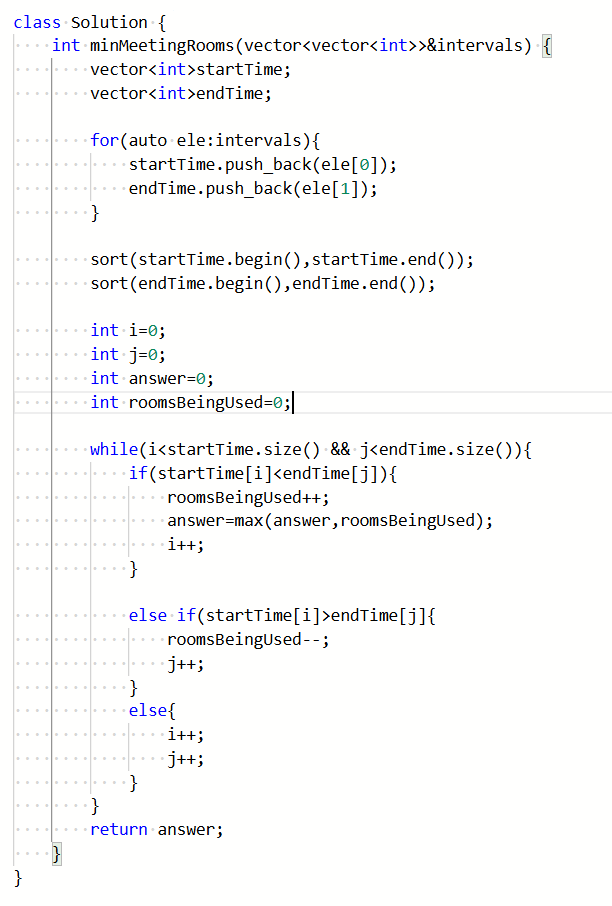
* 
* Eg🡪 
* Algorithm🡪sbse pehle jo largest element h , usko repeatLimit times likh do then second largest ko ek baar likh do and then again largest ko likho repeatLimit times (if characters are available in the string)  



* 

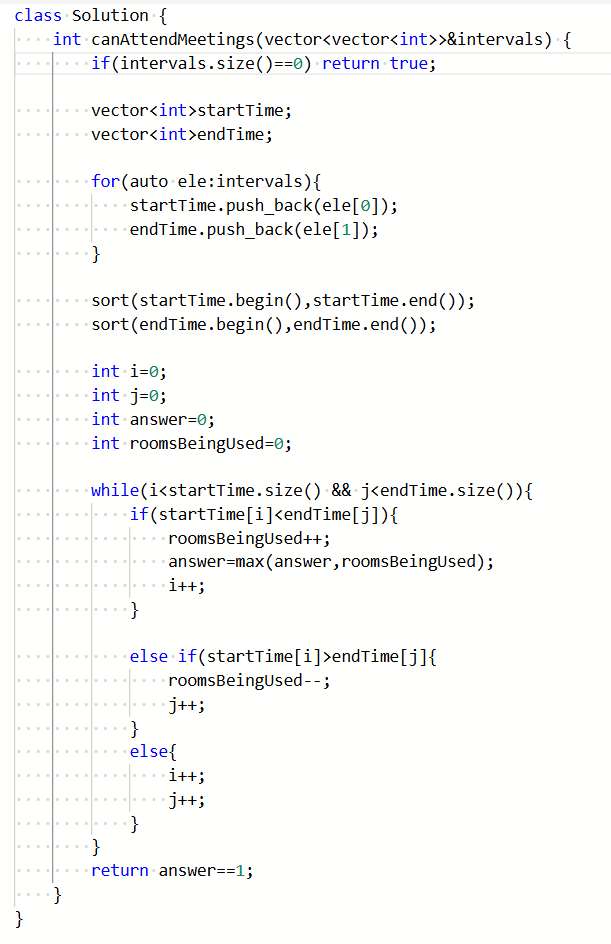
🡪premium question on leetcode🡪so use Lintcode

