DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Tamanna Gupta **UID: 22BCS14867**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

➢ DAY-4 [23-12-2024]

1. Minimum Elements in Constant Time *(Very Easy)*

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time. Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Example 1:
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
Output
[null,null,null,null,-3,null,0,-2]
Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();

minStack.top();   // return 0
minStack.getMin(); // return -2

Example 2:

Input:

["MinStack", "push", "push", "push", "push", "getMin", "pop", "getMin", "top", "getMin"]

[[], [5], [3], [7], [3], [], [], [], [], []]

Output

[null, null, null, null, null, 3, null, 3, 7, 3]

Explanation:

MinStack minStack = new MinStack();
minStack.push(5);   # Stack: [5], MinStack: [5]
minStack.push(3);   # Stack: [5, 3], MinStack: [5, 3]
minStack.push(7);   # Stack: [5, 3, 7], MinStack: [5, 3]
minStack.push(3);   # Stack: [5, 3, 7, 3], MinStack: [5, 3, 3]
minStack.getMin();   # Returns 3
minStack.pop();   # Removes 3; Stack: [5, 3, 7], MinStack: [5, 3]
minStack.getMin();   # Returns 3
minStack.top();   # Returns 7
minStack.getMin();   # Returns 3

- Minimum values are maintained as: [5] → [5, 3] → [5, 3] → [5, 3]
- After pops, the minimum values update accordingly.

Example 3:

Input:

["MinStack", "push", "push", "push", "getMin", "pop", "getMin", "pop", "getMin"]

[[], [2], [1], [4], [], [], [], [], []]

Output:

[null, null, null, null, 1, null, 1, null, 2]

Explanation:

minStack = MinStack()
minStack.push(2)
minStack.push(1)
minStack.push(4)
minStack.push(1)
print(minStack.getMin()) # Output: 1
minStack.pop()

```
print(minStack.getMin())  # Output: 1
minStack.pop()
print(minStack.getMin())  # Output: 1
minStack.pop()
print(minStack.getMin())  # Output: 2
```

- Minimum values are maintained as: [2] → [2, 1] → [2, 1] → [2, 1]
- After pops, the minimum values update accordingly.

Constraints:

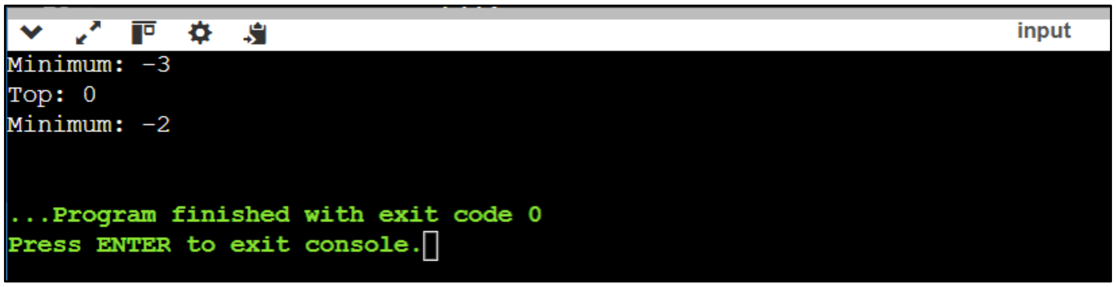
- -2^31 <= val <= 2^31 - 1
- Methods pop, top and getMin operations will always be called on non-empty stacks.
- At most 3 * 10^4 calls will be made to push, pop, top, and getMin.

Implementation/Code:

```
#include <iostream>                                          //Programming in C++
#include <stack>
#include <limits.h>

class MinStack {
private:
   std::stack<int> stack;
   std::stack<int> minStack;

public:
   MinStack() {}

   void push(int val) {
      stack.push(val);
      if (minStack.empty() || val <= minStack.top()) {
         minStack.push(val);
      }
   }

   void pop() {
      if (stack.top() == minStack.top()) {
         minStack.pop();
      }
```

```
        stack.pop();
    }

    int top() {
        return stack.top();
    }

    int getMin() {
        return minStack.top();
    }
};

int main()
{
    MinStack minStack;
    minStack.push(-2);
    minStack.push(0);
    minStack.push(-3);
    std::cout << "Minimum: " << minStack.getMin() << std::endl; // Output: -3
    minStack.pop();
    std::cout << "Top: " << minStack.top() << std::endl;        // Output: 0
    std::cout << "Minimum: " << minStack.getMin() << std::endl; // Output: -2
    return 0;
}
```

Output:



```
Minimum: -3
Top: 0
Minimum: -2


...Program finished with exit code 0
Press ENTER to exit console.
```

2. Number of Students Unable to Eat Lunch *(Easy)*

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step:

If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue.
Otherwise, they will leave it and go to the queue's end.
This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the ith sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the jth student in the initial queue (j = 0 is the front of the queue). Return the number of students that are unable to eat.

