National University of Computer & Emerging Sciences, Karachi
Fall-2019 CS-Department
Midterm II
November 04, 2019 Slot: 1 PM – 2 PM

Course Code: CS 218	**Course Name: Data Structures**
Instructor Name:	**Dr. Muhammad Rafi / Ms. Nida Pervaiz**
Student Roll No:	**Section No:**

- Return the question paper.
- Read each question completely before answering it. There are **3 questions and 2 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict with any statement in the question paper.
- All the answers must be solved according to the sequence given in the question paper.
- Be specific, to the point while coding, logic should be properly commented, and illustrate with diagram where necessary.

Time: 60 minutes. **Max Marks**: 50 points

Linked List and Variants	
Question No. 1	**[Time: 15 Min] [Marks: 15]**

a. You are given two pointers **p** and **q** of type compatible with doubly linked list node type. These two pointers point to the extreme nodes of the doubly linked list one at head and other at tail. You need to swap the two node such that a valid object of doubly linked list do exist after the operation (You are not supposed to swap data of the nodes). [5]

```
void swapExtremeDNodes( DNode<T> * p, DNode<T> * q) {

        q->next=p->next
        p->next=NULL;
        p->prev=q->prev
        q->prev=NULL
        q->next->prev=q
        p->prev->next=q

}
```

b. You are given an object of a singly linked list of length n, which contains all integers values between (1 to m). You need to find whether two adjacent nodes values sum to an integer Y or not for a given list. Only the head pointer is available in the singly linked list. Your function should return TRUE if such a pair of adjacent nodes do exist in the list or FALSE otherwise. [10]

```
bool AdjacentNodeSumToY( Node<int> *Head, int Y) {

        Node* temp=Head;
        while(temp->next != NULL ) {
          if(temp->data + temp->next->data == Y) {return true; break;}
         else
        {temp=temp->next;}

      } // while

} // function
```

Sorting Algorithms	
Question No. 2	**[Time: 20 Min] [Marks: 15]**

a. You are given an array of integers; find the maximum product of three integers from the given list. You can consider that the array is circular in nature. You only need to print the product. (Hint: find an efficient solution for it) [5]

Algorithm Max. Product of Three- from a circular array
Input: Array A [] of length n
Output: product – maximum from three elements over the array

1. Sort(A[], n) - using any sort of complexity n log n (merge-sort) in non-increasing order
2. Compute Maximum M1 from (A[0], A[1], A[2])
3. If ((A[n-1] < 0) and (A[n-2] < 0)) Computer Maximum M2 from (A[0], A[n-2], A[n-1]) else M2=0;
4. return max_of (M1,M2);

b. You are given two sorted arrays of positive integers **A[]** and **B[]** of the same size (say n). Assume $\mathbf{m_A}$ and $\mathbf{m_B}$ are the median of the two arrays. Find the median of an array combined with arrays A and B.
(Note: The median of the array is the quantity lying at the midpoint of a frequency distribution of observed values or quantities, such that there is an equal probability of falling above or below it.) [10]

Pseudocode # 1– O(n) time approach.

```
Algorithm MedianOfTheTwoArraysOfSameLength
Input: Array A [] and Array B [] of length n
Output: median
Assume m1 is the median of Array A
Assume m2 is the median of Array B
Count and skip, for index i, smallest n elements from both A and B, if A is empty return m2=B[i],
else if B is empty return m1=A[i], else m1=A[i+1] and m2=B[i+1+1] return (m1+m2)/2;
```

Pseudocode # 2– O(n) time approach.

```
Algorithm MedianOfTheTwoArraysOfSameLength
Input: Array A [] and Array B [] of length n
Output: median
Merge the two sorted Arrays A and B, you will get a third Array C of length 2n.
/* => 2n will be an even number by definition of even number. Hence the median will be the
average of the two middle elements. */
The median will be  m= {C[n] + C[n+1]} / 2;
```

Pseudocode # 3– O(log n) time approach.

```
Algorithm MedianOfTheTwoArraysOfSameLength
Input: Array A [] and Array B [] of length n
Output: median

1. Calculate median m1 for array A
2. Calculate median m2 for array B
3. If m1 == m2 return m1;
4. If  m1 > m2 then median will be from A[n/2….n] and B[1…n/2]
5. If m2 > m1 then median will be from B[n/2….n] and A[1…n/2]
6. Goto step 1 if A and B both contains more than 2 elements.
7. Else return m = (max(A[i], B[i]) + min(A[i+1], B[i+1]))/2;
```