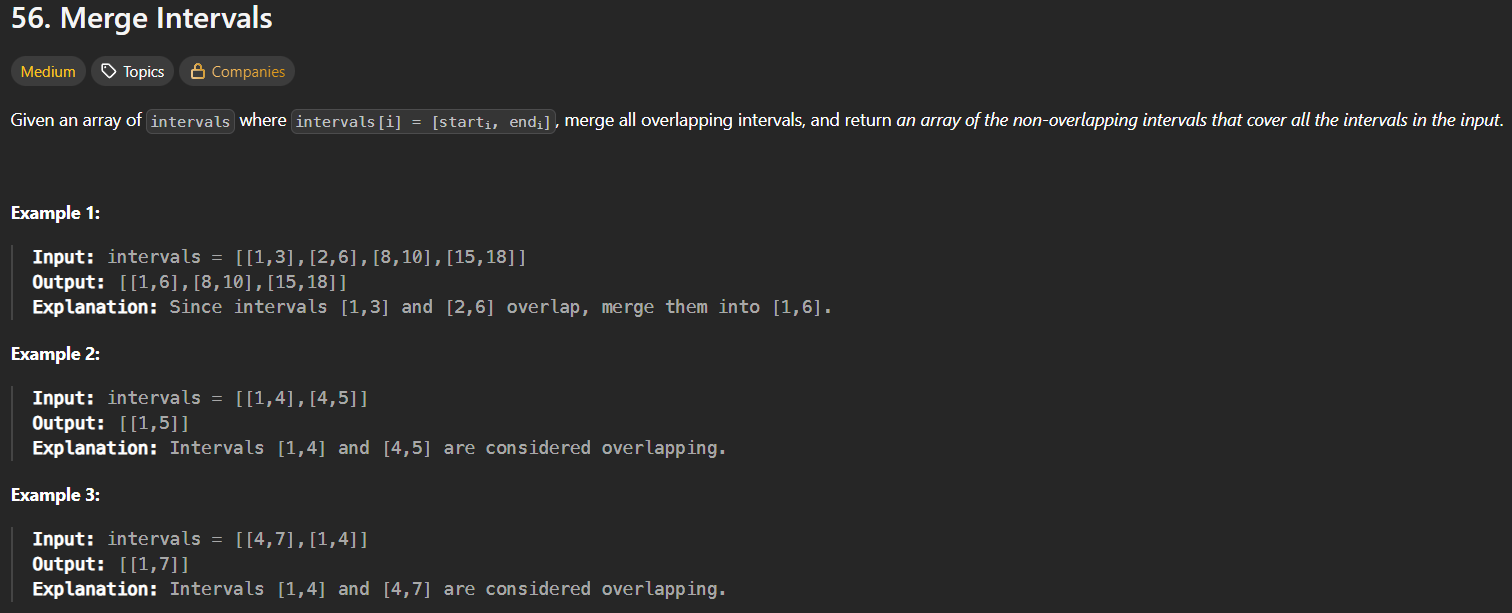
* 🡪 algorithm🡪pehle start time and end time vectors ko sort krlo. Now 2 variables bna lo “answer” (that stores total number of required rooms) and roomsBeingUsed (that stores how many rooms are being used currently). Then by using two pointers algorithm, iterate on both vectors, and compare start time and end time,
  + if start time<endTime 🡪 answer++ & roomsBeingUsed++
  + if start time>=endTime🡪new meeting started and previous one ended 🡪 so answer++ && roomsBeingUsed—

