UNIVERSITY OF RUHUNA

Faculty of Engineering

End-Semester 3 Examination in Engineering: August 2022

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

[Three Hours]

[Answer **all questions**, each question carries 10 marks.]

[This paper consists of 10 pages.]

Q1. a) i) Give two (2) examples each for physical data structures and logical data structures.

[1.0 mark]

ii) Draw the allocation of memory when the code snippet given in Figure Q1.a).ii) is executed. Specify the memory type (i.e., stack or heap) used by each scope.

[1.5 marks]

b) i) Give two (2) instances where the Array data structure and the Linked List data structure are different from each other.

[1.0 mark]

ii) Singly Linked List is a unidirectional linked list. Write a method to **Search** the first node with a given value in a Singly Linked List using C++ language.

[1.5 marks]

iii) Compute the best, worst, and average case time complexities of the Search operation in part Q1.b).ii) (use big O notation).

[1.5 marks]

c) i) Doubly Linked List is a bidirectional linked list. Write a C++ program for a Node class in Doubly Linked List.

[1.0 mark]

ii) Write a method for DeleteAt(int position) operation, which deletes the node at the given position, in a Doubly Linked List using C++ language. You may call existing methods to delete the values at first and last positions.

[2.5 marks]

Q2. a) i) State the main concept behind the design of a Stack data structure.

[0.5 marks]

ii) Assume a Stack data structure is implemented with **top** variable pointing to the last element **with a value** in an Array. Write a method using C++ language to implement the **POP** operation.

[1.0 mark]

iii) Explain how to check whether a given Queue is full or not, using only the **front** and **rear** variables.

[1.0 mark]

b) i) State one (1) advantage and one (1) disadvantage of Binary Search Tree (BST) over other data structures.

[1.0 mark]

ii) Draw the Binary Search Tree created after inserting the following numbers in the given order.

32, 17, 20, 5, 40, 35, 27, 8, 18

[1.0 mark]

iii) Write the order of the nodes printed in **Pre-order** traversal, when applied on the Binary Search Tree created in part Q2.b).ii).

[1.0 mark]

iv) Draw the resulting Binary Search Tree, if the node with value 17 is deleted in part Q2.b).ii).

[1.0 mark]

c) i) Compare and contrast Binary Search tree, AVL tree, and Red-Black tree.

[1.5 marks]

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

40, 60, 55, 30, 35

[2.0 marks]

Q3. a) Figure Q3.a) shows the C++ code for a particular sorting algorithm.

i) Complete the missing expressions/terms (A-D) in the code given in Figure Q3.a).

[1.0 mark]

ii) Is the sorting algorithm given in Figure Q3.a) is recursive? State two (2) non-recursive sorting algorithms.

[1.0 mark]

iii) Assume that **n** is the total inputs to the sorting algorithm, t_1 is the number of times that the if statement in the inner loop runs, and t_2 is the number of times that the swap function runs in the sorting algorithm given in Figure Q3.a).

I) Write an expression for the total running time of the algorithm (**T(n)**) using **n**, t_1, and t_2

[1.0 mark]

II) Derive an expression for t_1 in terms of **n**.

[1.0 mark]

III) State the condition for worst case runtime complexity of the algorithm and write this condition using t_1 and t_2. Thus, deduce the worst-case runtime complexity in big O notation, using the results you obtained in Q3.a).iii).I) and Q3.a).iii).II) above.

[1.0 mark]

b) A graph is a data structure consisting of a set of vertices (V) and Edges (E).

i) Figure Q3.b).i) shows a directed graph. State two (2) ways a graph can be implemented. Represent the graph given in Figure Q3.b).i) using these two (2) ways of representations.

[1.5 marks]

ii) Figure Q3.b).ii) shows an incomplete code written in C++ programming language for the Depth First Search (DFS) algorithm to traverse in a graph. Write down the missing expressions from A-D. Also write the output when the function g.DFS(2) is called where 'g' is the graph given in Figure Q3.b).i).

[1.5 marks]

iii) A minimum spanning tree is defined for a weighted undirected graph. Figure Q3.b).iii) shows a weighted undirected graph.

I) Define minimum spanning tree.

[0.5 marks]

II) Showing each of the steps graphically and by using a table if required, use the Kruskal's algorithm to obtain a minimum spanning tree for the weighted undirected graph given in Figure Q3.b).iii).

[1.5 marks]

Q4. a) i) What is an algorithm? List down two (2) properties of an algorithm.

[1.5 marks]

ii) State two (2) factors affecting the running time of an algorithm.

[1.0 mark]

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

[1.0 mark]

iv) The C++ code for the heap sort algorithm is given in Figure Q4.a).

I) Assuming the asymptotic running time of the **heapify(A,n,i)** function as **H(n)**, derive an expression for H(n) and deduce the best case and worst case values for H(n).

[1.0 mark]

II) Derive an expression for the asymptotic running time (T(n)) of the heap sort algorithm and deduce the worst-case asymptotic run time complexity (when the input array is in ascending order).