Example 1:

Input: students = [1,1,0,0], sandwiches = [0,1,0,1]
Output: 0
Explanation:
- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1].
- Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0].
- Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,1].
- Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1].

- Front student takes the top sandwich and leaves the line making students = [] and sandwiches = [].
Hence all students are able to eat.

Example 2:
Input: students = [1,1,1,0,0,1], sandwiches = [1,0,0,0,1,1]
Output: 3

Constraints:
- 1 <= students.length, sandwiches.length <= 100
- students.length == sandwiches.length
- sandwiches[i] is 0 or 1.
- students[i] is 0 or 1.

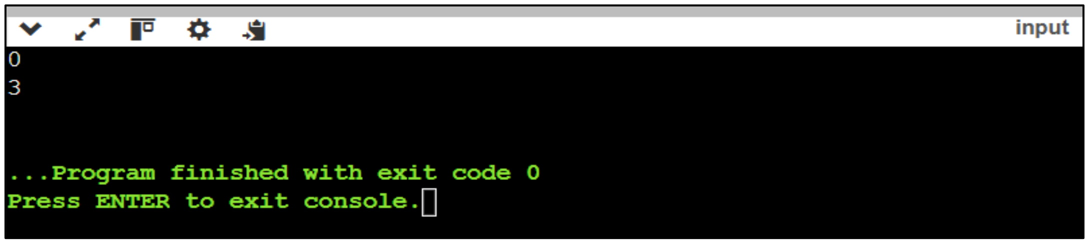
Approach
- Create two queues of students and sandwiches
- And a count variable to check if is loop in left student
- If students in the queue cannot have their ordered sandwiches, it makes a loop. If it is a loop, just break and return the result
- Then implement the program like the given rules.

Time complexity: O(n)
Space complexity: O(n)

Implementation/Code:

```
#include <iostream>                                        //Programming in C++
#include <queue>
#include <vector>

int countStudents(std::vector<int>& students, std::vector<int>& sandwiches) {
    std::queue<int> studentQueue;
    for (int student : students) {
        studentQueue.push(student);
    }

    int i = 0, loopCount = 0;
    while (!studentQueue.empty()) {
        if (studentQueue.front() == sandwiches[i]) {
            studentQueue.pop();
            i++;
            loopCount = 0;
        } else {
            studentQueue.push(studentQueue.front());
            studentQueue.pop();
            loopCount++;
        }

        if (loopCount == studentQueue.size()) {
            break;
        }
    }

    return studentQueue.size();
}

int main() {
    std::vector<int> students = {1, 1, 0, 0};
    std::vector<int> sandwiches = {0, 1, 0, 1};
    std::cout << countStudents(students, sandwiches) << std::endl; // Output: 0
    std::vector<int> students2 = {1, 1, 1, 0, 0, 1};
    std::vector<int> sandwiches2 = {1, 0, 0, 0, 1, 1};
    std::cout << countStudents(students2, sandwiches2) << std::endl; // Output: 3
    return 0;
}
```

Output:



```
0
3


...Program finished with exit code 0
Press ENTER to exit console.
```

3. Next Greater Element II *(Medium)*

Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return the next greater number for every element in nums.

The next greater number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number.

<u>Example 1:</u>
Input: nums = [1,2,1]
Output: [2,-1,2]
Explanation:

- The first 1's next greater number is 2;
- The number 2 can't find next greater number.
- The second 1's next greater number needs to search circularly, which is also 2.

