Problem Solving On Stacks

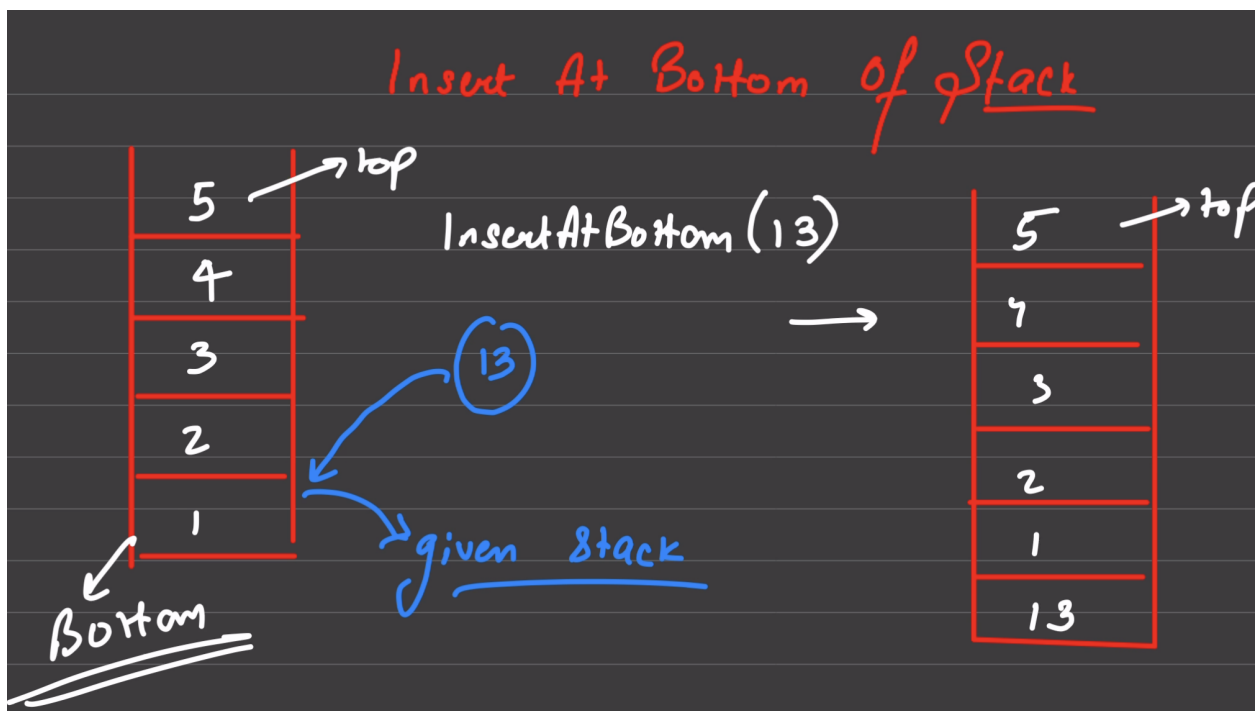|	
⊙ Created	@August 13, 2022 12:14 PM
⊙ Class	
⊙ Type	
⊘ Materials	
☑ Reviewed	☐

Problem 1:

Given a stack and an element X, insert the element at the bottom of the stack.

Example:

St → 1,2,3,4,5 where 5 is the top element and X = 13

ans → st → 13,1,2,3,4,5

We want to insert the element in stack at the bottom, but stack only gives us access to the top.

If we want to insert something at the bottom, we need to technically, remove the element from the stack and store them somewhere such that stack becomes empty. Once it is empty any element we insert will be at the bottom so now we can insert X. And now we can bring back the original elements in the stack in the same order. So to store the elements we can use another stack only, why ? because then the current topmost element which is 5 will be added first in the new temp stack, and when we bring back the elements 5 will be the last to be brought in the original stack and hence it stays on top.

This approach works because when we transfer elements from one stack to another they're added in reverse order in the new stack.