Stacks and Queues	
Question No. 3	**[Time: 25 Min] [Marks: 20]**

a. Given the algorithm below, you need to implement it as a program code. You can use already defined data structures like Stack and Queues (Any implementation discussed during lecture) to achieved the objective of this algorithm. [10]

Algorithm: Convert Infix to prefix expression
Input: Valid Infix Expression
Output: Valid Prefix Expression

Step 1: Read the expression from left to right one character at a time
Step 2: repeat till step 7, until the end of infix expression
Step 3: Initialize two character stacks S1 and S2
Step 4: if symbol is an operand put into S1
Step 5: if symbol is left parenthesis push into operator stack S2
Step 6. If right parenthesis, pop two operand and one operator and push string **operator + operand1 + operand 2** in operand stack until left parenthesis is found.
Step 7: if scan symbol is an operator then
Step 7.1: if operator has same or less precedence then operator available on the top of operator stack, then pop one operator and two operands form prefix expression & push (i.e. push string **operator + operand1 + operand 2)** into operand stack.
Step 8: pop all the remaining operator and corresponding operands and form prefix expression.
Step 9: The final prefix expression is present at top of operand stack. Exit.

```
#include <bits/stdc++.h>
using namespace std;

bool isOperator(char c)
{
   return (!isalpha(c) && !isdigit(c));
}


int getPriority(char C)
{
   if (C == '-' || C == '+')
      return 1;
   else if (C == '*' || C == '/')
      return 2;

   return 0;
}
```

```
string infixToPrefix(string infix)
{
   stack<char> operators;
   stack<string> operands;

   for (int i = 0; i < infix.length(); i++) {

      if (infix[i] == '(') {
         operators.push(infix[i]);
      }

      else if (infix[i] == ')') {
         while (!operators.empty() &&
               operators.top() != '(') {


            string op1 = operands.top();
            operands.pop();

            string op2 = operands.top();
            operands.pop();

            char op = operators.top();
            operators.pop();

            string tmp = op + op2 + op1;
            operands.push(tmp);
         }

          operators.pop();
      }
      else if (!isOperator(infix[i])) {
         operands.push(string(1, infix[i]));
      }

      else {
         while (!operators.empty() &&
               getPriority(infix[i]) <=
                 getPriority(operators.top())) {

            string op1 = operands.top();
            operands.pop();

            string op2 = operands.top();
            operands.pop();

            char op = operators.top();
            operators.pop();

            string tmp = op + op2 + op1;
            operands.push(tmp);
         }

         operators.push(infix[i]);
      }
   }
```

```
    while (!operators.empty()) {
        string op1 = operands.top();
        operands.pop();

        string op2 = operands.top();
        operands.pop();

        char op = operators.top();
        operators.pop();

        string tmp = op + op2 + op1;
        operands.push(tmp);
    }


    return operands.top();
}
```

b. One of the media channels is looking to decide non-overlapping intervals for their transmission. Each program from the channel comprises of a start time, duration and commercial time for the program. For example, an Interval (start_time, duration, ctime) = (2, 6, 3) implies that the program starts at 2 and can be finish by 11 with commercial time included for on airing the show. The program managers collect all programs information is a single file. All you need is a combined data structures that maintains non-overlapping compatible set of programs for the channel and maintain a separate file for all those program that are conflicting in time. For example, consider the file holding the entire programs list as input and the respective output in the form of compatible and conflicting programs list:

Programs list
2 3 2
9 10 1
16 4 4
7 5 3
3 3 1

Compatible Programs	**Conflicting Programs**
2 3 2	3 3 1
7 5 3	9 10 1
16 4 4	

Consider programs list as a 1D array which holds entire list of unsorted programs where each node of array holds three data variables; start_time, duration and ctime. Your first task to solve such problem should be sorting this list out.

1. Mention the sorting algorithm which you would want to use (do not code just mention the name and the reason as to why this sorting algorithm should be used). Mention the variable(s) on which sorting should be done for entire array. [3]

   Sort the intervals on start time using Merge Sort.

2. Now assume that list gets sorted up, code your program for using this sorted list in order to create two separate lists; compatible and conflicting programs list. Your code should deal the array of programs list as queue whereas compatible and conflicting programs list should be dealt as stack. [7]

```
Stack<interval> compatible;
Stack<Interval> Conflict;

int last=time=0;
for (int i=0; i<size; i++)
{
  temp=array[i].dequeue();
  time=array[i]->start + array[i]->duration + array[i]->ctime;
  if(time>=last)
  {
     compatible.push(array[i]);
      last=time;
   }
   else
    conflict.push(array[i]);

 }
```