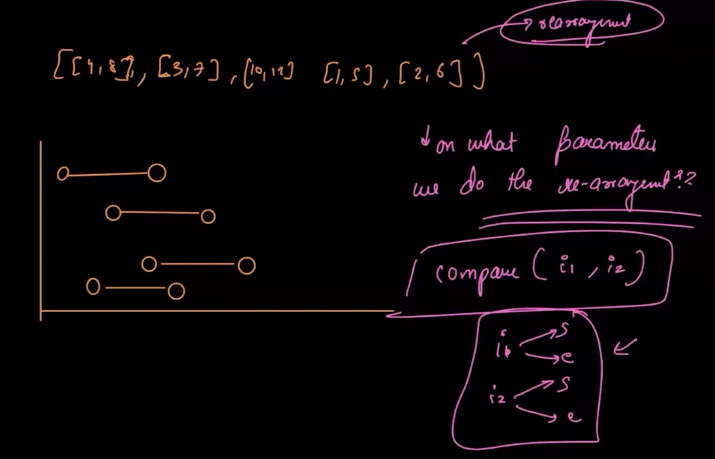
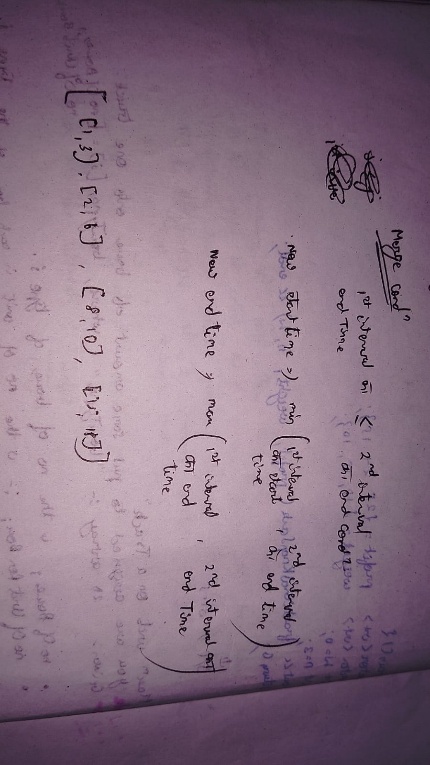
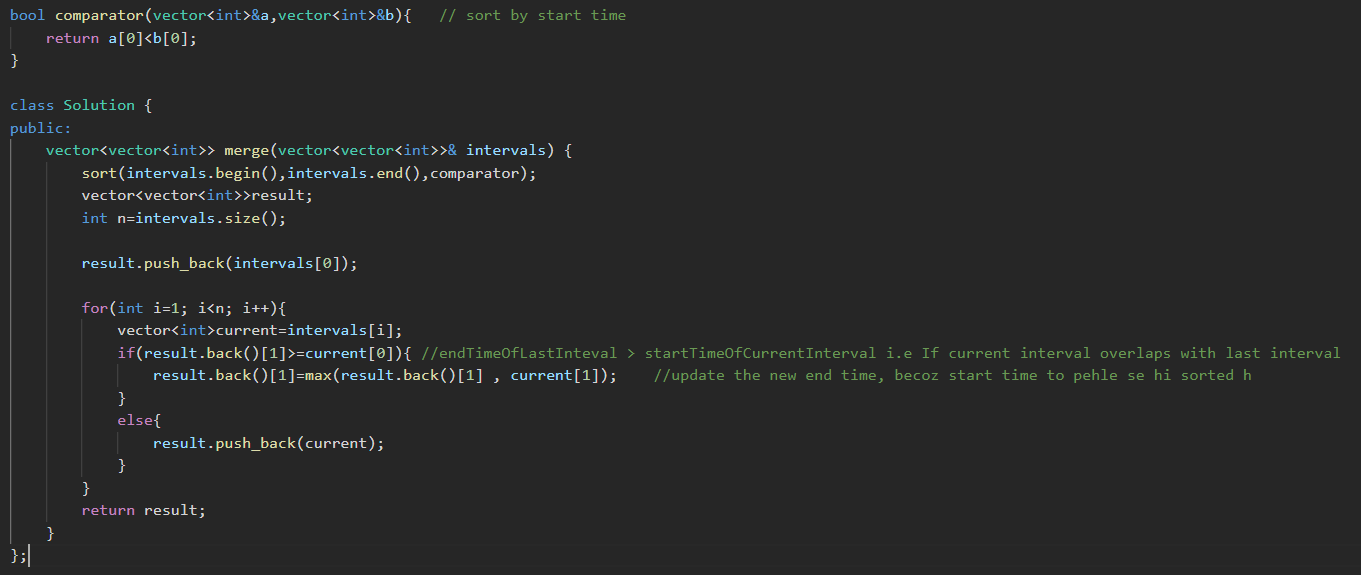


