**Kth smallest number again**

Attempted by: **613**

/

Accuracy: **21%**

/

Maximum Points: **30**

/

15 Votes

/

**Share**

Binary search algorithm, Sorting, Sorting algorithm

**PROBLEM**

**EDITORIAL**

**MY SUBMISSIONS**

**ANALYTICS**

[**DISCUSSIONS**](https://www.hackerearth.com/practice/algorithms/searching/binary-search/practice-problems/algorithm/kth-smallest-number-again-2/discussion/)NEW

**Dexter** was good in finding the ***K*** th smallest number from a set of numbers. He thought he could solve any problem related to ***K*** th smallest number. His friend **Pipi** challenged him with a problem.  
He gave him various ranges of number, These numbers were arranged in increasing order(only distinct numbers to be taken into account). Now he asked him to find the ***K*** th smallest number in the sequence, again and again.

**Input Format**  
The first line contains ***T***, the number of test cases.  
For each test case, there will be two integers ***N*** and ***Q***.  
Then ***N*** lines follow each line containing two integers ***A*** and ***B*** (denoting the range ***A***-***B***)  
Then ***Q*** lines follow each line containing a non-negative integer ***K*** .

**Output Format**  
For each query output the ***K*** th smallest number.

**Constraints**  
1 <= ***T*** <= 100  
1 <= ***N*** <= 100  
1 <= ***Q*** <= 1000  
-10^18 <= ***A*** <= ***B*** <= 10^18  
***K*** >= 1

**N.B. If Kth smallest number is not present in the series, print -1**

**SAMPLE INPUT**

1

1 3

1 5

1

3

6

**SAMPLE OUTPUT**

1

3

-1

**Explanation**

The numbers are "1 2 3 4 5". The 1st smallest number is 1  
The 3rd smallest number is 3 The 6th smallest number is not present. Hence answer is -1

The hurdles one might face in the problem are:  
1. All numbers are given in form of ranges. 2. How to find the kth largest number in the range?

Ranges can be of two types:  
1. Overlapping and non-sorted.  
2. Non-overlapping and sorted.

Considering the simple case first  
**Non-overlapping and sorted**:

Let us consider the case of non-overlapping ranges first:  
Suppose the given ranges are:  
1-4, 8-10, 12-20, 40-50  
Now we have to find **K** th largest number in these ranges:  
Following a pattern. 1. if K>=1 and K<=4 the numbers will lie in the first range i.e. 1-4 .  
2. If K>=5 and K<=7 the numbers will lie in the second range i.e. 8-10.  
3. If K>=8 and K<=16 the numbers will lie in the third range i.e. 12-20.  
4. If K>=17 and K<=27 the numbers will lie in the fourth range i.e. 40-50.

Hence we can think of building a array where:  
arr[i]=Sum of the length of all range till i (where i starts from 1)  
(and arr[0]=0)

arr for this example will be:  
ranges: 1-4, 8-10, 12-20, 40-50  
length: 0 , 4 , 3 , 9 , 11 (Appending a zero length) cumulative: 0 , 4 , 7 , 16 , 27  
There are 27 numbers in total:

Now we have to find that **K** belongs to which interval:  
0-4 or 5-7 or 8-16 or 17-27  
We can easily find this using binary search:  
**Pseudo code:**

binary\_search(arr,value,size,)

{

low=0,high=sz,mid,calc;

while(low<=high)

{

mid=(low+high)/2;

//Value must be just greater than the interval's lower limit and lesser than or equal to interval's higher limit

if(value>arr[mid] && (mid==sz || value<=bin[mid+1]))

{

if(mid==sz)

return -1;

else

{

return mid;

}

}

//If value is greater than the current value then it lies in the part mid+1 to high

else if(value>bin[mid])

low=mid+1;

//If value is lesser than the current value then it lies in the part low to mid-1

else

high=mid-1;

}

return -1;

}

Now we know the range in which the value lies we just have add the lower limit of the range with the difference of value and present cumulative value:

answer=value-arr[mid]+range[mid]

**Overlapping and non-sorted:**  
If the ranges are overlapping and given in non-sorted order. First we need to sort the ranges based on lower limit and if the lower limits of ranges are same sort them according to higher limit.  
Now it becomes an ad-hoc problem to merge this overlapping-intervals such that we get a non-overlapping sorted interval.  
One can get the idea by following this example:  
Consider the ranges 2-4, 1-8, 1-6, 9-10  
Sorting this ranges: 1-6, 1-8, 2-4, 9-10

The steps we need to follow to do the same are:  
1. First push the range 1-6 in the new interval array:  
2. Check whether the new range lower\_limit is less than upper limit of the latest range in new interval array.  
If it's in the interval then update the upper value of the interval to max(higher\_limit of new array,higher\_limit of old array).  
If not then insert this interval in the array.

Implementing this approach in this interval:  
1. first push **1-6** in the new array. 2. **1-8** lower limit is in the range **1-6**. Hence update the maximum value of new array(i.e. 6) to **max(6,8)** i.e. 8. The range in new array is changed to **1-8**.  
3. **2-4** lower limit is in the range **1-8**. Hence update the maximum value of new array(i.e. 8) to **max(8,4)** i.e. 8. The range in new array remains same **1-8**.  
4. **9-10** lower limit is not in range **1-8**. Hence this range needs to be inserted in the array. The range is new array are: **1-8, 9-10**.