Problem 2: Next Greater Element

Given an array of integers of length N, for every element in the array find the next greater element to the right. Next greater element to the right means the first element to the right which is greater than the current element.

ex:

[3, 6, 1, 7, 9, 13, 12, 16, 9]

For 3 the next greater element is 6

For 6 the next greater element is 7

For 1 the next greater element is 7

For 7 the next greater element is 9

For 9 the next greater element is 13

For 13 the next greater element is 16

For 12 the next greater element is 16

For 16 the next greater element is -1, as there is no element to the right of 16 which is greater than 16

For 9 the next greater element is -1, as there is no element to the right of 9 which is greater than 9
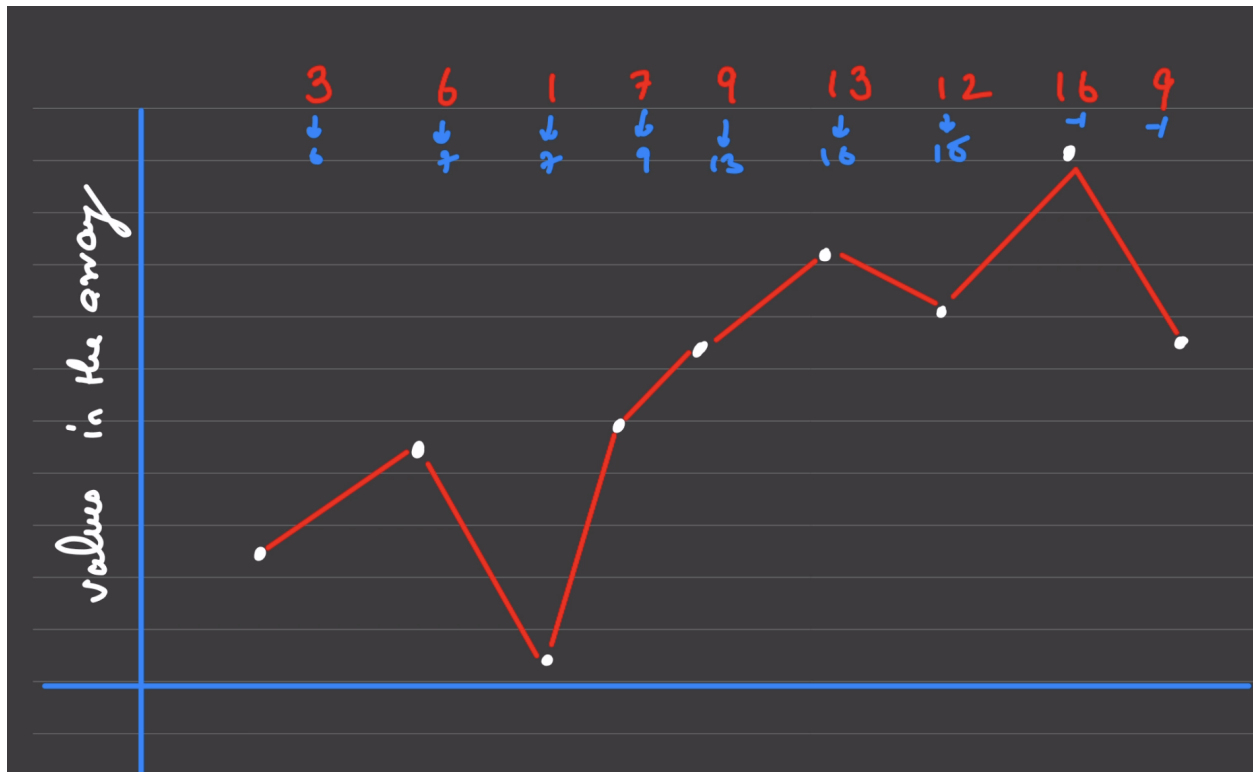
ans →[6,7,7,9,13,16,16,-1,-1]

Constraints: N ≤ 10^6

Solution:

In bruteforce solution for every element we can iterate on the right and find the first greater element. O(n^2)

How to optimise ?

The above curve represents the plot of the values of the given array.

If we carefully see the plot and only focus on the parts of the curve which is increasing in value, then we can say all the values on an increasing curve have their next greater element as the immediate right element i.e. if arr[i] is on an increasing curve then the next greater element for `arr[i] is arr[i+1]`