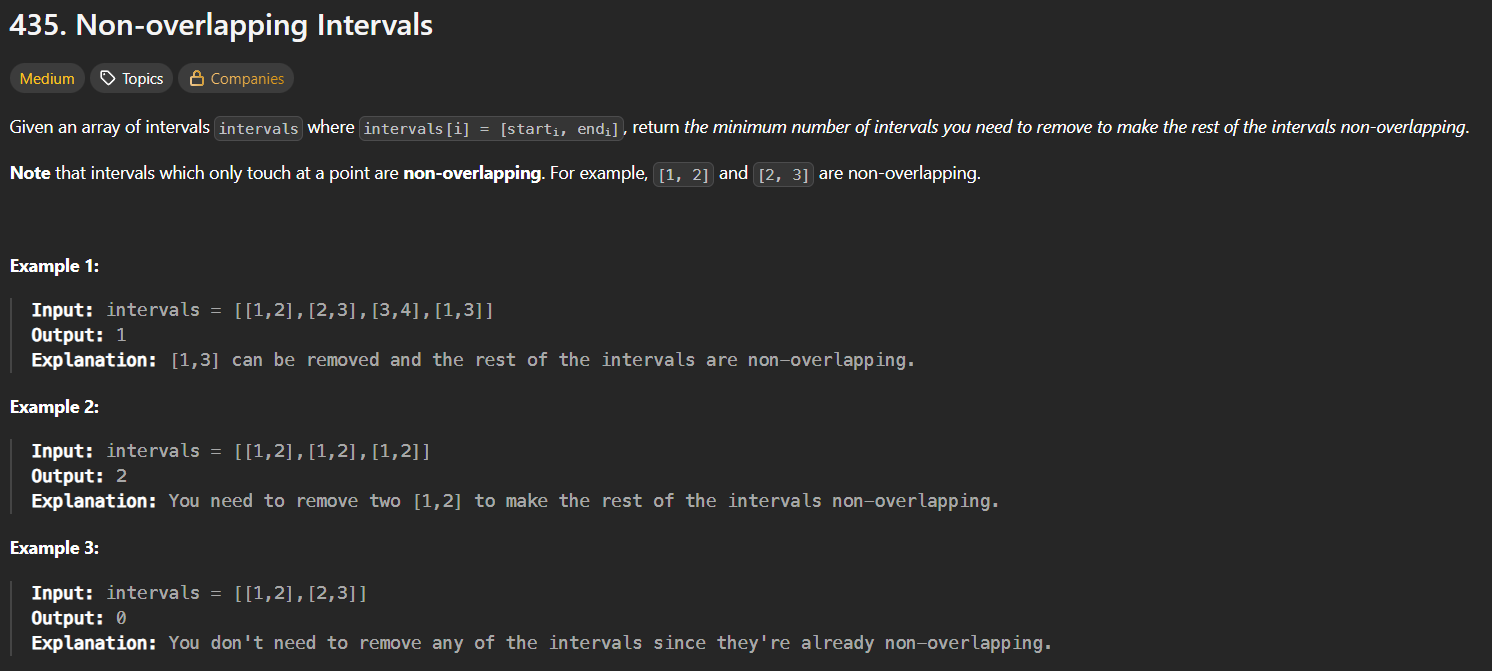
🡪premium question🡪code is same as above question just do the change in return wala line at the end.

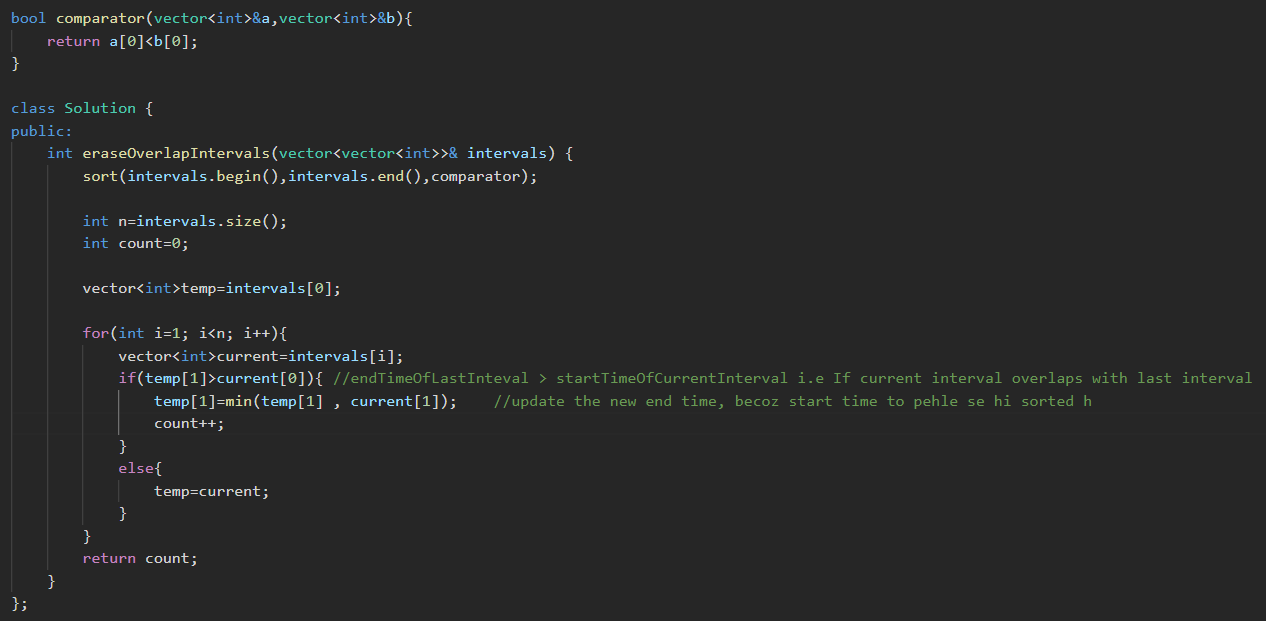
* Algorithm 🡪agar saara meeting ek hi room m ho then only person can attend all the meetings , so bs last m check krlo that whether answer==1 or not, if true then person can attend all the meetings.

