University of Bahrain

College of Information Technology

Department of Computer Science

First Semester, 2017-2018

ITCS214 / ITCS215 (Data Structures)

#### Final Exam

Date: 07/01/2018 Time: 11:30 - 13:30

|  |
| --- |
|  |

**STUDENT NAME**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

**STUDENT ID #**

|  |
| --- |
|  |

**SECTION #**

NOTE: THERE ARE TEN **(10) PAGES** IN THIS TEST

ONLY ONE SOLUTION WILL BE CONSIDERED FOR EACH QUESTION

|  |  |  |  |
| --- | --- | --- | --- |
| QUESTION # | MARKS | | COMMENTS |
| 1 | 12 |  |  |
| 2 | 12 |  |  |
| 3 | 12 |  |  |
| 4 | 18 |  |  |
| 5 | 15 |  |  |
| 6 | 11 |  |  |
| TOTAL | 80 |  |  |

**Question 1 [12 Marks]**

Write a method **insertDeleteInCorrespondingIndex** to be included in the **KWLinkedList** class that accepts three parameters, **list1** and **list2** oftype **KWLinkedList** and **value** of type **E**. The method looks for nodes in **list1** having its data as **value** and if found inserts **value** in **list2** in the same location (index) as in **list1**.

After this, the method deletes all occurrences of **value** from **list1**.

Assume that **list1** and **list2** are not empty and **list2** is having at least the same number of nodes as **list1**. Write your method using **ListIterator**.

Example**: value = 30**

|  |
| --- |
| **Before method call**:  index: 0 1 2 3 4 5 6  **list1** : 40 20 **30** 50 25 **30** 10  **list2:** 15 5 25 3512 18 22 27  **After method call**:  index: 0 1 2 3 4 5 6  **list1**: 40 20 50 25 10  **list2:** 15 5 **30** 25 35 **30** 12 18 22 27 |
|  |

Method heading:

**public boolean insertDeleteInCorrespondingIndex (KWLinkedList<E> list1,**

**KWLinkedList<E> list2, E value)**

**Question 2 [12 Marks]**

Write a method called **retainMaxStack** in a class called **StackEx** that receives an object **st1** of type **ArrayStack** as parameter. The method finds the maximum of two consecutive elements of the stack **st1** and retains the maximum in the stack and deletes the smaller one from the stack **st1**.

Use common stack operations only such as **push**, **pop**, **peek** and **isEmpty**. You can create local objects of type **ArrayStack** in your method. Do not use array or any other data structure. Assume that the stack **st1** is not empty and has even number of elements.

Example:

|  |
| --- |
| **Before method call**:  top  **st1**: **25**, 20, 30, **50**, **70**, 60, 15, **35** |
|  |

|  |
| --- |
| **After method call:**  top  **st1**: **25**, **50**, **70**, **35** |
|  |
|  |

public class StackEx

{

public static void retainMaxStack(ArrayStack<Integer> st1)

{

ArrayStack<Integer> temp1= new ArrayStack <Integer> ();

Int x,y;

While(!st1.isEmpty())

{

X=st1.pop();

if(!st1.isEmpty())

{

Y= st1.pop();

}

If(x>y)

Temp1.push(x);

Else

Temp1.push(y);

}

While(!temp1.isEmty())

St1.push(temp1.pop());

}

}

**Question 3 [12 Marks]**

Write a method **rearrangeQueue** to be included in class **QueueEx**. The method accepts two parameters, **q1** of type **ArrayQueue** and **item** of type E. Assume that **q1** contains integer values. The method rearranges the elements of **q1** in such a way that it puts all values greater than or equal to **item** at the front of **q1** in the same order as in the original queue. If any reorder is done, the function returns true. If **q1** is empty or no reorder is done, the function returns false. Assume that class **ArrayQueue** is available for use.

**Example :**

**item = 50**

|  |
| --- |
| **After method call:**  **q1**: **56 78 90 103** 24 6 44  front rear |
|  |
|  |

**Before method call**:

**q1**: 24 56 78 90 6 44 103

front rear

|  |
| --- |
|  |
|  |

public class QueueEx

{

public static boolean rearrangeQueue(ArrayQueue<Integer> q1,

int item)

{

Boolean flag=false;

ArrayQueue<Integer>temp1=new ArrayQueue<Integer>();

ArrayQueue<Integer>temp2=new ArrayQueue<Integer>();

Int x;

If(q1.isEmpty())

Return false;

While(!q1.isEmpty())

{

X=q1.poll();

If(x.compareTo(item)>0&&x.compareTo(item)=0)

{

Temp1.offer(x);

Flag=true;

}

Else

{

Temp2.offer(x);

Flag=true;

}

While(!temp1.isEmpty())

Q1.offer(temp1.poll());

While(!temp2.isEmpty())

Q1.offer(temp2.poll());

If(flag)

Return true

Else

Return false

}**Question 4 [6 + 12 Marks]**

**(A) [6 Marks]** Write a recursive private method called **countOneChild** to be included in class **BinaryTree** as discussed in the lectures. The method returns the number of nodes in a binary tree having exactly one child node, i.e., either non-empty left sub-tree and empty right sub-tree, **or** empty left sub-tree and non-empty right sub-tree.

This method is called from a public method **treeCountOne**, given as follows:

public int treeCountOne( )

{

return countOneChild(root );

}

Method heading:

**private int countOneChild(Node<E> node)**

**{**

**if(node==null)**

**return 0;**

**if((node.left==null&&node.right!=null) ||(node.left!=null&&node.right==null))**

**return 1+ countOneChild(node.left)+ countOneChild(node.right);**

**else**

**return countOneChild(node.left)+ countOneChild(node.right);**

**}**

**(B) Consider the following binary search tree:**

1. **[2 Marks]** List all the **leaf nodes** of this binary tree.

**20-42-55-65-75**

1. **[5 Marks]** Find the sequence of nodes, if the binary tree is traversed in **inorder traversal.**
2. **[2 Marks]** Redraw the above binary search tree after inserting the nodes with keys **35** and **52** consecutively, in the original binary search tree.
3. **[3 Marks]** Redraw the above binary search tree after deleting the nodes with keys **50** and **80** consecutively, from the original binary search tree.

**Question 5 [5 + 5 + 5 Marks]**

1. **[5 Marks]** For the following graph, find the **adjacency list** representation of the graph:
2. **[5 Marks]** For the following graph, find the sequence of vertices in the graph, if the graph is traversed using **Depth-First Traversal** algorithm. Assume vertex 0 as the starting vertex.

**3**

**0**

**1**

**2**

**5**

**4**

**8**

**7**

**6**

**(C)[5 Marks]** For the graph shown in **(B)**, find the sequence of vertices in the graph, if the graph is traversed using **Breadth-FirstTraversal** algorithm. Assume vertex 0 as the starting vertex.

**Question 6 [7 + 4 Marks]**

Given the following input keys: 24, 23, 30, 108, 35, 42, 54,

hash function h(X) = X mod 12, HTSize = 12 and bucket size = 1.

1. Obtain the resulting hash table when open addressing with **linear probing** is used to resolve collisions.
2. Obtain the resulting hash table when **chaining** is used to resolve collisions.