UNIVERSITY OF RUHUNA

Faculty of Engineering

End-Semester 3 Examination in Engineering: March 2021

Module Number: EE3302 **Module Name: Data Structures and Algorithms**

[Three Hours]

[Answer **all questions**, each question carries 10 marks.
This paper consists of 8 pages.]

(Q1)

(a)

(i) List three (3) properties of an algorithm and two (2) factors affecting selection of algorithms.

[1.5 marks]

(ii) State the main factor involved and the assumption regarding running time complexity analysis of algorithms.

[1.0 mark]

(iii) Briefly explain the Big Omega (Ω) notation in asymptotic complexity analysis using graphs and inequalities.

[2.0 marks]

(b)

(i) The piecewise function given below specifies the run time complexity for quick sort algorithm. Here, **q** is the array index of the pivot point, **u** and **l** are the upper and lower limit indices of the input array respectively. Specify the condi--tions for worst case runtime complexity and deduce the worst-case run-time complexity for quick sort using the given piecewise function.

$$T(u-l+1) = T(n) \begin{cases} \Theta(1) & ; u = l \\ \Theta(n) + T(q-l) + T(u-q) & ; u > l \end{cases}$$

[2.5 marks]

(ii) The code snippet given in Figure Q1 contains a code written in C Programming language for a particular sorting algorithm. Perform the asymptotic run time complexity analysis for the sorting algorithm given.

[3.0 marks]

(Q2)

(a)

(i) Briefly explain what is meant by recursion in Algorithms.

[1.0 mark]

(ii) State three (3) non-recursive sorting algorithms.

[1.5 marks]

(iii) State and briefly explain the main functions used in Merge-Sort Algorithm.

[1.5 marks]

(b)

Code snippet given in Figure Q2 shows an incomplete code written in C progr--amming language for the functions used in Heap Sort Algorithm.

(i) Complete the code in Figure Q2 by filling the missing words from 1 to 4.

[1.0 mark]

(ii) By using the given functions in Figure Q2; write a function using C program--ming language for implementing the Heap sort algorithm.

[1.0 mark]

(iii) Draw a flow chart for the algorithm that you wrote in part b (ii). You can use heapify( ) and swap( ) functions as processing blocks in the flow chart.

[2.0 marks]

(iv) Clearly showing each of the steps graphically; sort the following sequence using Heap Sort.

$$A = \{3, 1, 2, 6\}$$

[2.0 marks]

(Q3)

(a)

(i) What type of searching is involved in arrays and linked lists when the searching element is not known?

[1.0 mark]

(ii) State two (2) data structures which are used to store the addresses of each dat--a items of the hash table.

[1.0 mark]

(iii) State two (2) characteristics of a good hash function.

[1.0 mark]

(iv) Briefly explain two (2) collision handling techniques used in hashing.

[2.0 marks]

(b)

(i) Define a generic **Data_Item** structure to be used in hash table data structure having three (3) **values** of different data types and a **key** using C Programming Language.

[1.0 mark]

(ii) Figure Q3 shows a code snippet written using C programming language for defining a hash function. Using the given hash function and by using the **Data_Item** data structure you defined in part b(i) above; write a function using C programming language to **search** a **Data_Item** using a given **key**, **key_size** and **Hash_Array_size** where the collisions are handled using linear probing. The function should write to console whether the data item was found or not.

[4.0 marks]

(Q4)

(a) Tree is a non-linear data structure which can be used in searching.

(i) Briefly explain two (2) Binary Search Tree properties.

[0.5 marks]

(ii) Draw the binary search tree that is created if the following numbers are inserted into the tree in the given order.

25, 34, 56, 12, 30, 100, 40, 150

[1 mark]

(iii) Draw a balanced binary search tree containing the same numbers given in part Q4.a(ii).

[1 mark]

(iv) State two (2) properties which are specific to AVL trees.

[0.5 marks]

(v) Insert following numbers in the given order to an empty AVL tree. You must draw the resulting AVL tree after each insertion.

100, 120, 130, 110, 115, 50, 25, 70, 80, 75

[2 marks]

(b) Linked List is a linear data structure.

(i) Give one (1) advantage and one (1) disadvantage of a Linked List over an Array data structure.

[1 mark]

(ii) Write a structure or a class using C++ programming language which can be used to represent a Doubly Linked List Node.

[1 mark]

(iii) Write a method using C++ programming language to insert a Node into the front of a Double Linked List.

[1 mark]

(iv) Write a method using C++ programming language to reverse a given Doubly Linked List.

[2 marks]

(Q5)

(a)

(i) A string can be considered as a character array. A string is passed to the method as follows.

```
void reverse(string &str)
```

Write the method using C++ programming language which reverses the string using a stack data structure.

[2 marks]

(ii) *Balanced parentheses* means that each opening symbol has a corresponding closing symbol and the pairs of parentheses are properly nested. {[{ }{ }]}[( )] is a balance parentheses expression and {( )}[ ) is not a balance parentheses expression. Using stack data structure; write the method using C++ programming language as "`bool isBalParenthesis(string exp)`" which takes a string expression containing only parentheses and return whether the expression is balanced or not.

[2 marks]

b)

(i) You are given two sorted arrays. Write a method using C++ programming language which takes two arrays as inputs and print their union and intersection as given in Figure Q5.b(i).

[4 marks]

(ii) Given an array of integers and a number 'sum', write a function using C++ programming language to find and return the number of pairs of integers in the array whose addition is equal to the number 'sum'. Given in Figure Q5.b(ii) is a sample input and output to the function. In the sample in the Figure, you have 3 pairs (1, 5), (7, -1) and (1, 5) with addition equal to 6.

[2 marks]

```
void Swap(int * x, int * y)
{
      int temp = *x;
      *x = *y;
      *y = temp;
}

void Selection_Sort(int * A, int n)
{
      for(int j=0; j<(n - 1); j++)
      {
            int min_index = j;
            for(int i=n-1; i>j; i--)
            {
                  if(A[i]<A[min_index])
                  {
                        min_index = i;
                  }
            }
            Swap(A+j, A+min_index);
      }
}
```

Figure Q1: Code for the Sorting Algorithm

```
void Swap(int * x, int * y)
{
      int temp = *x;
      *x = *y;
      *y = temp;
}


void heapify(int A[], int n, int i)
{
      if(i < n/2)
      {
            int largest = i;
            int left = (2 * i) + 1;
            int right = (2 * i) + 2;

            if ((left < n) && (A[left] > A[largest]))

            {                          (1)
                  largest = --------;
            }
            if ((right < n) && (A[right] >A[largest]))
            {
                                       (2)
                  largest = --------;
            }
            if (largest != i)
            {
                  (3)
                  --------(&A[i], &A[largest]);

                  (4)
                  --------(A, n, largest);
            }
      }
}
```

Figure Q2: Incomplete code for the Functions involved in Heap Sort Algorithm

```
int Hash_Function(char * key, int key_size, int size)
{
      int sum = 0;
      int index = 0;
      for(int i=0; i< key_size; i++)
      {
            sum = sum + 'key[i]';
      }
      index = sum % size;
      return index;
}
```

Figure Q3: Code for the Hash Function

```
Input : arr1[] = {1, 3, 4, 5, 7}
        arr2[] = {2, 3, 5, 6}
Output: Union: {1, 2, 3, 4, 5, 6, 7}
        Intersection: {3, 5}
```

Figure Q5b(i): Input and Output Arrays

```
Input  :  arr[] = {1, 5, 7, -1, 5},
          sum = 6
Output :  3
```

FigureQ5b(ii): Sample input and output to the Function