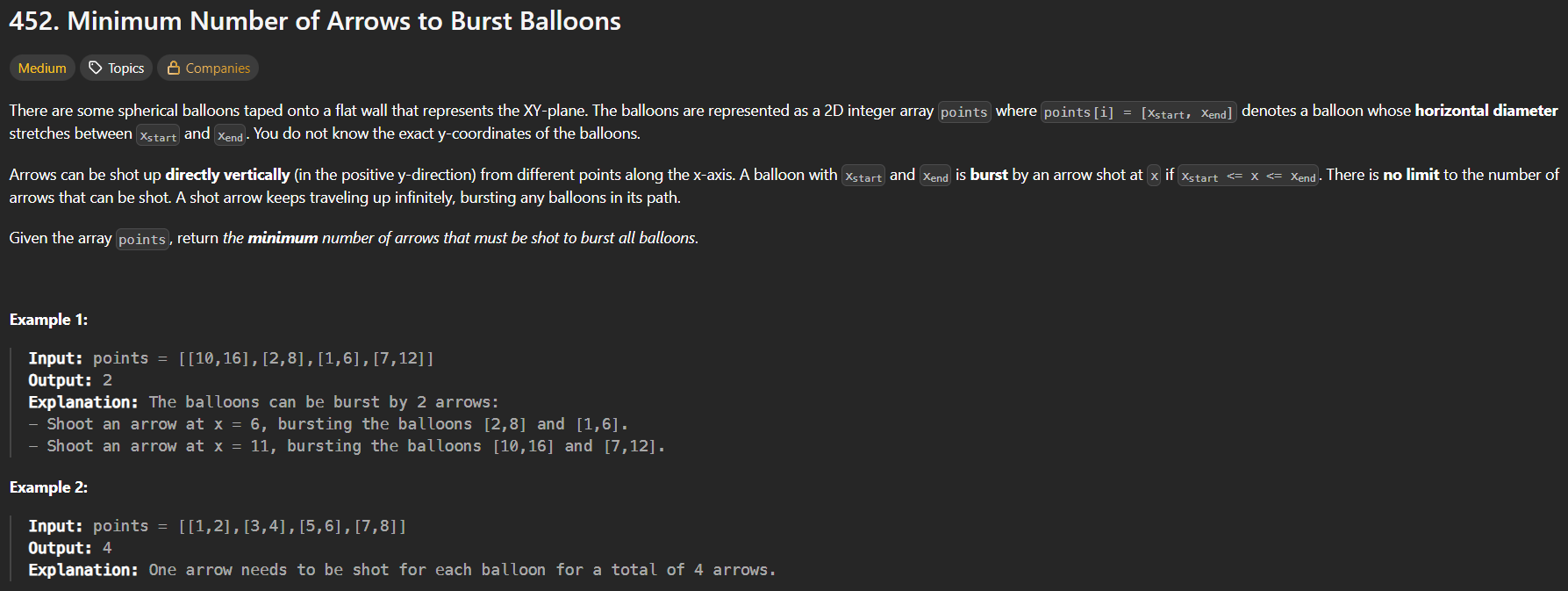
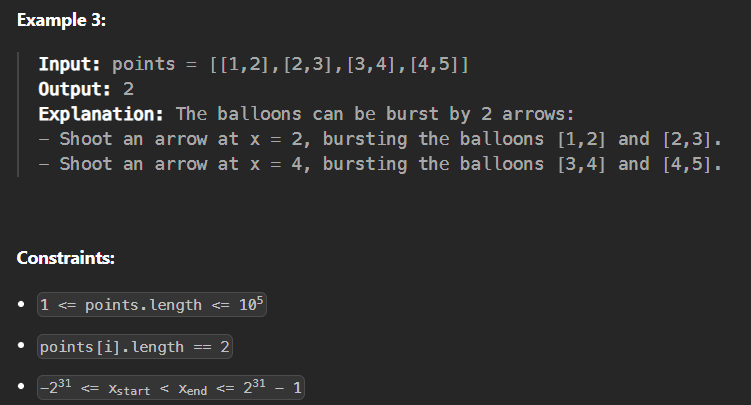