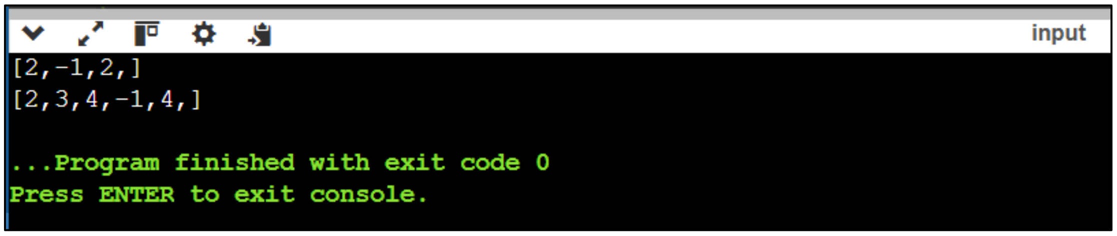
<u>Example 2:</u>
Input: nums = [1,2,3,4,3]
Output: [2,3,4,-1,4]
Constraints:

- 1 <= nums.length <= 104
- -109 <= nums[i] <= 109

Implementation/Code:

```
#include <iostream>                                        //Programming in C++
#include <vector>
#include <stack>

std::vector<int> nextGreaterElements(std::vector<int>& nums) {
    int n = nums.size();
    std::vector<int> result(n, -1);
    std::stack<int> s;

    for (int i = 0; i < 2 * n; i++) {
        while (!s.empty() && nums[s.top()] < nums[i % n]) {
            result[s.top()] = nums[i % n];
            s.pop();
        }
        if (i < n) {
            s.push(i);
        }
    }

    return result;
}

int main() {
    std::vector<int> nums = {1, 2, 1};
    std::vector<int> result = nextGreaterElements(nums);
    std::cout << "[";
    for (int val : result) {
        std::cout << val << ",";
    }
    std::cout << "]"<<std::endl; // Output: 2 -1 2
    std::vector<int> nums2 = {1, 2, 3, 4, 3};
    std::vector<int> result2 = nextGreaterElements(nums2);
    std::cout << "[";
    for (int val : result2) {
        std::cout << val << ",";
    }
    std::cout << "]"; // Output: 2 3 4 -1 4
    return 0;
}
```

Output:



```
[2,-1,2,]
[2,3,4,-1,4,]

...Program finished with exit code 0
Press ENTER to exit console.
```

4. Sliding Window Maximum *(Hard)*

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.
Return the max sliding window.

Example 1:
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:

```
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

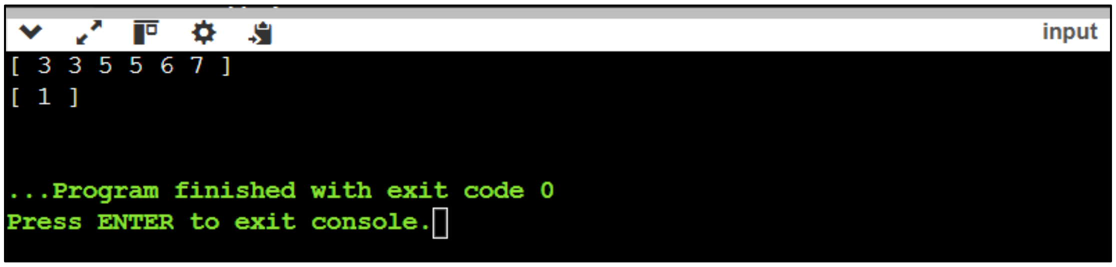
Example 2:
Input: nums = [1], k = 1
Output: [1]

Constraints:
- 1 <= nums.length <= 105
- -104 <= nums[i] <= 104
- 1 <= k <= nums.length

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <vector>
#include <deque>

std::vector<int> maxSlidingWindow(std::vector<int>& nums, int k) {
    std::deque<int> dq;
    std::vector<int> result;

    for (int i = 0; i < nums.size(); i++) {
        if (!dq.empty() && dq.front() == i - k) {
            dq.pop_front();
        }

        while (!dq.empty() && nums[dq.back()] < nums[i]) {
            dq.pop_back();
        }

        dq.push_back(i);

        if (i >= k - 1) {
            result.push_back(nums[dq.front()]);
        }
    }

    return result;
}

int main() {
    std::vector<int> nums = {1, 3, -1, -3, 5, 3, 6, 7};
    int k = 3;
    std::vector<int> result = maxSlidingWindow(nums, k);
    std::cout << "[ ";
    for (int val : result) {
        std::cout << val << " ";
    }
    std::cout <<"]"<<std::endl; // Output: 3 3 5 5 6 7

    std::vector<int> nums2 = {1};
    int k2 = 1;
    std::vector<int> result2 = maxSlidingWindow(nums2, k2);
    std::cout << "[ ";
    for (int val : result2) {
        std::cout << val << " ";
    }
    std::cout <<"]"<<std::endl; // Output: 1
    return 0;
}
```

Output:



```
[ 3 3 5 5 6 7 ]
[ 1 ]


...Program finished with exit code 0
Press ENTER to exit console.
```

5. Poisonous Plants *(Very Hard)*

There are a number of plants in a garden. Each of the plants has been treated with some amount of pesticide. After each day, if any plant has more pesticide than the plant on its left, being weaker than the left one, it dies.

