Nearest Greater To Left(NGL)

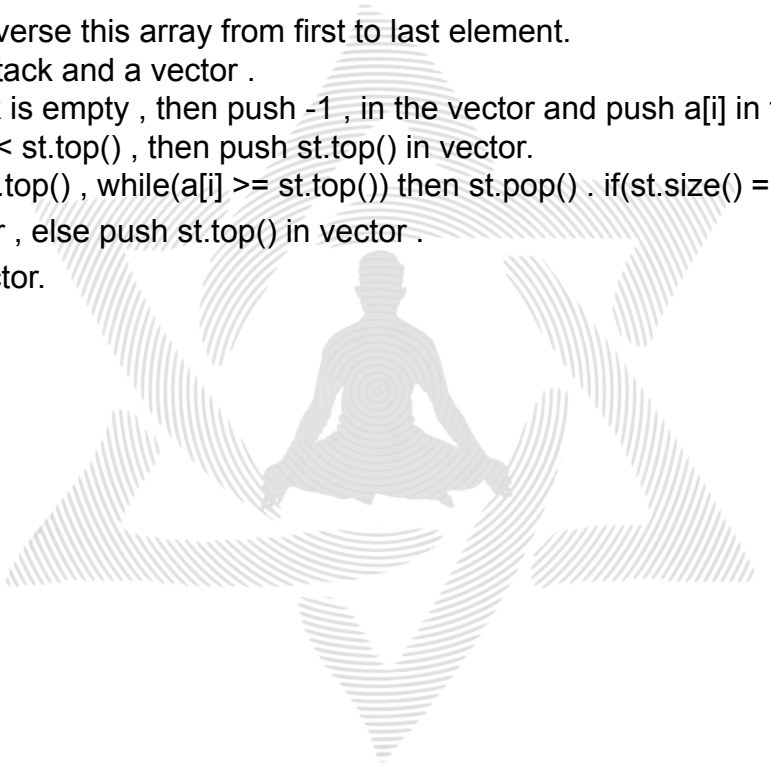
For every integer in an array , you have to find the nearest largest element to the left of the array .
Let's take an example ,

1 3 0 0 1 2 4 → -1 -1 3 3 3 3 -1

In this array , for 2 , nearest largest for 2 is 3 , for 1 is 3 , for 0 is 3 and so on ,
If in any case , you come across any element for which there is no greater element on its left , then print -1.

ALGORITHM:

1. We will traverse this array from first to last element.
2. Create a stack and a vector .
3. If the stack is empty , then push -1 , in the vector and push a[i] in the stack.
4. Else if a[i] < st.top() , then push st.top() in vector.
5. If a[i] >= st.top() , while(a[i] >= st.top()) then st.pop() . if(st.size() == 0) , push -1 in vector , else push st.top() in vector .
6. Return vector.



CODE :

```
class
{
    public:
  vector<long long> nextLargerElement(vector<long long> arr, int n){
    vector<long long> v; // creating a vector for storing result
    stack <long long> s; // creating a stack for temp. hold the values from array
    for (int i=0;i<n;i++){
      if(s.size() ==0) // when stack size is empty there is no element in stack return output as -1;
        v.push_back(-1);
      else if (s.size()>0 && s.top()>arr[i]) // when there is element in stack and stack top is greater then array element
      {
        v.push_back(s.top()); // take stack top in the result vector
      }
      else if (s.size()>0 && s.top()<=arr[i]) // when there is element in stack and that element is less then array element
      {
        while(s.size()>0 && s.top()<=arr[i]) // upto when there is element and stack top is less then array's element delete the element from stack
        {
          s.pop(); // delete the element from stack
        }
        if (s.size()==0) // when stack became empty return -1
        {
          v.push_back(-1);
        }
        else
        {
          v.push_back(s.top()); // else push stack top in the vector
        }
      }
      s.push(arr[i]); // push array in stack
    }
    return v;
  }
};


// Time Complexity: O(N)
// Auxiliary Space: O(N)
```