M23CS1.304 Data Structures and Algorithms for Problem Solving

Assignment 2

Deadline: 11:59 p.m. September 16th, 2023

Important Points:

1. **Only C++ is allowed**.
2. **Directory Structure:**
```
2023201001_A2
        |_____2023201001_A2_Q1a.cpp
        |_____2023201001_A2_Q1b.cpp
        |_____2023201001_A2_Q2.cpp
        |_____2023201001_A2_Q3.cpp
        |_____2023201001_A2_Q4.cpp
```
   Replace your roll number in place of 2023201001

3. **Submission Format:** Follow the above mentioned directory structure and zip the RollNo_A2 folder and submit RollNo_A2.zip on Moodle.
   Note: All submissions which are not in the specified format or submitted after the deadline will be awarded **0** in the assignment.
4. C++ STL (including vectors) is **not allowed** for any of the questions unless specified otherwise in the question. So "#include <bits/stdc++.h>" is **not** allowed.
5. You can ask queries by posting on Moodle.

Any case of plagiarism will lead to a 0 in the assignment or "F" in the course.

1. Priority Queue

a. Problem Statement: Implement priority queue

What is a priority queue?

- A priority queue is a data structure that maintains a collection of elements, each associated with a priority or value. Elements are stored in a way that allows the retrieval of the element with the highest (or lowest) priority quickly.
- Unlike deques or lists, a priority queue doesn't maintain the elements in any specific order based on their values, except for ensuring that the highest (or lowest) priority element is readily accessible.

Operations : The C++ standard specifies that a legal (i.e., standard-conforming) implementation of priority queue must satisfy the following performance requirements.

1. priority_queue() - initialize an empty priority queue. Time complexity: O(1)
2. int size() - returns the current size of the priority queue. Time complexity: O(1)
3. void push(int el) - insert an element *el* in the priority queue. Time complexity: O(log(size()))
4. int top() - returns the top (highest or lowest priority) element in the priority queue. Time complexity: O(1)
5. void pop() - remove the top element of the priority queue. Time complexity: O(log(size()))
6. bool empty() - returns true if the priority queue is empty else returns false. Time complexity: O(1)

b. Problem Statement : *David's Baker*y has introduced a new promotion for students. Under this promotion, he monitors his daily sales, and if the sales on a specific day is greater than or equal to the combined sum of the median sale for a trailing number of *d* days and the median sale from the first day of the promotion, then he offers free *'Cheeeeeese Maggi'* to the first five customers on the following day. However, David will only provide this free offer if he has transaction data for at least the trailing number of *d* prior days.

Given the number of trailing days *d* and the daily sales record for David's Bakery over a period of *n* days, can you determine how many days david will offer free maggi to students during this entire period of *n* days?

Input Format :

- The first line contains two space-separated integers *n* and *d*, the number of days of daily sales data, and the number of trailing days respectively.
- The second line contains *n* space-separated non-negative integers where each integer *i* denotes *sales[i]*.

Output Format :

- *int:* the total number of days david will offer free maggi to students.

Constraints :

- $1 <= n <= 2 * 10^5$
- $1 <= d < n$
- $1 <= sales[i] <= 10^5$

Sample test cases:

a. **Sample input**:
   8 5
   2 3 4 2 1 6 3 6
   Sample output:
   2
   Explanation: For the first five days, david will not offer any free maggi to students because he didn't have sufficient sales data.

On the sixth day, David has d = 5 days worth of trailing sales data, which is {2, 3, 4, 2, 1}. The median sale of the preceding five days is 2, and he also has the entire sales data from the beginning, which remains {2, 3, 4, 2, 1}, with a median sale value of 2 as well. Since, David's sales on the 6th day is greater than the combined sum, David offers free Maggi to students for the 6th day.

On the seventh day, David has d = 5 days worth of trailing sales data, which is {3, 4, 2, 1, 6}. The median sale of the preceding five days is 3, and he also has the entire sales data from the beginning, which is {2, 3, 4, 2, 1, 6}, with a median sale value of (2 + 3) / 2 = 2.5 . Since, David's sales on the 7th day is less than the combined sum, he will not offer free Maggi to students for the 7th day.

