Next Greater Element Using Stack

Problem Statement: Given a circular integer array **A**, return the next greater element for every element in A. The next greater element for an element x is the first element greater than x that we come across while traversing the array in a clockwise manner. If it doesn’t exist, return -1 for this element.

Examples:

Example 1:

Input: N = 11, A[] = {3,10,4,2,1,2,6,1,7,2,9}

Output: 10,-1,6,6,2,6,7,7,9,9,10

Explanation: For the first element in A ,i.e, 3, the greater element which comes next to it while traversing and is closest to it is 10. Hence,10 is present on index 0 in the resultant array. Now for the second element,i.e, 10, there is no greater number and hence -1 is it’s next greater element (NGE). Similarly, we got the NGEs for all other elements present in A.

Example 2:

Input: N = 6, A[] = {5,7,1,7,6,0}

Output: 7,-1,7,-1,7,5

Solution

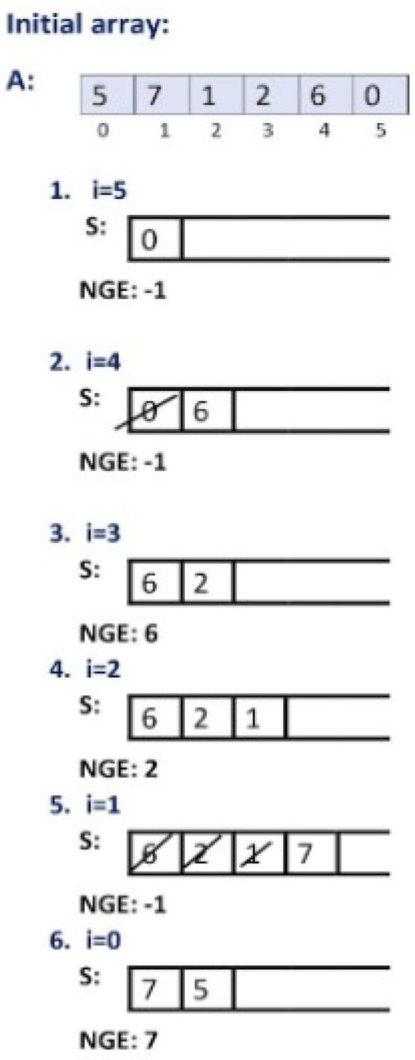
Disclaimer: *Don’t jump directly to the solution, try it out yourself first.*

Approach:

This problem can be solved easily and efficiently by using the stack data structure as it is based on the Last in First out (LIFO) principle.

To make it a bit easier let’s first try to solve without considering the array as circular. To find the next greater element we start traversing the given array from the right. As for the rightmost element, there is no other element at its right. Hence, we assign -1 at its index in the resultant array. Since this can be the next greater element (NGE) for some other element, we push it in the stack S. We keep checking for other elements. Let’s say we are checking for an element at index i. We keep popping from the stack until the element at the top of the stack is smaller than A[i]. The main intuition behind popping them is that these elements can never be the NGE for any element present at the left of A[i] because A[i] is greater than these elements. Now, if the top element of S is greater than A[i] then this is NGE of A[i] and we will assign it to res[i], where res is the resultant array. If the stack becomes empty then it implies that no element at the right of A[i] is greater than it and we assign -1. At last, we push A[i] in S.

Dry run: Let’s apply this algorithm for A[] = {5,7,1,2,6,0}:



So, the resultant array is {7,-1,2,6,-1,-1}. Remember that we have considered the array to be non-circular. For a circular array, the resultant array should be {7,-1,2,6,7,5}.
Now we need to make this algorithm work for a circular array. The only difference between a circular and non-circular array is that while searching for the next greater element in a non-circular array we don't consider the elements left to the concerned element. This can be easily done by inserting the elements of the array A at the end of A, thus making its size double. But we actually don't require any extra space. We can just traverse the array twice. We actually run a loop 2*N times, where N is the size of the given array.
Code:

- C++ Code
- Java Code
- Python Code

```
#include<bits/stdc++.h>

using namespace std;
class Solution {
  public:
    vector < int > nextGreaterElements(vector < int > & nums) {
      int n = nums.size();
      vector < int > nge(n, -1);
      stack < int > st;
      for (int i = 2 * n - 1; i >= 0; i--) {
        while (!st.empty() && st.top() <= nums[i % n]) {
          st.pop();
        }

        if (i < n) {
          if (!st.empty()) nge[i] = st.top();
```

```
            }
            st.push(nums[i % n]);
         }
         return nge;
      }
};
int main() {
   Solution obj;
   vector < int > v {5,7,1,2,6,0};
   vector < int > res = obj.nextGreaterElements(v);
   cout << "The next greater elements are" << endl;
   for (int i = 0; i < res.size(); i++) {
      cout << res[i] << " ";
   }
}
```

Output:

```
The next greater elements are
7 -1 2 6 7 5
```

Time Complexity: O(N)

Space Complexity: O(N)