National University of Computer & Emerging Sciences, Karachi

Fall-2020 CS-Department

Midterm II

November 23, 2020 Slot: 9 AM – 10 AM

Course Code: CS 218	**Course Name: Data Structures**
Instructor Name:	**Muhammad Rafi / Dr. Ghufran/Basit Jasani/ Zain ul Hassan**
Student Roll No:	**Section No:**

- Return the question paper.
- Read each question completely before answering it. There are **3 questions and 1 page, 2 sides.**
- 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**: 40 points

Advanced Sorting Techniques	
Question No. 1	**[Time: 20 Min] [Marks: 15]**

a. Insertion sort goes sequentially through the array when making comparisons to find a proper place for an element currently processed from right to the left of the sorted array. As the input increases this time increases as well, hence making it quite inefficient. A Fastian develop a solution to this problem. He suggested to use Binary Search for inserting every ith element of the array to the left part of the sorted array. He called it BinaryInsertionSort. You are required to write the code of BinaryInsertionSort and discuss the worst time complexity of this algorithm.

```
void BinaryInsertionSort (int a[], int n)
{
    int ins, i, j;
    int tmp;
    for (i = 1; i < n; i++) {
        ins = BinarySearch (a, 0, i, a[i]);
        tmp = a[i];
        for (j = i - 1; j >= ins; j--)
            a[j + 1] = a[j];
        a[ins] = tmp;
    }
}

int BinarySearch (int a[], int low, int high, int key)
{
    int mid;
    if (low == high)
        return low;
    mid = low + ((high - low) / 2);
    if (key > a[mid])
        return BinarySearch (a, mid + 1, high, key);
    else if (key < a[mid])
        return BinarySearch (a, low, mid, key);
    return mid;
}
```

b. An IndirectIndexSort is a sorting algorithm that prepare an index over a collection. Given an array X [] of **n** numbers, this sort will produced an array Y[] of indexes of sorted order of these numbers. For example X[] ={13,29,91,34,56} then Y[] ={0,1,3,4,2} Write an efficient implementation of this approach, you cannot move elements of array X. Note that X[] may contains duplicate number and your IndirectIndexSort should be stable in nature.

```
template<class T>
void Indirectsort(T data[], int index[], const int n) {
    for (int i = 0, least,temp, j; i < n; i++) {
        least=i;
        for (j = i+1; j < n; j++)
        {
            if (data[index[least]] > data[index[j]])
                least = j;
        }
         if (i != least)
             {
             temp=index[i];
             index[i]=index[least];
             index[least]=temp;

             }

    }
}
```

Stack, Queue and Priority Queue	
Question No. 2	**[Time: 10 Min] [Marks: 10]**

A fellow Fastian propose a new data structures DualStack, which is an array based implementation that supports two different stacks in the same linear array. He proposed pair of functions like (Push1, Push 2), (Top1, Top2) and (Pop1, Pop2) such that these methods efficiently utilize the available space within the array. There should not be any overflow. Define the template based class for DualStack with proper functionality of each operator on this ADT that is in Bold text.

```
template <class T>
class DualStack
{
private:
 T *data;
 int top1;
 int top2;
 unsigned int Max_size;
public:
DualStack(unsigned int Max_Size = 100, int top1, int top2);
~DualStack();
DualStack( const DualStack & rhs);
DualStack & operator = ( const DualStack & rhs );

void Push1(T el);
void Push2(T el);
T& Pop1();
T& Pop2();
 isFull();
 isEmpty();

 //rest of declarations
 };
```

The most suitable idea is to maintain the two stacks in a linear memory (array) from the two ends (left and right). Let's the left end stack is Stack1 and right end stack is Stack2.

The constructor initializes both index of stacks as top1=-1 and top2=Max_size, we will use Stack1 by incrementing top1 and Stack2 by decrementing top2; we only need to check that top1 should be less than top2+1;

-1	**0**	**1**	**2**	**3**	**.**	**..**	**..**	**Max_size-1**	**Max_size**
top1									**top2**

```
bool IsFull() {
  return(top1==top2-1);
}
bool IsEmpty() {
 return((top1==-1) &&(top2==max_size))
}
void Push1(T & el) {
if (top1 < top2 - 1) {
            top1++;
            stk[top1] = el;
        }
}
void Push2(T & el) {
if (top1 < top2 - 1) {
            top2--;
            stk[top2] = el;
        }
}
T& Pop1(){
if (top1 >= 0) {
            T x = stk[top1];
            top1--;
            return x;
        }
}
T& Pop2(){
if (top2 < Max_size) {
          T x = stk[top2];
            top2++;
            return x;
        }

}
```

Trees and its variants	
Question No. 3	**[Time: 15 Min] [Marks: 15]**

a. Write a recursive function that takes a Binary Search Trees (BST) root pointer and return height of the given tree. [5]

```
int HeightOfTheTree(const BTNode<T> *tree)
```

```
Height of a Binary Tree:  The height for a null tree is 0,
which the height of a binary tree is the distance from root
to the farthest leaf node.

int HeightOfTheTree(BTNode* root)
{
    // Base case: empty tree has height 0
    if (root == 0)
        return 0;

 // recur for left and right subtree and consider farthest
//each call add one  to the height
    return 1 + max(HeightOfTheTree(root->left),
HeightOfTheTree(root->right));
}
```

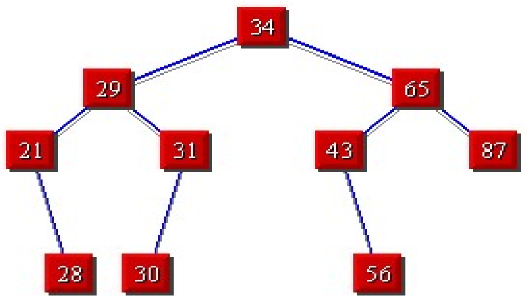
b. Write a recursive function that takes a Binary Search Trees (BST) root pointer and return size of the given tree. [5]

```
int SizeOfTheTree(const BTNode<T> *tree)
```

```
Size of a Binary Tree is defined as the numbers of node in the
tree. We can count the node recursively using the following code.

int SizeOfTheTree(BTNode* root)
{
    int size=0;
   // Base case: empty tree has 0 node in the tree
    if (root == 0)
        return size;
    else
    {// recur for count the node in the left and right subtree
     // add them together to get the total nodes ->size
    size +=SizeOfTheTree(root->left)
    size +=SizeOfTheTree(root->right)
    return size;
   }
}
```

c. Consider the following BST:



(i) What will be the pre-order sequence of the above BST? [2.5]

34,29,21,28,31,30,65,43,56,87

(ii) What will be post order sequence of the above BST? [2.5]

28,21,30,31,29,56,43,87,65,34

<The End.>