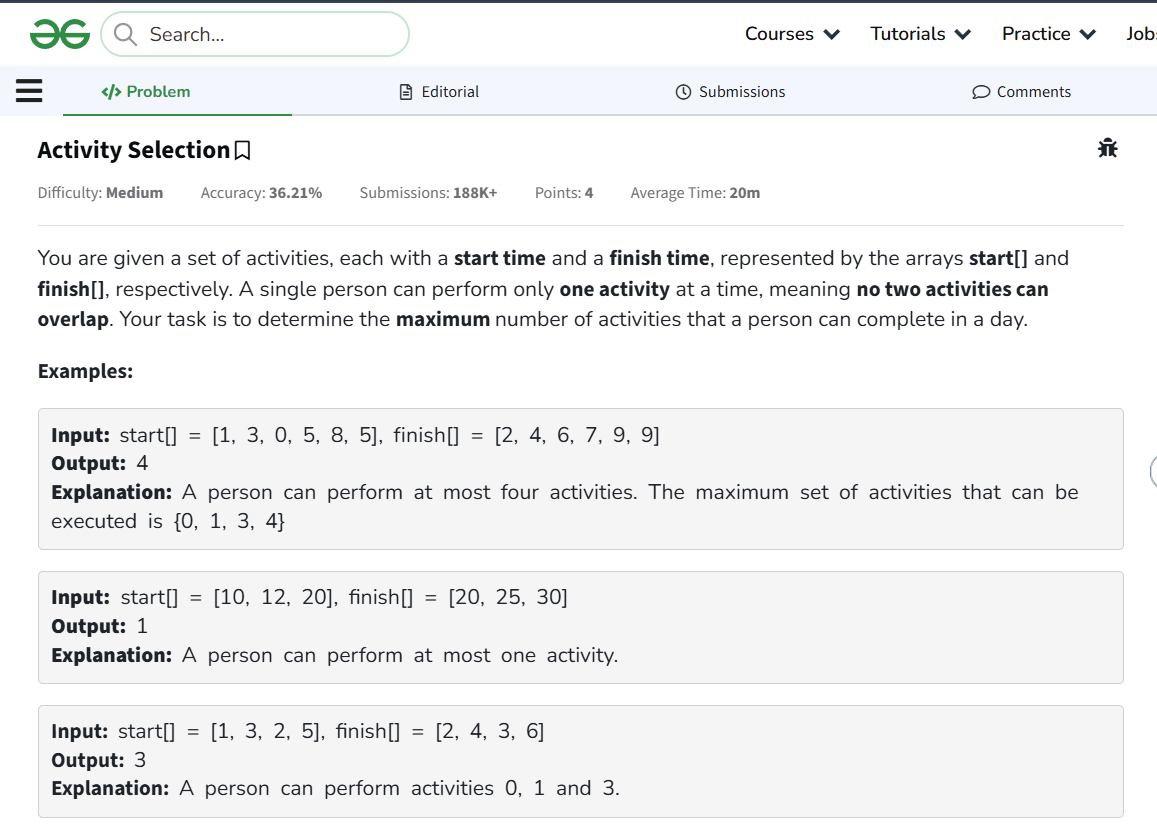
* 
* Algorithm :🡪we can sort the intervals on the basis of
  + Start time
  + Or, end time
* I’’ll solve by sorting on the basis of start time🡪then 1st element ko result vector m daal do, becoz, agar sirf ek hi element hua question m , then atleast who return ho jaayega. Then check
  + If (endTimeOfLastInteval >= startTimeOfCurrentInterval) 🡪 update the new end time, becoz start time to pehle se hi sorted h
  + And else if (endTimeOfLastInteval < startTimeOfCurrentInterval) 🡪then current waale ko directly answer m daal do
  + 
  + 