What about the elements on decreasing curve, whenever we have element on decreasing curve, we are going to put them in a pending state. And the moment we shift to an increasing curve by getting a new greater element we will try to compare this new

elements with all the prev pending elements. And the comparison starts with the immediate left element, and then with the prev pending elements and so on. So we need to keep a track of the previous pending elements and maintain their order same as in the array. So here we can use stacks as they help us to maintain previous states.
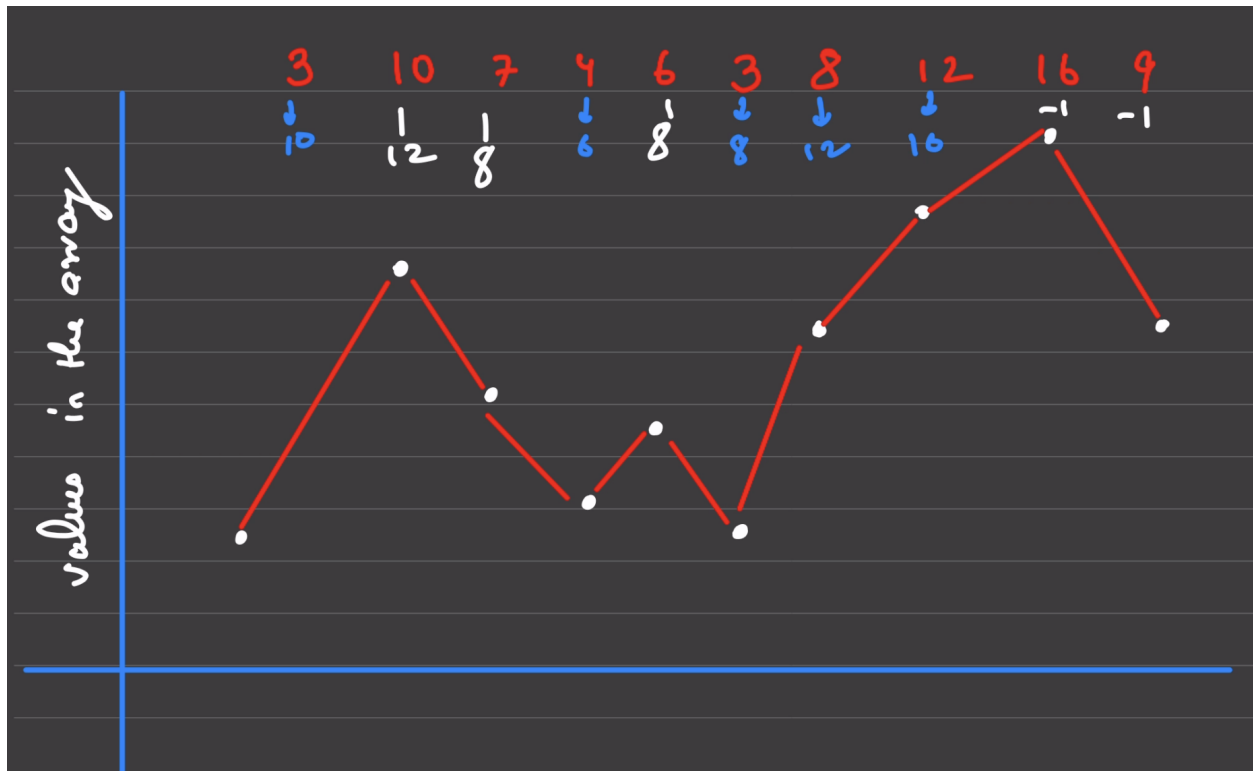
In the above example, starting from 10 we have got a decreasing curve, then 7 also lies in the decreasing curve and 4 as well. The moment we hit 6, then we shift to an increasing curve, and as discusses whenever we take a shift, we need to resolve the prev elements. And the shift happens to an increasing curve so definitely we will get ans for at least one pending element. Then one by one with the current element we will try to resolve the pending elements and 6 can only resolve 4, so we got ans for 4 but now 6 is in a pending state. then 3 also in pending state as 3 lies on decreasing curve. The moment we hit 8, we again take a shift to the increasing curve, hence we will try to resolve as many pending elements as possible one by one. So, we compare 8 with 3, 6 and then 7. For all these 3 the next greater is 8 as 8 is bigger than these 3. But 8 canot resolve 10, so now 8 is also in a pending state.

```
function nextGreater(arr) {
    /**
```

```
 * Time: O(n)
 * Space: O(n)
 */
let st = new Stack(); // stack of indexes
let output = Array(arr.length);
st.push(0);
for(let i = 1; i < arr.length; i++) {
    while(!st.isEmpty() && arr[i] > arr[st.top()]) {
        // we took a shift, we need to resolve
        output[st.top()] = arr[i];
        st.pop();
    }
    st.push(i);
}
while(!st.isEmpty()) {
    output[st.top()] = -1;
    st.pop();
}
return output
}
```