CSE214

EXAMS

PRACTICE QUESTIONS – MIDTERM 2

THIS REVIEW PAGE DOES NOT IMPLY THAT THE ACTUAL MIDTERM QUESTIONS WILL BE OF THE SAME FORMAT.

1. Complete the following method that reverses the elements of an integer queue recursively. By reversing the queue, we mean that the first element becomes the last, the 2nd element becomes the 2nd-to-last element, … and the last element becomes the first element. You may assume the IntQueue class has the standard methods isEmpty,enqueue and dequeue.

```
   public static void reverse(IntQueue Q) {
         int temp;
         if ____Q.isEmpty()________________________   // stopping
     case
             return;
         else {       // recursive case: must be exactly 3
     statements
               ____temp = Q.dequeue();_____________;
               ____reverse(Q);_____________________________;
               ____Q.enqueue(temp);______________________________;
         }
     }
```

2. Consider the following recursive method on an integer array data with n integers:

```
    public static int mystery(int[] data, int n) {
     int sum;
     if (n <= 0) return 0;
     else {
        if (data[n-1] % 2 == 0)
           sum = mystery(data, n-1) + 1;
        else
            sum = mystery(data, n-1);
        return sum;
     }
    }
```

(a) What is the final return value of this method if it called with the following parameters initially: data = {2, 2, 3, 3, 3, 4, 4, 4, 4}, n = 9? 6

(b) Assuming an int is 2 bytes and a memory reference (address) is 4 bytes, how many bytes are required for an activation record for this method?

return address: 4 bytes

parameters: 4 B (int *, memory reference) + 2 B (int n) = 6 B
local varibles: 2B (int sum);
so one activation record costs 12 bytes. We'll need a stack of 10 activations records, and the size of this stack would be 12 x 10 = 120 bytes.

USE THE FOLLOWING BINARY SEARCH TREE FOR QUESTIONS 3 AND 4:

```
       M
     /   \
   J       Q
 /        /  \
E        N     X
 \            /
  H         T
 /            \
G              U
```

3.

(a) Write the inorder traversal of the binary tree above. EGHJMNQTUX

(b) Write the preorder traversal of the binary tree above. MJEHGQNXTU

(c) Write the postorder traversal of the binary tree above. GHEJNUTXQM

(d) For any binary search tree with *n* letter nodes, what is the maximum number of nodes that have to be searched to find any letter? n

4.

(a) What is the depth of the binary tree above? 4

(b) What is the maximum number of nodes that can be inserted to the binary tree above without increasing its depth? 31 - 10 = 21

(c) Draw the binary search tree after the node containing M is removed

```
        N
       / \
     J     Q
    /       \
   E          X
    \        /
     H      T
    /        \
   G          U
```

5. This tree is a 2-3-4 tree:

```
  [34   56   78]

     /    |    |    \

[12 23] [45]  [67]  [89]
```

(a) Show the 2-3-4 tree after the integer 84 is inserted into the 2-3-4 tree.

```
  [34   56   78]

     /    |    |    \

[12 23] [45]  [67]  [84 89]
```

(b) Convert the original 2-3-4 tree into an equivalent red-black tree. Label red nodes with the letter R and black nodes with the letter B.

```
  [34   56   78]

     /    |    |    \

[12 23] [45]  [67]  [89]
      [56]
     /    \
  [34]     [78]

     /    |    |    \

[12 23] [45]  [67]  [89]

      [56]
     /    \
  [34]     [78]

     /    |    |    \

  [23] [45]  [67]  [89]
  /
[12]
```

6. A heap is stored using an array as follows:

INDEX	0	1	2	3	4	5	6
CONTENTS	83	45	62	29	14	38	11

(a) Show the contents of the array after the value 37 is inserted into the heap.

INDEX	0	1	2	3	4	5	6	7

CONTENTS	83	45	62	29	14	38	11	37

INDEX	0	1	2	3	4	5	6	7
CONTENTS	83	45	62	37	14	38	11	29

(b) Show the contents of the array after the remove operation is performed on the original heap.

