National University of Computer & Emerging Sciences, Karachi
Fall-2021 CS-Department
Midterm II (Sol)
November 22, 2021 Slot: 8:30AM to 9:30AM

Course Code: CS 2001	**Course Name: Data Structures**
Instructor Name:	**Muhammad Rafi / Dr. Ali Raza/Shahbaz/ M Sohail/ Mubashra Fayyaz**
Student Roll No:	**Section No:**

- Return the question paper.
- Read each question completely before answering it. There are **4 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**: 40 points

Advanced Sorting Techniques	
Question No. 1	**[Time: 10 Min] [Marks: 10]**

We have discussed many sorting algorithms and their applications and performance characteristics. You are given the following problem to perform sorting as per the specific needs. All you need to decide how would you solve this problem and which sorting algorithm would you prefer to use and why?

You are given an array of positive integers of size n, you need to sort these integer values using sum of the digits of each integer. For example, given the following array of integers

index	0	1	2	3	4	5	6
Array	291	124	39	231	473	601	101

The sorted array is as below:

index	0	1	2	3	4	5	6
Array	101	231	124	601	291	39	473

Adding 1+0+1=2 which is the smallest value in the given array and hence appears as the very first element of the sorted array. Similarly, 6,7, 7,12,12, and 14 are the sum of the given integers in sorted order.

There are two things that we need in this problem

(1) Write a function that perform sorting as per the given requirement. [7.5]
(2) Write your justification why your proposed approach is good? [2.5]

```
template<class T>
int SumOfDigits(T x)
{
    if (x == 0)
       return 0;
    return (x % 10 + SumOfDigits(x / 10));
}

template<class T>
void InsertionSort(T a[], int n)
{
      for(int i=1,j; i <n; i++)
      {
            T original=a[i];
            T temp=SumOfDigits(a[i]);// place i at proper position

            for(j=i; j>0 && temp < SumOfDigits(a[j-1]); j--){
                  a[j]= a[j-1];  // creating place for i
            }
            a[j]=original; //put i at the right position
      }
}
```

Insertion sort is a stable sorting algorithm; we maintain partially sorted list on the left hand side by doing comparisons of sum of the digits of every number. Insertion sort seems the best choice as we will do minimum efforts to tune it for the given task.

Stacks, Queues and Priority Queues	
Question No. 2	**[Time: 15 Min] [Marks: 10]**

a. Convert the following infix expression to postfix using algorithm discussed in class using stack. Show all your intermediate stack and output step by step.

Infix Expression = ((A + (B * C)) / (D - E))

Input String	Output Stack	Operator Stack
((A + (B * C)) / (D E))		(
((A + (B * C)) / (D E))		((
((A + (B * C)) / (D E))	A	((
((A + (B * C)) / (D E))	A	((
((A + (B * C)) / (D E))	A	((+
((A + (B * C)) / (D E))	A	((+
((A + (B * C)) / (D E))	A	((+(
((A + (B * C)) / (D E))	A B	((+(
((A + (B * C)) / (D E))	A B	((+(
((A + (B * C)) / (D E))	A B	((+(*
((A + (B * C)) / (D E))	A B	((+(*
((A + (B * C)) / (D E))	A B C	((+(*
((A + (B * C)) / (D E))	A B C*	((+
((A + (B * C)) / (D E))	A B C*+	(
((A + (B * C)) / (D E))	A B C*+	(
((A + (B * C)) / (D E))	A B C*+	(/
((A + (B * C)) / (D E))	A B C*+	(/
((A + (B * C)) / (D E))	A B C*+	(/(
((A + (B * C)) / (D E))	A B C*+ D	(/(
((A + (B * C)) / (D E))	A B C*+ D	(/(
((A + (B * C)) / (D E))	A B C*+ D	(/(
((A + (B * C)) / (D E))	A B C*+ D	(/(
((A + (B * C)) / (D E))	A B C*+ D E	(/(
((A + (B * C)) / (D E))	A B C*+ D E	(/
((A + (B * C)) / (D E))	A B C*+ D E/	
((A + (B * C)) / (D E))	A B C*+ D E/A B C*+ D E/	

Postfix expression: A B C*+ D - E/

b. You developed a template based ArrayStack in class with standard stack operations. Assume that an integer instance of this class is given to you as input with some arbitrary values and you are only allowed to use as many instances of ArrayStack class as you want to sort the given stack (Input). You need to provide implementation of the function as below, which return a sorted stack, with top as maximum integer.

