Department of Computer Science and Engineering

Name:- Vikash Ranjan Kumar UID:- 22BCS11322 Section:- 22KPIT-901-B

DAY-4

Q.1 You are given two arrays:

1. students: an array representing the type of sandwiches each student prefers, where:
   - 1 represents a student who prefers a sandwich with cheese.
   - 0 represents a student who prefers a sandwich without cheese.
2. sandwiches: an array representing the type of sandwiches available, where:
   - 1 represents a sandwich with cheese.
   - 0 represents a sandwich without cheese.

The task is to determine the number of students who are unable to eat a sandwich, given the sandwiches available. Students will eat the first sandwich in the stack that matches their preference, and if a student's preference is not met, they will go to the back of the line. The process continues until all sandwiches are processed or no student can eat.

Write a function countStudents that returns the number of students who cannot eat.

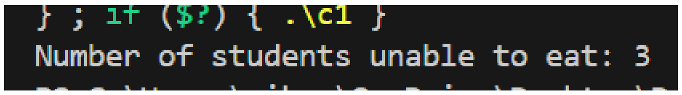
Program Code:-

```
#include <stack>
```

```
#include <iostream>
using namespace std;

class MyQueue {
private:
    stack<int> inputStack;
    stack<int> outputStack;

    // Helper function to transfer elements from inputStack to outputStack
    void transferElements() {
        while (!inputStack.empty()) {
            outputStack.push(inputStack.top());
            inputStack.pop();
        }
    }

public:
    // Constructor
    MyQueue() {}

    // Push element to the back of the queue
    void push(int x) {
        inputStack.push(x);
    }

    // Pop the front element from the queue
    int pop() {
        // If outputStack is empty, transfer elements from inputStack
        if (outputStack.empty()) {
            transferElements();
        }
        // Pop and return the top element of outputStack (front of the queue)
        int front = outputStack.top();
        outputStack.pop();
        return front;
    }

    // Return the front element of the queue without removing it
    int peek() {
        // If outputStack is empty, transfer elements from inputStack
        if (outputStack.empty()) {
            transferElements();
        }
        // Return the top element of outputStack (front of the queue)
        return outputStack.top();
    }
```

```
    // Return true if the queue is empty, false otherwise
    bool empty() {
        return inputStack.empty() && outputStack.empty();
    }
};

int main() {
    MyQueue myQueue;

    // Push elements to the queue
    myQueue.push(1);
    myQueue.push(2);

    // Get the front element
    cout << "Front element (peek): " << myQueue.peek() << endl;  // Output: 1

    // Pop the front element
    cout << "Popped element: " << myQueue.pop() << endl;  // Output: 1

    // Check if the queue is empty
    cout << "Is queue empty? " << (myQueue.empty() ? "Yes" : "No") << endl;  //
Output: No
```

```
    return 0;
}
```
Output:-



```
} ; if ($?) { .\c1 }
Number of students unable to eat: 3
```

Q.2 You are given an integer array nums and an integer k.

Find the longest subsequence of nums that meets the following requirements:

- The subsequence is strictly increasing and
- The difference between adjacent elements in the subsequence is at most k.

Return the length of the longest subsequence that meets the requirements.

A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input: nums = [4,2,1,4,3,4,5,8,15], k = 3

Output: 5

Explanation:

The longest subsequence that meets the requirements is [1,3,4,5,8].

The subsequence has a length of 5, so we return 5.

Note that the subsequence [1,3,4,5,8,15] does not meet the requirements because 15 - 8 = 7 is larger than 3.

Example 2:

Input: nums = [7,4,5,1,8,12,4,7], k = 5

Output: 4

Explanation:

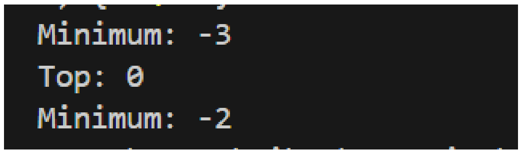
The longest subsequence that meets the requirements is [4,5,8,12].

The subsequence has a length of 4, so we return 4..