INDEX	0	1	2	3	4	5	6
CONTENTS		45	62	29	14	38	11

INDEX	0	1	2	3	4	5
CONTENTS	11	45	62	29	14	38

INDEX	0	1	2	3	4	5
CONTENTS	62	45	11	29	14	38

INDEX	0	1	2	3	4	5
CONTENTS	62	45	38	29	14	11

7. An array is sorted in non-increasing order and contains 64 data values.

(a) If sequential search is used, what is the maximum number of comparisons that is needed to search for a target in this array? 64

(b) If binary search is used, what is the maximum number of comparisons that is needed to search for a target in this array? 7

(c) TRUE OR FALSE: If the target occurs more than once in the array, binary search will find the target with the lowest index. (Explain.) False. Suppose we have an array 2 1 1 1 0, find(1) will return the middle 1, which is neither the lowest indexed one, nor the highest indexed one.

(d) If the target is in position 0 of the array, which search technique would find the data faster? Why? sequential search. because we only need O(1) time to get to the first element.

8. A hash table is created to store integer keys using a hash function h(k) = k mod 13. The current state of the hash table is shown below. All keys were inserted without any collisions.

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12
KEY		14		29		83	45				62	11	38
HAS_BEEN_USED	F	T	F	T	F	T	T	F	F	F	T	T	T

(a) At what position will the key 23 be stored in the hash table using h(k) above if linear probing is used to resolve collisions? position 0.

(b) At what position will the key 23 be stored in the original hash table if double hashing is used to resolve collisions,

assuming $h_1(k) = h(k)$ and $h_2(k) = 1 + (k \bmod 11)$? 7

(c) What is the load factor of the original table? 7/13

(d) Assuming linear probing is used and no removals have occurred, what is the approximate average number of table elements examined in a successful search of the original hash table? *(Express your answer as a simplified fraction.)* 0.5x(1+1/(1-7/13))

9. A hash table is defined using an array of 100 `IntNode` references, so that collisions can be handled by using chaining, as follows:

```
public class Table {
 private int manyItems;
 private IntNode[] keys;
```

```
      public Table() {
           keys = new IntNode[100];
           manyItems = 0;
      }
     private int hash(int key) {
          return key % 100;
      }
     // other Table methods
}
```

Write Java code for the following `Table` methods below. You may assume that `IntNode` is a class that defines a singly-linked integer node with the methods `getData`, `setData`, `getLink`, `setLink`, and a default constructor.

(a) `public void put(int key)`

Inserts the key into the appropriate “chain” of the hash table. You may assume that the key is not a duplicate.

```
{ int k;
   k = hash(key);
   IntNode newNode = new IntNode(key);
   if (keys[k] == null) keys[k] = newNode;
   else {
      IntNode node = keys[k];
      while (node.getLink()!=null)
         node = node.getLink();
      node.setLink(newNode);
   }
}
```

(b) `public boolean containsKey(int key)`

Returns true if the key is in the hash table, or false otherwise.

```
{   IntNode node;
      node = keys[hash(key)];
      while (node != null) {
         if (node.getData() == key) return true;
         node = node.getLink();
      }
      return false;
}
```

10. Let the class BTNode represent a node of a binary tree that stores an integer, defined as follows:

```
public class BTNode {
     private int data;
     private BTNode left, right;
     // BTNode methods
}
```

Write Java code for the following recursive BTNode methods.

(a) `public void inorder()`

Prints the contents of the binary tree rooted at this node using an inorder traversal.

```
{   if (left != null) left.inorder();
      System.out.println(data);
      if (right != null) right.inorder();
}
```

(b) `public int count()`

Returns the number of leaves in the binary tree rooted at this node.

```
{  if  ((left == null) && (right == null)) return 1;
   int sum = 0;
   if (left != null) sum = sum + left.count();
   if (right != null) sum = sum + right.count();
   return sum;
}
```