On the eighth day, David has d = 5 days worth of trailing sales data, which is {4, 2, 1, 6, 3}. The median sale of the preceding five days is 3, and he also has the entire sales data from the beginning, which is{2, 3, 4, 2, 1, 6, 3}, with a median sale value of 3 as well. Since, David's sales on the 8th day is equal to the combined sum, David offers free Maggi to students for the 8th day.

Therefore, in total david will offer free maggi for 2 days (6th and 8th day).

Note :

1. You need to implement priority queue for Integer data type only.
2. You need to implement both minimum and maximum priority queue (Implementation details are up to you).

2. Sparse Matrix Operations

Problem Statement : A matrix is generally represented using a 2D array. A sparse matrix is a matrix which has the majority of its elements set as 0. So using a conventional matrix representation scheme wastes a lot of memory. You are required to come up with a representation of how you will store a sparse matrix in memory using an array and linked list. And after that you are required to implement an algorithm to perform addition, multiplication and transpose of the matrices. For addition and multiplication operations you will be provided two sparse matrices in conventional format (conventional format is the usual representation of a 2D matrix in row x column) and the return value should be the sum and product of the two matrices itself, respectively. Incase of transpose operation you will be provided only one sparse matrix in conventional format and you have to return its transpose.

Operations :

1. **add:**
    - Takes two matrices as input (M1 and M2).
    - Performs matrix addition on these two matrices.
    - Test cases will be designed such that size of M1 = size of M2, no need to handle the cases of unequal size
    - Print the result of matrix addition.

2. **transpose:**
    - Takes one matrix as input (M1).
    - Performs matrix transpose.
    - Print the result of matrix transpose

3. **multiply:**
    - Takes two matrices as input (M1 and M2).
    - Performs matrix multiplication on these two matrices. (calculate M1 x M2)
    - Test cases will be designed such that matrix M1 and M2 are multiplication compatible.
    - Print the result of matrix multiplication.

Input Format:

- First line contains the type of data structure.
    1. Array
    2. Linked List
- Second line of input contains type of operation
    1. Addition
    2. Transpose
    3. Multiplication
- Third line of input contains two space separated integers N1, M1, number of rows and columns of the first matrix respectively (note that for transpose there will be only one input matrix)
- Following N1 lines will contain M1 space separated elements of the matrix.
- Next line will contain two space separated integers N2, M2, number of rows and columns of the second matrix respectively. (only in the case of addition and multiplication)
- Following N2 lines will contain M2 space separated elements of matrix

Output Format:

- Print the Output matrix in conventional format (N rows, space separated M values in each row)

Constraints:

$1 <= N, M <= 1000$

Number of non-zero cells $<= \min(N*M, 10^5)$

Note:

1. You are required to implement all the three operations using both the matrix representation i.e. using arrays and as well as linked lists.
2. Make your code generic. The final code will be tested on all primitive data types: int, double, float, etc(excluding string). So make sure you handle these cases as well.
3. Do not store the whole matrix in conventional format at any point of time (not during taking the input as well as while printing). If found, you'll be penalized. Take input in sparse matrix format and perform all the operations on the sparse representation of the matrix itself.
4. You can use inbuilt sort function only for this question.

3. The Wolf of the Wall Street

Problem Statement : In the bustling world of stock trading, Harshad stands as a formidable figure. Armed with insider information, he seeks to establish an enduring legacy as a true "The Wolf of Wall Street." His ambition? To execute a series of trades so as to maximize the trade days, without incurring any losses on any day. He's now pondering the total number of ways he can achieve this monumental feat.

In pursuit of this legacy, Harshad has set certain rules for himself. He'll initiate a trade by buying stock only if he has sold on the same day, with the exception of the initial purchase. Moreover, he can make just one buy transaction per day. Now, his burning question is how many distinct ways he can accomplish his goal.

As he contemplates this challenge, Harshad seeks an efficient solution to count the number of ways he can execute these trades. His mission to carve his name in the annals of Wall Street history has just begun, and it all hinges on the mastery of these intricacies.

Input Format :-

- The first line contains an integer, N, denoting the number of days for which Harshad has insider information.
- The second line contains N space-separated integers, representing the stock prices for each day.

