ECE242 Data Structures and Algorithms

Fall 2008

Final Examination

(120 Minutes, closed book)

Name: ___SOLUTION_SET_______

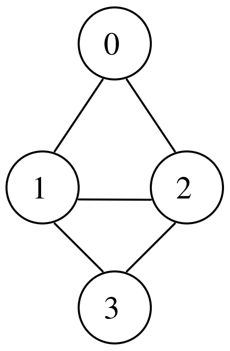
Student ID: ________________________

Question	**Score**
1 (20)	
2 (20)	
3 (08)	
4 (11)	
5 (15)	
6 (11)	
7 (05)	
8 (10)	

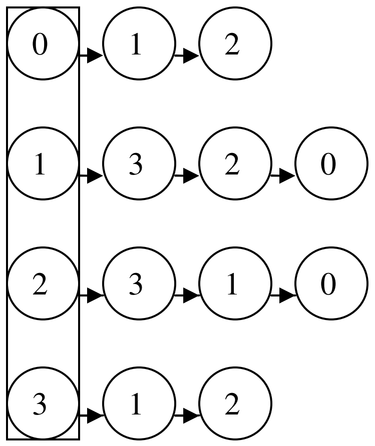
NOTE: Any questions on writing code must be answered in Java using Data Structures topics covered in the lectures.

1. [20 pts] The goal of this question is to write a code to convert a given *adjacency list* to an *adjacency matrix* for a graph as illustrated in the following figure.

Graph



Adjacency List



Adjacency Matrix

A[i][j]	**0**	**1**	**2**	**3**
0	0	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	0

Consider an undirected graph consisting of N vertices that are named 0, 1, 2,…, (N-1)

a) **[7 pts]** Create a class, **AdjList**. Write a method called **populateAdjList()** that reads in information from the keyboard using Scanner for each vertex and the vertices which are linked to the vertex. The information appears in the following input format *vertex neighbor1 neighbor2* … (e.g. "0 1 2", for vertex 0). One line is input per vertex. Before the vertex information is input, the user indicates the total number of vertices. This information is stored in an adjacency list. *Hint: You could use an array of ArrayList's to store the adjacency list.*

```
public class AdjList {
      private final int DEFAULT_CAPACITY = 20;
      Vertex adj_list[];
      AdjList(){
            // Array of objects storing the list of vertices linked to
the vertex
            adj_list = new Vertex[DEFAULT_CAPACITY];
      }

      AdjList(int size){
            adj_list = new Vertex[size];
      }


      /*
       * Taking user inputs fill out the adjacency list structure
       */
      public void populateAdjList(){
            int i=0;
            Scanner scan = new Scanner(System.in);
            for (i=0; i<adj_list.length; i++){
                  adj_list[i]=new Vertex();
                  System.out.println("Enter the number of vertices
linked to vertex"+i);
                  int count = scan.nextInt();
                  System.out.println("Enter these vertices one on a
line...");

                  for (int j=0; j<count; j++){
                        int temp = scan.nextInt();
                        adj_list[i].add(temp);
                  }
            }
      }//populateAdjList

      public void viewAdjList(){
            for (int i=0; i<adj_list.length; i++){
                 System.out.println("Elements linked to vertex"+i+":");
                  for(int j=0; j<adj_list[i].size(); j++)
                        System.out.println(adj_list[i].get(j));
            }
      }


      public Vertex[] getAdjList(){
            return adj_list;
      }
}
```