[1.5 marks]

b) i) Briefly explain the **partition** function used in Quick-Sort Algorithm. You must specify the inputs and output of the **partition** function, and the overall task performed by the **partition** function in your explanation.

[1.0 mark]

ii) Code snippet given in Figure Q4.b) shows an incomplete code written in C++ programming language for the functions used in Merge Sort Algorithm.

I) Complete the code in Figure Q4.b) by filling the missing words from 1 to 4.

[1.0 mark]

II) By using the given functions in Figure Q4.b), write a function using C++ programming language for implementing the Merge sort algorithm.

[1.0 mark]

III) Clearly showing each of the steps graphically and writing each of the steps of merge sort at each recursive calling of merge sort; sort the following sequence using Merge Sort.

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

[1.0 mark]

Q5. a) A string can be considered as a character array. A string is passed to the method as follows.

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

Write a method using C++ programming language which reverses the string using a stack data structure. Sample input and output for this function is given in Figure Q5.a).

[2.0 marks]

b) The cryptographic algorithm for the Caesar's cipher defined as follows. Encrypt a string by shifting the characters by 3 positions to the right. Decrypt by doing the reverse. The example inputs and outputs for encryption and decryption operations of this cryptographic algorithm are given in Figure Q5.b). Write functions using C++ programming language for implementation of the Encryption and Decryption operations of the Caesar's cypher.

[3.0 marks]

c) A string compression algorithm works by counting the number of consecutive equal characters and outputting the character count as given in Figure Q5.c). The data decompression algorithm works by reconstructing the original string back from the given compressed string as shown in Figure Q5.c). Write the method for decompression using C++ programming language.

[2.0 marks]

d) i) State two (2) factors that you should consider when selecting an algorithm to perform a specific task.

[1.0 mark]

ii) Write two (2) applications of algorithms.

[1.0 mark]

iii) Write a pseudocode to prepare a cup of tea.

[1.0 mark]

```
int getTemperature() {
    int* days = new int[7];
    /*
    * Some processing
    */
    return days[2];
}

int main() {
    int temp = getTemperature();
}
```

Figure Q1.a).ii): An example C++ code

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

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

Figure Q3.a): C++ code for a particular sorting algorithm

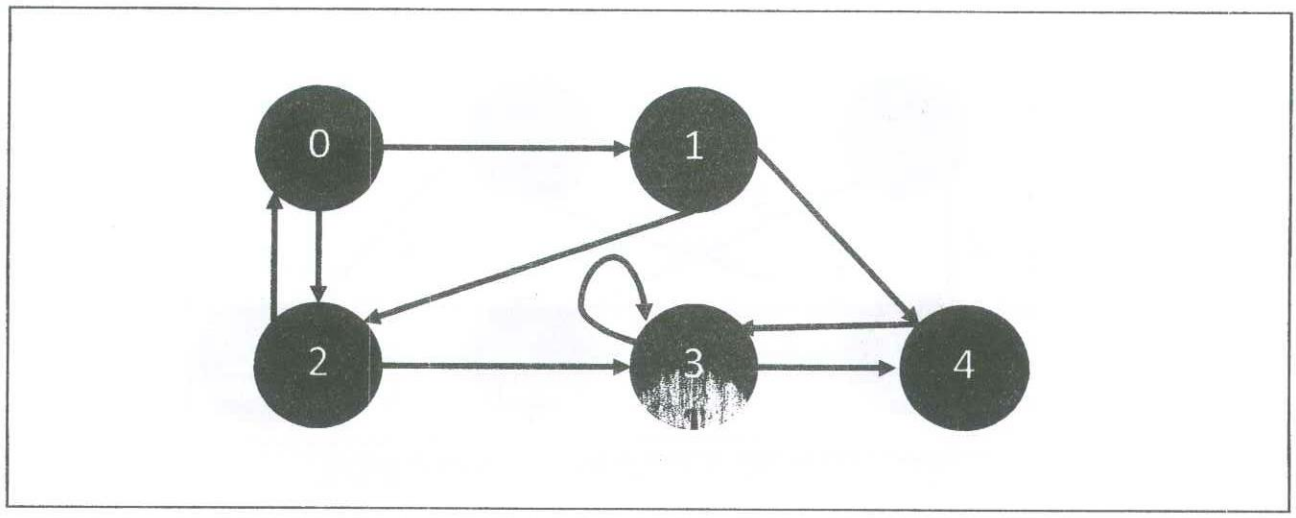


Figure Q3.b).i): A directed graph

```
void DFS(int s)
{
      bool * visited = new bool[V];
      for(int i = 0; i < V; i++)
      {
            (A);
      }
      DFS_visit(visited, s);
}


void DFS_visit(bool * visited, int k)
{
     (B);
      cout << k << ", ";
      list<int>::iterator i;
      for (i = adj[k].begin(); (C); ++i)
      {
            if (!visited[*i])
            {
                  (D);
            }
      }
}
```

