**Problems in Data Structures and Algorithms – Midterm Exam**

**Duration: 7 days**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name**: | Yusuf Furkan Yücesoy | **ID:** | 152120151005 | **Section:** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q1** | **Q2** | **Q3** | **Q4** | **Total** |
|  |  |  |  |  |

**Q1. Polynomial Addition (25 points)**

You are given two polynomials each represented as a singly linked list. You are to implement a routine AddPolynomials that takes in two polynomials and returns a **new** **polynomial** that represents the sum of the two polynomials. For example, if you are given P1 = 3x4 + 3x2 + 5 and P2 = 5x4 – 3x2 + 3x, then you must return P3 = 8x4 + 3x + 5. The test code for this function can be found in main.cpp with the name PolynomialAdditionTest(). Make sure that you test your code thoroughly. During grading we will use other tests to test your code.

**Q2. Two Dimensional Word Search (25 points)**

You are given a two dimensional array of characters as shown below:

| A B C E |

M[3][4] = | S F C S |

| A D E E |

Given a search pattern p = “ABCCED”, you are asked to find out whether the pattern exists in the two dimensional array. The rules of walking over the matrix are the following: If you are at a location [i, j], you are only allowed to move left [i, j-1], right [i, j+1], up [i-1, j] or down [i+1, j]. Furthermore, you cannot move to a place you have already visited.

Notice that the given pattern p = “ABCCED” exists in the matrix as follows. We start at [0, 0], that contains ‘A’. We then move right to B, then to right to C, then down to C, then down to E and then to left to D, which is the end of the given pattern.

You algorithm TwoDimensionalWordSearch takes in the two dimensional array of characters and the search pattern, and returns either true if the pattern exists in the two dimensional array, or false if the pattern does not exist. If the pattern exists, you must also return the sequence of indices that make up the pattern.

Fill in the function TwoDimensionalWordSearch in Midterm.cpp. The test code for this function can be found in main.cpp with the name TwoDimensionalWordSearchTest(). Make sure that you test your code thoroughly. During grading we will use other tests to test your code.

**Q3. Interval Sum (25 points)**

You are given an array of numbers where each number occurs at least twice. You are asked to implement an algorithm that takes such an array as input and returns another array that contains the sum of the numbers between a number A[i] and the last occurrence of the same number in A[0..i-1]. If this is the first occurrence of a number, then the sum is 0. For example, assume you are given A=[2, 5, 7, 2, 6, 7, 6, 5, 6, 5]. Then Sum=[0, 0, 0, 12, 0, 8, 7, 28, 5, 6]. Here is how we obtain Sum:

|  |  |
| --- | --- |
| A[0]=2 | Since this is the first occurrence of 2, Sum[0] = 0 |
| A[1]=5 | Since this is the first occurrence of 5, Sum[1] = 0 |
| A[2]=7 | Since this is the first occurrence of 7, Sum[2] = 0 |
| A[3]=2 | The last 2 occurs in A[0]. So Sum[3] = 5+7 = 12 |
| A[4]=6 | Since this is the first occurrence of 6, Sum[3] = 0 |
| A[5]=7 | The last 7 occurs in A[2]. So Sum[4] = 2+6 = 8 |
| A[6]=6 | The last 6 occurs in A[4]. So Sum[5] = 7 |
| A[7]=5 | The last 5 occurs in A[1]. So Sum[6] = 7+2+6+7+6 = 28 |
| A[8]=6 | The last 6 occurs in A[6]. So Sum[7] = 5 |
| A[9]=5 | The last 5 occurs in A[7]. So Sum[8] = 6 |

Your algorithm may use O(n) amount of extra space, and **must run in expected O(n) time**. If you implement the brute-force O(n2) algorithm, you will get 0 points ☺

Fill in the function IntervalSum in Midterm.cpp. The test code for this function can be found in main.cpp with the name IntervalSumTest(). Make sure that you test your code thoroughly. During grading we will use other tests to test your code.

**Q4. Merge “k” Sorted Sequences (25 points)**

You are given “k” sorted sequences of integers. The total number of elements in all sequences is equal to “n”. Each sequence is designated by two iterators “start” that points to the first element of the sequence, and “end” that marks the end of the sequence. The element pointed to “end” does not belong to the sequence. In other words, the elements of the sequence line between [begin, end). Each sequence may contain an arbitrary number of elements. Do not assume that all sequences have the same number of elements. Your job is to implement an algorithm that takes these “k” sorted sequences and merge them all into a single sorted sequence. Your algorithm must take at most O(nlogk) time. In this problem, go to Midterm.cpp, and fill in the function MergeKSortedSeqs. The test code for this function can be found in main.cpp with the name MergeKSortedSeqsTest. Make sure that you test your code thoroughly. During grading we will use other tests to test your code. Do not make any assumptions about “n” or “k”. They both may be very large numbers in the order of millions. So, implement your code efficiently. Any implementation that does not work in O(nlogk) will get 0.