b) **[10 pts]** Create a class, **List2MatrixC**, and write a method, **list2matrix,** that takes *adj_list* (created in part **a**) as one of the inputs and returns a 2D array called **adj_matrix** that stores the adjacency matrix information for the graph described by *adj_list*. Also print out the information in the adjacency matrix at the end of **list2matrix** method.

```
public class AdjMatrix {
	int matrix[][];
	AdjMatrix(int num_vertices){
		matrix = new int[num_vertices][num_vertices];
	}

	public void setEdge(int vertex1, int vertex2){
		matrix[vertex1][vertex2]=1;
		matrix[vertex2][vertex1]=1;//Undirected graph
	}

	public void viewAdjMat(){
		int num_ver = matrix.length;
		for (int i=0; i<num_ver;i++){
			for (int j=0; j<num_ver;j++)
				System.out.print(matrix[i][j]+"\t");
			System.out.print("\n");
		}
	}

}


/*
 * This class describes a method that converts a given adjacency list
into
 * an adjacency matrix. Assume that you are given an adjacency list
representation
 * of a graph having N vertices and M edges
 */

public class List2MatrixC {

	/*
	 * Declare two arrays, a normal one for storing all the vertices
in the
	 * graph and the other, an ArrayList, for storing the list of
vertices
	 * connected to each node.
	 */
	public void list2matrix(Vertex list[], int num_ver, AdjMatrix
matrix){
		for (int i=0; i<4; i++){
			for (int j=0; j<list[i].size();j++){
```

```
                        matrix.setEdge(i,(Integer)list[i].get(j));
                  }
            }
      }

      public static void main(String[] args) {
            //
            List2MatrixC obj = new List2MatrixC();
            AdjMatrix matrix = new AdjMatrix(4);
            AdjList list = new AdjList(4);
            list.populateAdjList();
            list.viewAdjList();

            obj.list2matrix(list.getAdjList(), 4, matrix);
            matrix.viewAdjMat();
      }

}
```

c) **[3 pts]** Analyze the running time of the *list2matrix* method. Be sure to include a description of how you determined the running time.

Solution:

O(n^2)

2. [20 pts]

a) **[4 pts]** In your own words, how can a stack be implemented using two queues? *You are free to draw figures.*

Solution:

Use enqueue() method of Q1 for push() operation. For pop() operation, first dequeue() all but one from Q1 and enqueue() them into Q2, then dequeue() the last element from Q1 and return. Before returning dequeue() all from Q2 and enqueue() into Q1

b) [5 pts] In your own words, how can a queue be implemented using two stacks? *You are free to draw figures.*

Solution:

Use push() method of S1 for enqueue() operation. For dequeue() operation, first pop() all but one from S1 and push() them into S2, then pop() the last element from S1 and return. Before returning pop() all from S2 and push() into S1

c) [5 pts] Insert the following sequence of numbers into a heap structure. Show the necessary steps required to create the heap: 7,6,14,3,8

Solution:

```
            3
       /         \
      6           14
   /     \
  7       8
```

d) [3 pts] Add a new node 5 to the heap created in part c). *Show all steps for credit.*

Solution:

```
            3
       /         \
      6           5
   /     \     /
  7       8    14
```

e) **[3 pts]** Delete node 5 from the heap constructed in part d). *Show all steps for credit.*

Solution:

```
              6
         /         \
        7           14
      /
    8
```

3. [8 pt] Consider the weighted, undirected graph $G = \langle V, E \rangle$ as shown in Figure 1.

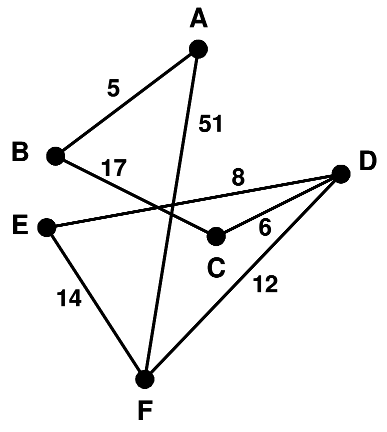


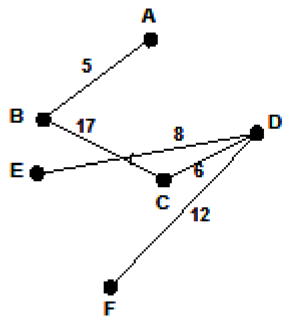
Figure 1

Apply Dijkstra's shortest path algorithm to find a shortest path tree starting with the source vertex A. *Show all steps for full credit. No Java code necessary.*

Node	From node A to other nodes				
B	**5**	**5**	**5**	**5**	**5**
C	inf	**22**	**22**	**22**	**22**
D	inf	inf	**28**	**28**	**28**

E	inf	inf	Inf	**36**	**36**
F	51	51	51	40	**40**
Best	B	C	D	E	F

The resulting shortest path is as follows:



4. [11 pt]

(a) [3 pt] In your own words, define a minimum spanning tree for a given weighted graph $G = \langle V, E \rangle$?