Figure Q3.b).ii): C++ code for DFS in a graph with adjacency list representation

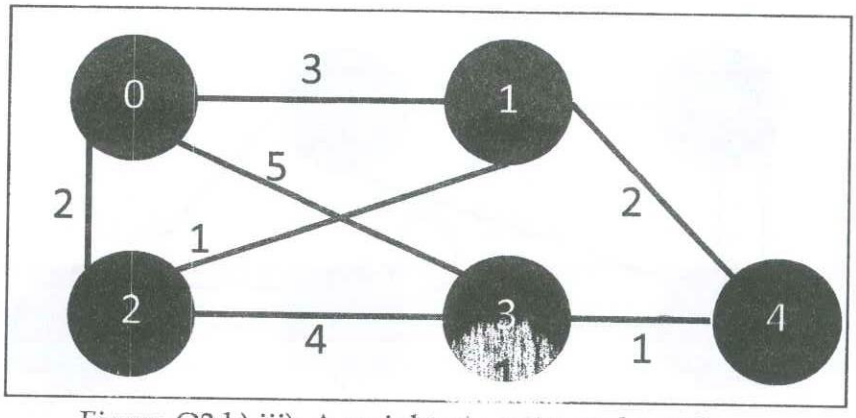


Figure Q3.b).iii): A weighted undirected graph

```
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])) {largest = left;}
          if ((right < n) && (A[right] >A[largest])){largest = right;}
          if (largest != i) {
               swap(&A[i], &A[largest]);
               heapify(A, n, largest);
          }
     }
}
void heap_sort(int A[], int n) {
     for(int i = n/2-1; i >= 0; i--) { heapify(A, n, i);}
     for(int i = n-1; i>0; i--) {
          swap(A[0], A[i]);
          heapify(A,i,0);
     }
}
```

Figure Q4.a): C++ code for the Heap Sort Algorithm

```
void Divide(int * A, int size, int * B, int * C)
{
      int half = ((size-1)/2) + 1;
      for(int i=0;i<half;i++){.....1.....}
      for(int j=half;j<size;j++){.....2.....}
}

void Merge(int * B, int * C, int * A, int size)
{
      int sizeB = ((size-1)/2) + 1;
      int sizeC = size - sizeB;
      int i=0;int j=0;int k=0;
      while ((i<sizeB) && (j<sizeC))
      {
            if(B[i] <= C[j]){A[k] = B[i];k++;i++;}
            else if(C[j] < B[i]){A[k] = C[j];k++;j++;}
      }
      if(i==sizeB)
      {
            for(int l=j;l < sizeC;l++){.....3.....;k++;j++;}
      }
      else if(j==sizeC)
      {
            for(int l=i; l<sizeB;l++){.....4.....;k++;i++;}
      }
}
```

Figure Q4.b): Functions required for Merge Sort Algorithm

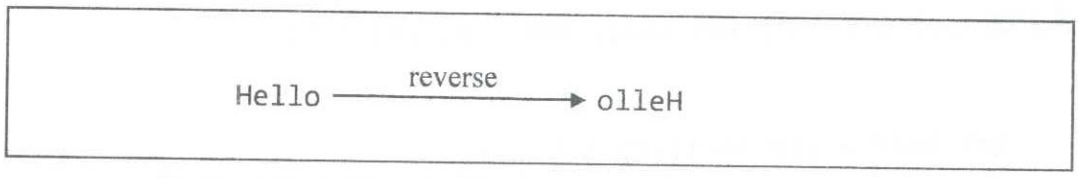


Figure Q5.a): Sample inputs and outputs for reverse function

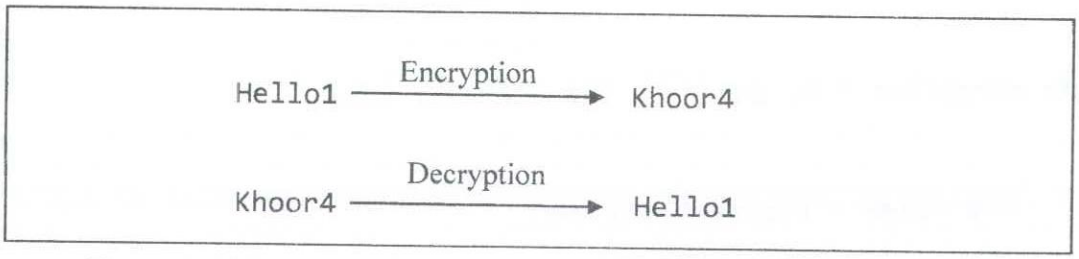
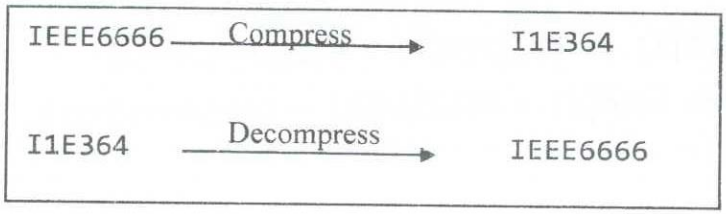


Figure Q5.b): Sample inputs and outputs for encryption and decryption operations of Caesar's cipher



FigureQ5.c): Sample inputs and outputs of data compression and decompression algorithms