Output format :-

- Output a single integer, denoting the number of ways Harshad can achieve his legacy. Since the output may be large, print it as modulo 1000000007.

Constraints:-

- $1 <= N <= 10^5$
- $1 <=$ stock prices $<= 10^6$

Sample Input 1 :
6
8 1 3 5 4 7
Sample Output 1 :
2
Explanation : The maximum trade days that can be achieved is 4. This can be achieved in **2** ways:

1. The first way can be achieved by buying the first stock at a price of 1 and then selling it at 3. Next, buy again at 3 and sell at 5, followed by buying at a price of 5. The next trade involves selling at a price of 7. Thus, the total number of trade days is 4.
   Trades were done on 4 days - 2nd, 3rd, 4th and 6th.

2. The second way can be achieved by buying the first stock at a price of 1 and then selling it at 3. Next, buy again at 3 and sell at 4, followed by buying at a price of 4. The next trade involves selling at a price of 7. Total number of trade days is again 4.
   Trades were done on 4 days - 2nd, 3rd, 5th and 6th.

Sample Input 2 :-
2
2 1
Sample Output 2 :-
2
Explanation: The maximum trade days that can be achieved is 1. The trades can be done in **2** ways, either buy and sell on the 1st day or on the 2nd day.

4. Skip List:

The skip list is an example of a probabilistic data structure because it makes some of its decisions at random. While the skip list is not guaranteed to provide good performance, it will provide good performance with extremely high probability. References: Skip List

Problem Statement : Implement Skip List with following Operations.

Sl. No.	Operations	Expected Time Complexity
1	Insertion	O(log N)
2	Deletion	O(log N)
3	Search	O(log N)
4	Count element occurrences	O(log N)
5	lower_bound	O(log N)
6	upper_bound	O(log N)
7	Closest element to a value	O(log N)

Important Points:

- Implement it with a class or struct. It should be generic, with the primitive data structures (integer, float, string, etc) your code should also be able to handle class objects.
- Duplicates are allowed.
- For strings, you can simply compare them but for Class data type, you have to pass the comparator object so that you can compare two objects.
- For strings and custom Class objects, you don't need to implement the Closest Element operation.
- Instruction for Class data type: Name your custom comparator function as "comp". Strictly follow this convention as we'll also be evaluating the program on our custom class and its custom comparator function (which will be named "comp")

- In your driver code, you need to print the skip list (bottom level) after every operation.

Operations : Consider **T e**, where **T** is type of element **e**

1. void insert(e) : Inserts e into the skip list.
2. delete(e) : Deletes all the occurrences of the element e, if it is present in the skip list.
3. bool search(e) : Returns true if e is present in the skip list, otherwise returns false.
4. int count_occurence(e) : Returns the count of occurrences of the element e.
   Eg. If the skip list has the following elements: 1, 1, 2, 2, 2, 3
   count_occurence(2) will return 3.
   count_occurence(734) will return 0.
5. T lower_bound(e) : Return the first element that is greater than or equal to e. If no such element exists, return default value for type T.
   Eg. If the skip list has the following elements: 1, 1, 2, 2, 2, 3
   lower_bound(2) will return an element corresponding to 2 i.e. 3rd element from the left.
   If the skip list has the following elements 1, 1, 2, 5, 6, 6, 7 lower_bound(3) will return an element corresponding to 5 i.e. 4th element from the left.
6. T upper_bound(e) : Return the first element that is greater than e. If no such element exists, return default value for type T.
   Eg. If the skip list has the following elements: 1, 1, 2, 2, 2, 3
   upper_bound(2) will return an element corresponding to 3 i.e. last element from the right.
   If the skip list has the following elements: 1, 1, 2, 5, 6, 6, 7 upper_bound(7) will return 0 i.e. the default value for the type of element e (integer in this case)
7. T closest_element(e) : Returns the element closest to e. If no such element exists, return default value for type T.
   Eg. If the tree has the following elements: 1, 1, 2, 2, 2, 5, 7.
   closest_element(2) will return 2. closest_element(3) will return 2.
   closest_element(4) will return 5. closest_element(-1472) will return 1.
   closest_element(6) will return 5 (If there are multiple closest elements present, return the minimum).