HW_Sort Characters By Frequency 3

Given a string **s**, sort it in decreasing order based on the **frequency** of the characters.If frequency of two characters are same sort them in ascending order. The **frequency** of a character is the number of times it appears in the string.

Return the **sorted** string. If there are multiple answers, return any of them.

Input Format

First line contains a string **s**.

Constraints

```
1 <= s.length <= 5 * 10^5
s consists of uppercase and lowercase English letters and digits.
```

Output Format

Returns a string as an output.

Sample Input 0

```
tree
```

Sample Output 0

```
eert
```

Explanation 0

'e' appears twice while 'r' and 't' both appear once. So 'e' must appear before both 'r' and 't'.

Sample Input 1

```
magjjkkwq
```

Sample Output 1

```
jjkkagmqw
```

Submitted Code

Language: Java 8

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
         String str= sc.nextLine();

         Map<Character , Integer> hm = new HashMap<>();

         for(int i=0;i<str.length();i++){
             char ch = str.charAt(i);
             hm.put(ch, hm.getOrDefault(ch,0)+1);
         }
         PriorityQueue<Character> pq = new PriorityQueue<>((a,b)->{
             int freqA = hm.get(a);
             int freqB = hm.get(b);
             if(freqA == freqB){
                 return a-b;
             }else{
                 return freqB-freqA;
             }
             });
         pq.addAll(hm.keySet());
         StringBuilder result = new StringBuilder();
         while(!pq.isEmpty()){
             char curr = pq.remove();
             int len = hm.get(curr);
             while(len-->0)result.append(curr);
         }
         System.out.print(result);
}
}
```

Submitted Code

Language: Java 8

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
         String str= sc.nextLine();

         Map<Character , Integer> hm = new HashMap<>();

         for(int i=0;i<str.length();i++){
             char ch = str.charAt(i);
             hm.put(ch, hm.getOrDefault(ch,0)+1);
         }
         PriorityQueue<Character> pq = new PriorityQueue<>((a,b)->{
             int frq = hm.get(b)-hm.get(a);
             return (frq!=0) ? frq : a-b;
             });
         pq.addAll(hm.keySet());
         StringBuilder result = new StringBuilder();
         while(!pq.isEmpty()){
             char curr = pq.remove();
             int len = hm.get(curr);
             while(len-->0)result.append(curr);
         }
         System.out.print(result);
}
}
```

HW_Furthest Building You Can Reach

You are given an integer array heights representing the heights of buildings, some bricks, and some ladders.

You start your journey from building 0 and move to the next building by possibly using bricks or ladders.

While moving from building i to building i+1 (0-indexed),

- If the current building's height is greater than or equal to the next building's height, you do not need a ladder or bricks.
- If the current building's height is less than the next building's height, you can either use one ladder or (h[i+1] - h[i]) bricks.

Print the furthest building index (0-indexed) you can reach if you use the given ladders and bricks optimally.

Question Link: https://leetcode.com/problems/furthest-building-you-can-reach/

Input Format

First line contains an integer input n

Next line contains an array heights[] of size n.

Next line contains an integer input which denotes number of bricks

Next line contains an integer input which denotes the number of ladder.

Constraints

- 1 <= heights.length <= 10^5
- 1 <= heights[i] <= 10^6
- 0 <= bricks <= 10^9
- 0 <= ladders <= heights.length

Output Format

Print the furthest building index (0-indexed) you can reach if you use the given ladders and bricks optimally.

Sample Input 0

```
7
4 2 7 6 9 14 12
5
1
```

Sample Output 0

```
4
```

Explanation 0

Starting at building 0, you can follow these steps:

- Go to building 1 without using ladders nor bricks since 4 >= 2.
- Go to building 2 using 5 bricks. You must use either bricks or ladders because 2 < 7.
- Go to building 3 without using ladders nor bricks since 7 >= 6.
- Go to building 4 using your only ladder. You must use either bricks or ladders because 6 < 9.

It is impossible to go beyond building 4 because you do not have any more bricks or ladders.

Language: Java 8

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] arr = new int[n];
        for(int i=0;i<n;i++)arr[i]=sc.nextInt();

        int bricks = sc.nextInt();
        int ladders = sc.nextInt();

        PriorityQueue<Integer> pq  = new PriorityQueue<>();

        for(int i=1;i<n;i++){
            int diff = arr[i] - arr[i-1];
            if(diff<= 0)continue;

            pq.add(diff);
            if(pq.size()>ladders)bricks-=pq.poll();
            if(bricks<0){
                System.out.print(i-1);
                return;
            }

        }
        System.out.print(arr.length-1);
    }
}
```