You are given the initial values of the pesticide in each of the plants. Determine the number of days after which no plant dies, i.e. the time after which there is no plant with more pesticide content than the plant to its left.

Example 1:

Input:
p = [3,6,2,7,5]   // pesticide levels
Output:
2

Use a 1-indexed array. On day 1, plants 2 and 4 die leaving p'=[3,2,5] . On day 2, plant 3 in p' dies leaving p"=[3.2] . There is no plant with a higher concentration of pesticide than the one to its left, so plants stop dying after day 2 .

Function Description:

Complete the function poisonousPlants in the editor below.

poisonousPlants has the following parameter(s):
int p[n]: the pesticide levels in each plant
Returns
- int: the number of days until plants no longer die from pesticide

Input Format
- The first line contains an integer n , the size of the array p.
- The next line contains n space-separated integers p[i].

Constraints
- $1 \leq n \leq 10^5$
- $1 \leq p[i] \leq 10^9$

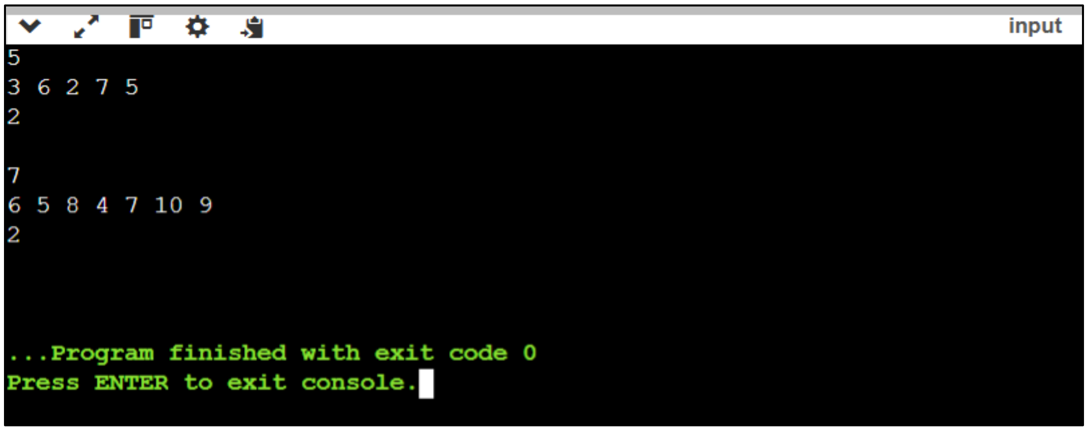
Example 2:

Sample Input:
7
6 5 8 4 7 10 9
Sample Output:
2
Explanation:
Initially all plants are alive.
Plants = {(6,1), (5,2), (8,3), (4,4), (7,5), (10,6), (9,7)}
Plants[k] = (i,j) => jth plant has pesticide amount = i.
After the 1st day, 4 plants remain as plants 3, 5, and 6 die.
Plants = {(6,1), (5,2), (4,4), (9,7)}
After the 2nd day, 3 plants survive as plant 7 dies.
Plants = {(6,1), (5,2), (4,4)}
Plants stop dying after the 2nd day.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <vector>
#include <stack>
#include <algorithm>

int poisonousPlants(std::vector<int>& p) {
        int n = p.size();
        std::vector<int> days(n, 0);
        std::stack<int> s;

        int maxDays = 0;

        for (int i = 0; i < n; i++) {
            int day = 0;

            while (!s.empty() && p[s.top()] >= p[i]) {
                day = std::max(day, days[s.top()]);
                s.pop();
            }

            if (!s.empty()) {
                days[i] = day + 1;
            }

            s.push(i);
            maxDays = std::max(maxDays, days[i]);
        }

        return maxDays;
    }

    int main() {
        int j=0;
        while(j<2)
        {
            int n;
            std::cin >> n; // Input: size of the array
            std::vector<int> p(n);

            for (int i = 0; i < n; i++) {
                std::cin >> p[i]; // Input: pesticide levels of plants
            }

            int result = poisonousPlants(p);
            std::cout << result << std::endl<<std::endl; //
            j++;
        }

        return 0;
    }
```

Output:



```
5
3 6 2 7 5
2

7
6 5 8 4 7 10 9
2



...Program finished with exit code 0
Press ENTER to exit console.
```