3RD JAN 2023

FIRST :-

Minimize number of Students to be removed

N Students of different heights are attending an assembly. The heights of the students are represented by an array **H[].** The problem is that if a student has less or equal height than the student standing in front of him, then he/she cannot see the assembly. Find the minimum number of students to be removed such that maximum possible number of students can see the assembly.

Example 1:

```
Input:
N = 6
H[] = {9, 1, 2, 3, 1, 5}
Output:
2
Explanation:
We can remove the students at 0 and 4th index.
which will leave the students with heights
1,2,3, and 5.
```

Example 2:

```
Input:
N = 3
H[] = {1, 2, 3}
Output :
0
Explanation:
All of the students are able to see the
assembly without removing anyone.
```

Your Task:

You don't need to read input or print anything. Your task is to complete the function **removeStudents()** which takes an integer N and an array H[ ] of size N as input parameters and returns the minimum number of students required to be removed to enable maximum number of students to see the assembly.

Expected Time Complexity: O(N logN)
Expected Auxiliary Space: O(N)

Constraints:
$1 \le N \le 10^5$
$1 \le H[i] \le 10^5$

CODE SECTION:-

```
class Solution
{
public:
    int removeStudents(int H[], int N)
    {
        vector<int> v;
        v.push_back(H[0]);

        for (int i = 1; i < N; i++)
        {

            if (H[i] > v.back())
            {
                v.push_back(H[i]);
            }

            else
            {
                int index = lower_bound(v.begin(), v.end(), H[i]) - v.begin();
                v[index] = H[i];
            }
        }
        return N - v.size();
    }
```

Help section: -

1. Creted a vector v in which we will push the first element from the given vector so that can compare in the next call

2. Use a for loop from the first index element to the last element index

   a. compare the element if element of vector is greater then v.back(){ last element in the vector v}

   b. else find the index where should be inserted in vector using { lower_bound(v.begin(),v.end(),H[i])-v.begin() } for that element coz,we have to insert that element at that index.

3. After this loop we will get the largest increasing sub-sequence

4. Now just return the (size of original array-size of new array v)

SECOND:-

Reverse a Stack : : Medium

You are given a stack **St**. You have to reverse the stack using recursion.

Example 1:

```
Input:
St = {3,2,1,7,6}
Output:
{6,7,1,2,3}
```

Example 2:

```
Input:
St = {4,3,9,6}
Output:
{6,9,3,4}
```

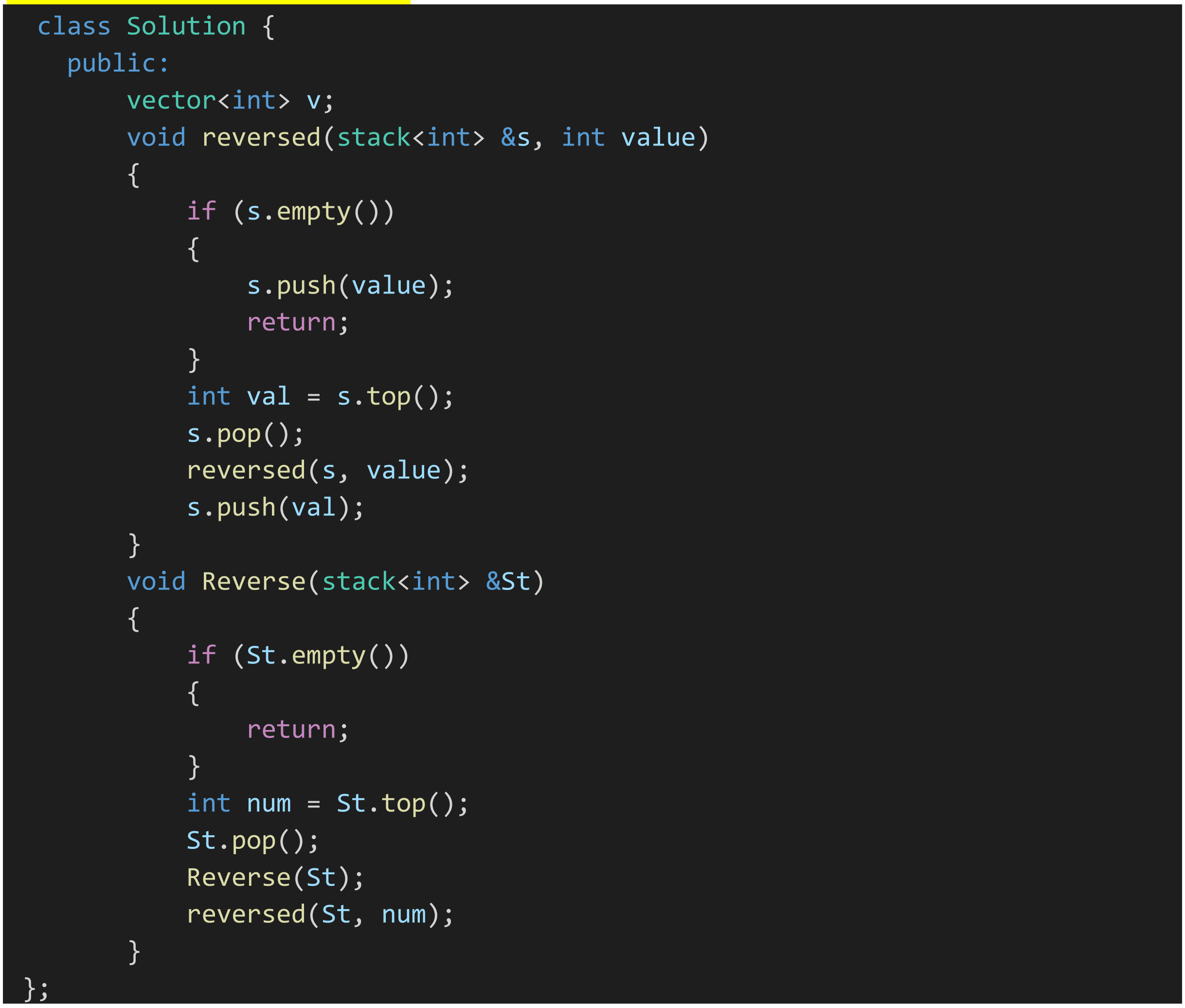
Your Task:

You don't need to read input or print anything. Your task is to complete the function **Reverse()** which takes the stack **St** as input and returns the reversed stack.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(N)

Constraints:
1 <= size of the stack <= 10^4
-10^9 <= Each element of the stack <= 10^9
Sum of N over all test cases doesn't exceeds 10^6
Array may contain duplicate elements.

CODE SECTION:-



```
 class Solution {
  public:
    vector<int> v;
    void reversed(stack<int> &s, int value)
    {
        if (s.empty())
        {
            s.push(value);
            return;
        }
        int val = s.top();
        s.pop();
        reversed(s, value);
        s.push(val);
    }
    void Reverse(stack<int> &St)
    {
        if (St.empty())
        {
            return;
        }
        int num = St.top();
        St.pop();
        Reverse(St);
        reversed(St, num);
    }
};
```

Help section :-

1. Create another function which will insert the value of st.top() during returning time….
2. Save the value of the top in another variable and pop the stack and now call our function recursively…
3. Our new function will take two parameter one is stack and another is value which will be inserted….
4. “ kuch nhi hoga is help section se. Code revise karo .-. ”

-: Done for the today :-