ArrayStack<int> ArrayStackSort(const ArrayStack<int> rhs)

```
// This function return the sorted stack
stack<int> sortStack(stack<int> &input)
{
    stack<int> tmpStack;

    while (!input.empty())
    {
        // pop out the first element
        int tmp = input.top();
        input.pop();

        // while temporary stack is not empty and top
        // of stack is greater than temp
        while (!tmpStack.empty() && tmpStack.top() > tmp)
        {
            // pop from temporary stack and push
            // it to the input stack
            input.push(tmpStack.top());
            tmpStack.pop();
        }

        // push temp in tempory of stack
        tmpStack.push(tmp);
    }

    return tmpStack;
}
```

Heap and Variants	
Question No. 3	**[Time: 15 Min] [Marks: 10]**

a. Define Heap data structure with its properties. Outline one situation in which heap data structure is very useful.

A heap is a specialized tree-based data structure that satisfied the heap properties. It is considered as almost complete binary tree and usually implemented in Arrays. It has two properties:

1. Shape Property: It is almost complete Binary Tree, which means a level is first filled to move to next level on the tree. It makes it very suitable for Arrays based implementation.

2. Order Property: The element at root is always greater than the two child nodes. There are no comparisons between sibling nodes through. (MaxHeap)

A heap has many applications, including the most efficient implementation of priority queues, which are useful in many applications. In particular, heaps are crucial in several efficient graph algorithms.

b. Write a function (with the following signature) to decide whether a given integer array satisfy the condition of a Heap data structure or not.

```
bool IsHeap ( DynamicSafeArray <int>  A[], int n)
{
    for (int i = 0; i * 2 + 1 < n; ++ i) {
        int index = (i << 1) + 1;
        if (A[i] < A[index]) {
            return false;
        }
        if (index + 1 < n && (A[i] < A[index+ 1])) {
            return false;
        }
    }
    return true;
}
```

Searching & Hashing	
Question No. 4	**[Time: 15 Min] [Marks: 10]**

a. Compare the collision resolution strategies Open Addressing vs. Separate Chaining.

Open Addressing	**Separate Chaining**
All the keys are stored only inside the hash table. No key present outside the hash table.	Keys stored outside hash table with a linked list, only pointer to the list kept inside the table.
Number of keys never exceeds hash table size.	Number of keys can be higher than hash table size.
Deletion is difficult	Deletion is easy.
No extra memory other than hash table, cache performance is good.	Extra memory due to linked structures, cache performance is not very good as compared to Open Addressing
Bucket is filled when there is no key mapped to these buckets (probing locations)	Buckets can be empty as there is no key mapped to these.

b. Taking an initially empty HasHTable of size 10, insert the following keys using hash function **h(key) = key % 7;**
(Keys appears in the following order)
984, 376, 32, 776, 49, 453, 231
Provide the content of the HashTable when collision resolution strategies used are (i) Linear Probing and (ii) Separate Chaining, also state number of collisions in each case.

Linear Probing

Index	HashTable[Index]
0	231
1	
2	
3	
4	984
5	376
6	32
7	776
8	49
9	453

No. of Collisions: 2

Separate chaining

Index	Linked List Chain
0	231
1	
2	
3	
4	984;32
5	376;453
6	776
7	49
8	
9	

No. of Collisions: 2

<The End.>