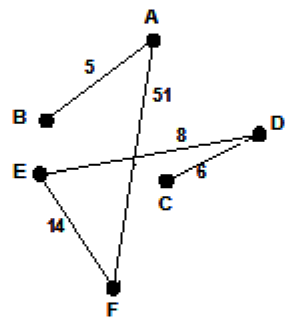
Solution: Minimum spanning tree for a given graph is a tree that contains all the vertices that are present in the graph and these vertices are interconnected by minimum cost connections available in the original graph.

(b) [8 pt] Apply Kruskal's or Prim's algorithm (indicate which) to construct a Minimum Spanning Tree starting with vertex C of the weighted undirected graph $G = \langle V, E \rangle$ shown in Figure 1 in question 3 above. *Show all steps for full credit. No Java code necessary.*

Solution:

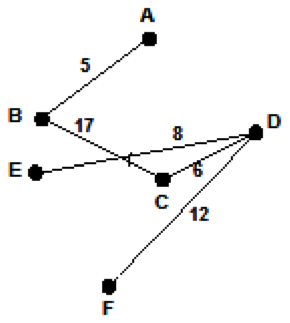
Prim's algorithm applied to Figure 1:

C – D – E – F – A – B



Kruskal's algorithm applied to Figure 1:

A – B – C – D –E - F


A
5
B
17
8
D
E
C
6
12
F

5. [15 pts] Write a Java method **mergeFactorial**() that calculates the factorial of a number **n** using a *divide and conquer* approach using *recursion*. During each call to **mergeFactorial()** the number of values to be multiplied is split in half and *two calls* to **mergeFactorial()** are performed. *Hint: Think similar to Merge Sort, which has both divide and conquer and recursion.* **Implementations of recursive factorial that match the code given in class (e.g. one recursive call) will receive no credit.**

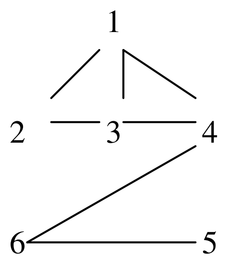
```java
/*
 * Using the divide and conquer approach, calculate factorial of the
first 10
 * natural numbers. Use recursion.
 */
public class MergeFactorial {

      public long printFactorial(){
            int sequence[] = new int[10];
            // Initialize array with first 10 natural numbers
            for (int i=0; i<10; i++){
                  sequence[i]=i+1;
            }
            return mergeFactorial(sequence, 0, 9);
      }

      static long mergeFactorial(int data[], int start, int end){

        if( start < end)
        {
            int mid = (start+end)/2;
            long fact1=0, fact2=0;
            fact1=mergeFactorial(data, start, mid);
            fact2=mergeFactorial(data, mid+1, end);
            return (fact1*fact2);
        }
        return (data[start]);
      }


      public static void main(String[] args) {
            // TODO Auto-generated method stub
            MergeFactorial m = new MergeFactorial();
            System.out.println("Factorial= "+m.printFactorial());
      }

}
```

6. [11 pts] Let G be an undirected graph whose vertices are the integers 1 through 6 and let the adjacent vertices of each vertex be given by the table below:

Vertex	**Adjacent Vertices**
1	2,3,4
2	1,3,4
3	1,2,4
4	1,2,3,6
5	6
6	4,5

Assume that in the traversal of G, the adjacent vertices of a given vertex are returned in the same order as they are listed in the above table.

a) **[3 pts]** Draw G.



b) **[4 pts]** Perform depth first search (DFS) traversal on the graph, G above starting at vertex 1 and list the vertices in that order.