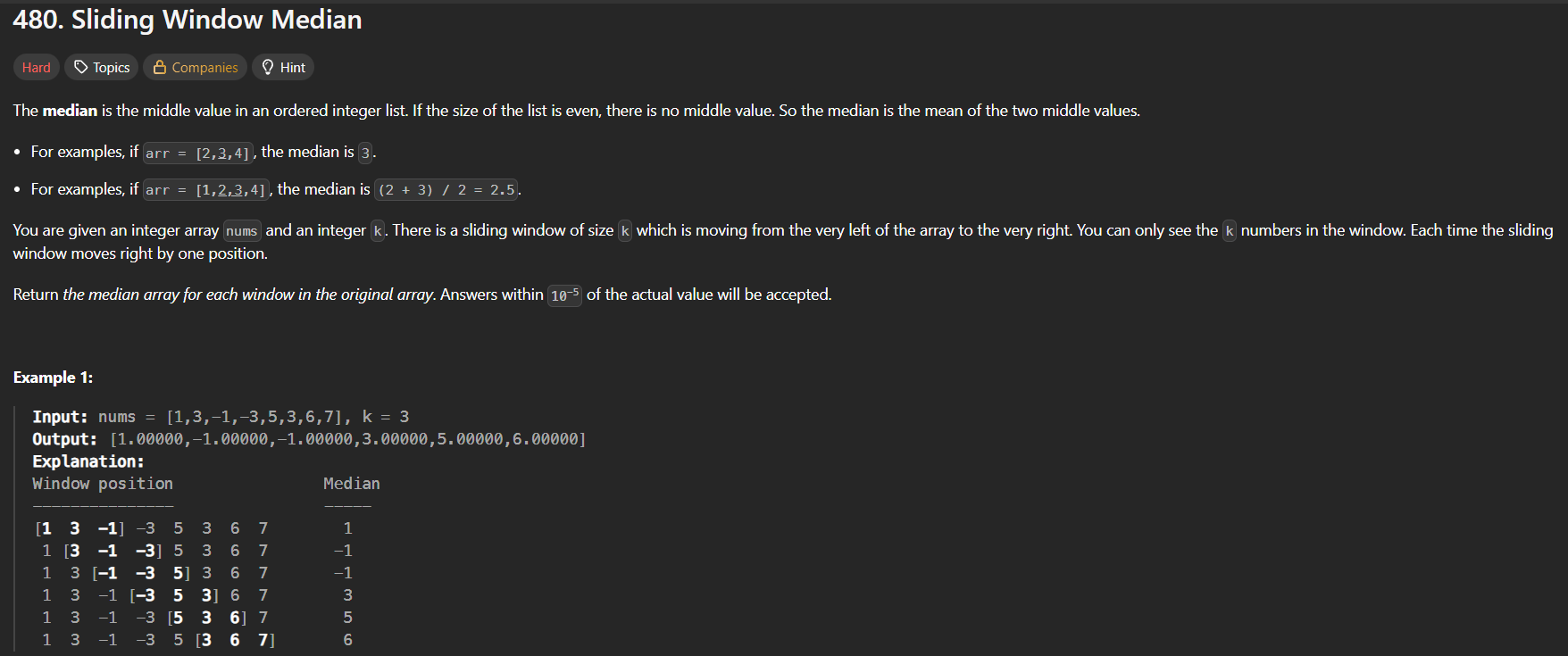
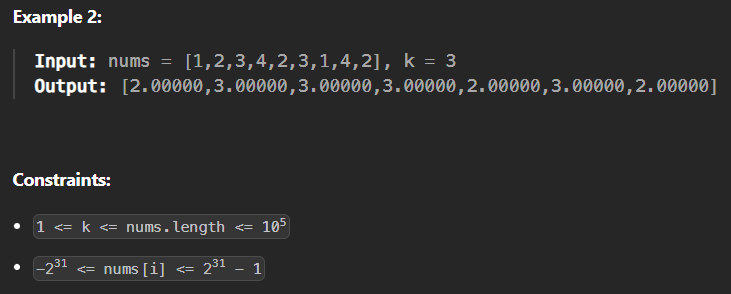
🡪same as above question, just maintain the count variable and return it and one major change is that, we will choose the **minimum of end times to minimize the no. of removals to make the interval non-overlapping**

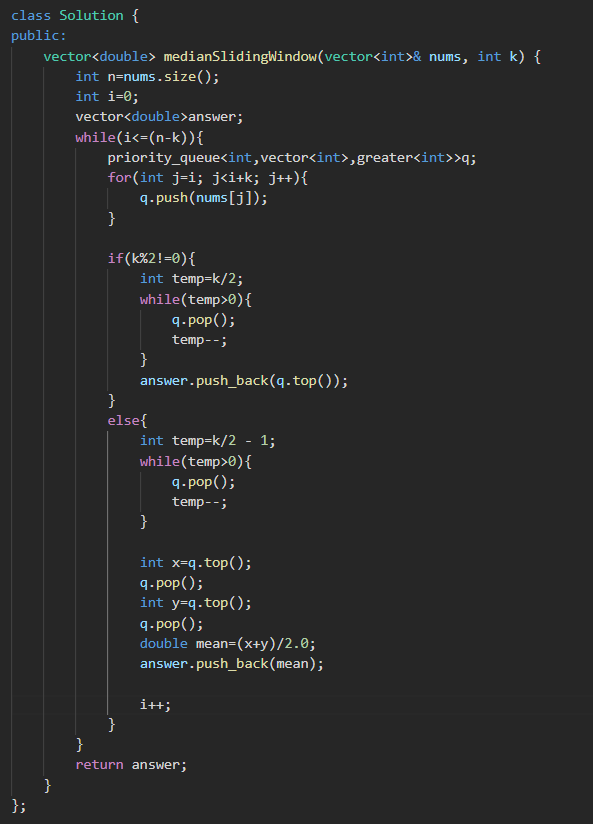
* ****

**** ****🡪same as above question

* Meaning of question🡪 balloon start and end kha ho rhi h, in the x-coordiate is given, and koii neeche se shoot krega balloons p, ar uske range m jitne bhi balloons aayenge wh for dega , so we need to find the minimum number of ballons.
* The question is exactly same as above question, hm pehle saare ballons jinka coordinates overlap kr rha h usko hm ek saath for denge.
* 