Program Code:-

```
#include <iostream>

#include <stack>

using namespace std;


class MinStack {

private:

    stack<int> mainStack;

    stack<int> minStack;

public:

    MinStack();

    void push(int val) {

        mainStack.push(val);
        if (minStack.empty() || val <= minStack.top()) {
            minStack.push(val);
        }
    }
    void pop() {
        if (mainStack.top() == minStack.top()) {
            minStack.pop();
        }
        mainStack.pop();
    }
    int top() {
        return mainStack.top();
    }
    int getMin() {
        return minStack.top();
    }
};

int main() {
    MinStack minStack;
    minStack.push(-2);

    minStack.push(0);

    minStack.push(-3);

    cout << "Minimum: " << minStack.getMin() << endl;

    minStack.pop();

    cout << "Top: " << minStack.top() << endl;

    cout << "Minimum: " << minStack.getMin() << endl;

    return 0;

}
```

Output:-



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

Q.3. You are given an integer array nums and an integer k.

Find the longest subsequence of nums that meets the following requirements:

- The subsequence is strictly increasing and
- The difference between adjacent elements in the subsequence is at most k.

Return the length of the longest subsequence that meets the requirements.

A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input: nums = [4,2,1,4,3,4,5,8,15], k = 3

Output: 5

Explanation:

The longest subsequence that meets the requirements is [1,3,4,5,8].

The subsequence has a length of 5, so we return 5.

Note that the subsequence [1,3,4,5,8,15] does not meet the requirements because 15 - 8 = 7 is larger than 3.

Example 2:

Input: nums = [7,4,5,1,8,12,4,7], k = 5

Output: 4

Explanation:

The longest subsequence that meets the requirements is [4,5,8,12].

The subsequence has a length of 4, so we return 4.

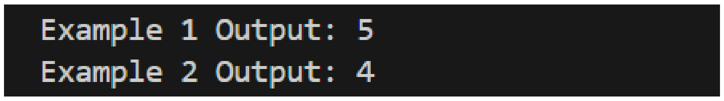
Program Code:-

```
#include <iostream>

#include <vector>

#include <map>

#include <algorithm>
```

```
using namespace std;


int longestSubsequence(vector<int>& nums, int k) {
    map<int, int> dp;  // Map to store the longest subsequence ending at a specific
value
    int maxLength = 0;


    for (int num : nums) {
        int maxPrev = 0;
        for (int x = num - k; x <= num - 1; ++x) {
            if (dp.count(x)) {
                maxPrev = max(maxPrev, dp[x]);
            }
        }
        dp[num] = maxPrev + 1;
        maxLength = max(maxLength, dp[num]);
    }


    return maxLength;
}
```

```
int main() {

    vector<int> nums1 = {4, 2, 1, 4, 3, 4, 5, 8, 15};

    int k1 = 3;

    cout << "Example 1 Output: " << longestSubsequence(nums1, k1) << endl;


    vector<int> nums2 = {7, 4, 5, 1, 8, 12, 4, 7};

    int k2 = 5;

    cout << "Example 2 Output: " << longestSubsequence(nums2, k2) << endl;


    return 0;

}
```

Output:-



```
Example 1 Output: 5
Example 2 Output: 4
```

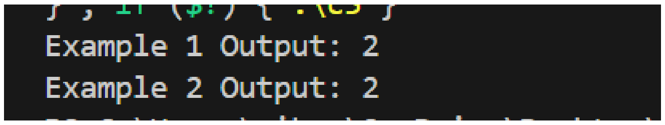
Q.4. There are a number of plants in a garden. Each of the plants has been treated with some amount of pesticide. After each day, if any plant has more pesticide than the plant on its left, being weaker than the left one, it dies. You are given the initial values of the pesticide in each of the plants. 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 p = [3,6,2,7,5] // pesticide levels 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≤n≤105 ● 1≤p[i]≤109 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.