Solution:

1-2-3-4-6-5

c) **[4 pts]** Perform breadth first search (BFS) traversal on the graph, G above starting at vertex 1 and list the vertices in that order.

Solution:

1-2-3-4-6-5

7. [5 pts] Write a method **remDNode***(DList* d*, int* val*)* that traverses through a doubly linked list and deletes the node with the value *val*. You may use any method of the DList class that you deem necessary.

You can assume that all the values in the DList are unique and none are repeated.

Class definition for **DNode**:

```
public class DNode {
      public DNode next, prev;
      public Object value;

      public DNode(DNode p, DNode n, Object v){
            prev = p;
            next = n;
            value = v;
      }
}
```

DList class and its method declarations:

```
public class DList{

    public DNode head, tail;
    public int count;

    public void Insert(DNode p, Object item);
    Object remove(DNode p);
    public void printAll();
    public DNode search(Object name);
}
```

Solution:

```
/*
```

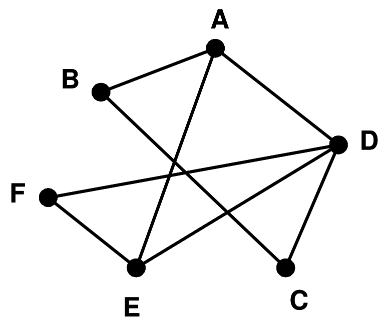
```
 * Write a method remDNode(DList d, int val) that traverses through a
doubly
 * linked list and deletes the node with the value val. You may use any
other
 * method of the DList class that you deem necessary.
 * You can assume that all the values in the DList are unique and none
are repeated.
 */
public class RemoveDNode {

      public void remDNode(DList d, int val){
            DNode currNode = d.returnHead();
            while (currNode.next!=d.returnTail()){
                  currNode = currNode.next;
                  if ((Integer)currNode.value==val)
                        d.remove(currNode);
                  break;
            }
      }

      public static void main(String[] args) {
            // TODO Auto-generated method stub

      }

}
```

8. [10 pts] Write a method **InverseInsertionSort**(*int* input_array[]) that takes a sequence of unsorted integers present in the *input_array* as an argument and arranges them in ***<u>descending</u>*** order starting at index 0 in the array. **Note: Your method must make use of the Insertion sort algorithm.**

```
// Question 8: Inverse Insertion Sort

import java.util.Scanner;

public class InverseInsertionSort {
    static void InverseInsertionSort(int[] array) {
        int temp;
        int i, j;
        for( i=1; i<array.length; i++ )
        {
            temp = array[i];
            for( j=i; j>0 && temp>array[j-1]; j--)
                array[j] = array[j-1];

            array[j] = temp;
            int n;
            System.out.print("In sorting: " );
            for( n=0; n<array.length; n++ )
                System.out.print(" " + array[n]);
            System.out.println("");
        }
    }

    static void inputArray( int in[], int num ) {
        Scanner easy = new Scanner(System.in);
        int i;
        for( i=0; i<num; i++ )
        {
            System.out.print("Enter val[" + i + "]: ");
            in[i] = easy.nextInt();
        }
    }

    public static void main(String args[]) {
        Scanner easy = new Scanner(System.in);

        /* input the number of students */
        System.out.print("Input the number of integers: ");
        int number = easy.nextInt();

        int [] array = new int [number];

        inputArray( array, number );

        int n = 0;
        System.out.println("\n");
        System.out.print("Before sorting: " );
        for( n=0; n<number; n++ )
            System.out.print(" " + array[n]);
        System.out.println("");

        InverseInsertionSort(array);

        System.out.print("After sorting: " );
        for( n=0; n<number; n++ )
            System.out.print(" " + array[n]);
        System.out.println("");
  }

}
```

Extra credit [2 pts] During lecture Prof. Tessier mentioned several times that he worked as a software developer at which company while on sabbatical in 2005? __ALTERA____________

Extra credit [2 pts] Find an Eulerian path for the graph shown in the figure below.



Ans: A-B-C-D-A-E-D-F-E