🡪question is similar to above question

* Method 1 🡪 using sorting 🡪 time limit exceeded 
* Method 2🡪 using single heaps 🡪 time limit exceeded 
* Method 3: 🡪using 2 heaps 🡪time limit exceeded

**🧠 Core Idea (Simple Words)**

To find the median **quickly** for every sliding window:

👉 We maintain **two heaps**:

**1️⃣ Max Heap (left side)**

* Stores the **smaller half** of numbers
* Top = **largest** element of smaller half

**2️⃣ Min Heap (right side)**

* Stores the **larger half** of numbers
* Top = **smallest** element of larger half

**🔑 Why Two Heaps?**

Because **median** depends on the **middle elements**:

**If k is odd:**

* Median = top of **max heap**

**If k is even:**

* Median = average of  
  maxHeap.top() and minHeap.top()

**📐 Heap Size Rule (VERY IMPORTANT)**

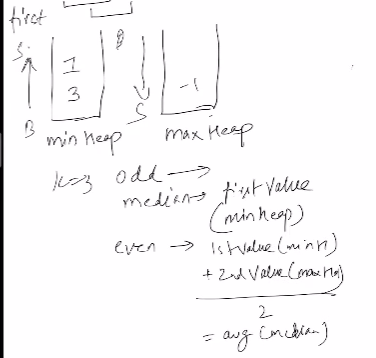
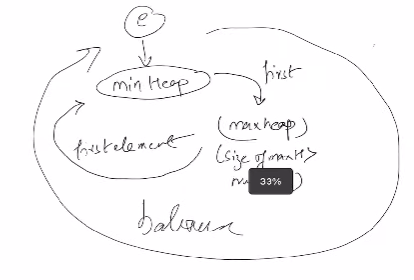
We always maintain:

maxHeap.size() == minHeap.size()

OR

maxHeap.size() == minHeap.size() + 1

So maxHeap is **never smaller** than minHeap.

* 

**Algorithm 🡪 make two heaps 🡪 1 maxheap and one minheap,**

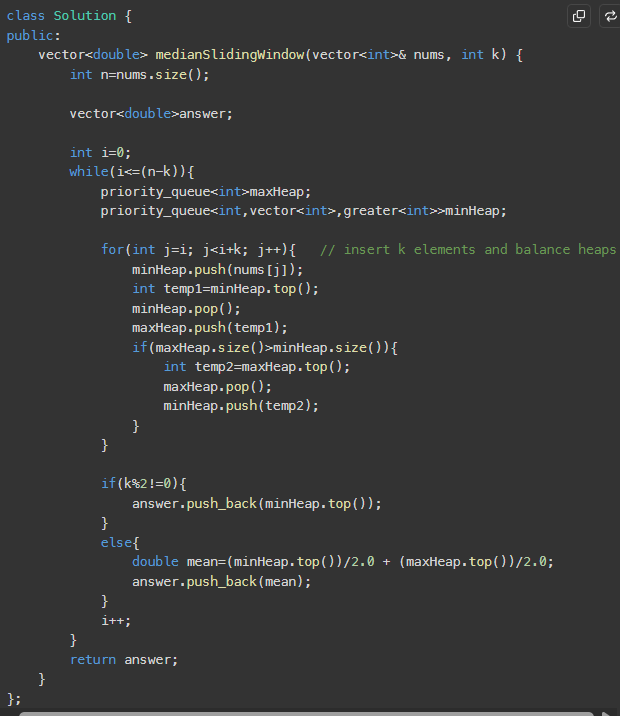
**🡪then iterate on the vector, and put the element in minheap 1st , then put it in the maxheap, and if the size of the maxheap gets larger than the minheap then again put the element in the minheap,**

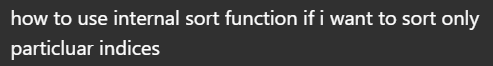
**🡪when the size of the window gets equal to k then,**

**🡺if k is odd 🡪 meadian = minheap.top().**

**🡺 if k is even 🡪 meadian = (minheap.top()+maxheap.top())/2;**

**🡺and lastly remove the 1st element of the window from the heap, whereever it is present and then do the same thing in the next window**

**🡪then push the above calculated median in the answer array.** ****

**Note:** 

* 