Program Code:-

```
#include <iostream>

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;



int poisonousPlants(vector<int>& p) {

    int n = p.size();

    vector<int> days(n, 0); // Array to store the number of days each plant survives
before dying

    stack<int> s;           // Stack to track indices of plants

    int maxDays = 0;



    for (int i = 0; i < n; ++i) {
        int day = 0;
        while (!s.empty() && p[s.top()] >= p[i]) {
            day = max(day, days[s.top()]);
            s.pop();
        }
        if (!s.empty()) {
            days[i] = day + 1;
        }
        s.push(i);
        maxDays = max(maxDays, days[i]);
    }

    return maxDays;
}

int main() {
    vector<int> plants1 = {3, 6, 2, 7, 5};
    cout << "Example 1 Output: " << poisonousPlants(plants1) << endl;

    vector<int> plants2 = {6, 5, 8, 4, 7, 10, 9};
    cout << "Example 2 Output: " << poisonousPlants(plants2) << endl;


    return 0;
}
```

Output:-



```
Example 1 Output: 2
Example 2 Output: 2
```

Q.5. Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty). Implement the MyQueue class: void push(int x) Pushes element x to the back of the queue. int pop() Removes the element from the front of the queue and returns it. int peek() Returns the element at the front of the queue. boolean empty() Returns true if the queue is empty, false otherwise. Notes: You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid. Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations. Example 1: Input ["MyQueue", "push", "push", "peek", "pop", "empty"] [[], [1], [2], [], [], []] Output [null, null, null, 1, 1, false] Explanation ● MyQueue myQueue = new MyQueue(); ● myQueue.push(1); // queue is: [1] ● myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue) ● myQueue.peek(); // return 1 ● myQueue.pop(); // return 1, queue is [2] ● myQueue.empty(); // return false
Example 2: Input: ["MyQueue", "push", "push", "push", "peek", "pop", "peek", "empty"] [[], [3], [5], [7], [], [], [], []] Output: [null, null, null, null, 3, 3, 5, false] Explanation: MyQueue myQueue = new MyQueue() myQueue.push(3) # queue is: [3] myQueue.push(5) # queue is: [3, 5]

```
#include <stack>

#include <iostream>

using namespace std;


class MyQueue {

private:

    stack<int> inputStack;

    stack<int> outputStack;


    // Helper function to transfer elements from inputStack to outputStack

    void transferElements() {

        while (!inputStack.empty()) {

            outputStack.push(inputStack.top());

            inputStack.pop();

        }

    }


public:

    // Constructor

    MyQueue() {}


    // Push element to the back of the queue
```

```
void push(int x) {
    inputStack.push(x);
}

// Pop the front element from the queue
int pop() {
    // If outputStack is empty, transfer elements from inputStack
    if (outputStack.empty()) {
        transferElements();
    }
    // Pop and return the top element of outputStack (front of the queue)
    int front = outputStack.top();
    outputStack.pop();
    return front;
}

// Return the front element of the queue without removing it
int peek() {
    // If outputStack is empty, transfer elements from inputStack
    if (outputStack.empty()) {
        transferElements();
    }
```

```
        // Return the top element of outputStack (front of the queue)
        return outputStack.top();
    }

    // Return true if the queue is empty, false otherwise
    bool empty() {
        return inputStack.empty() && outputStack.empty();
    }
};

int main() {
    MyQueue myQueue;

    // Push elements to the queue
    myQueue.push(1);
    myQueue.push(2);

    // Get the front element
    cout << "Front element (peek): " << myQueue.peek() << endl;  // Output: 1

    // Pop the front element
    cout << "Popped element: " << myQueue.pop() << endl;  // Output: 1
```

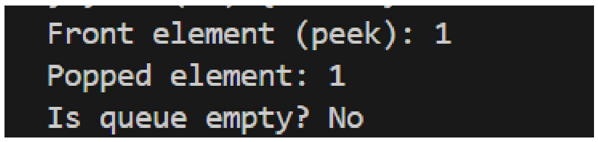
    // Check if the queue is empty

    cout << "Is queue empty? " << (myQueue.empty() ? "Yes" : "No") << endl; // Output: No

    return 0;

}

Output:-



```
Front element (peek): 1
Popped element: 1
Is queue empty? No
```