# CS-M12 2015

# Software Concepts and Efficiency

(Answer **all** questions, Total 50 marks)

## **Question 1:** Algorithm and Complexity

Suppose `arr` is a given array of positive integers of length `N > 5`. Each element of `arr` is an integer between `1` and `N+1`. All the elements of `arr` are different. Consider the following function:

```
public static int something(int arr[])
{
      for (int j = 1; j <= arr.length+1; j++) {
            boolean found = false;
            for (int i = 0; i < arr.length; i++) {
                  if (arr[i] == j) {
                        found = true;
                        break;
                  }
            }
            if (!found) {
                  System.out.println(j);
                  return;
            }
      }
}
```

**a)** Describe what task the function performs in just one sentence. (Do not explain how the function executes the task.) Hint: use the following input array to understand the function: {5,6,2,4,1,7}

**[5 Marks]**

**b)** What is the time complexity of the function in Big-O notation?
Detail how you arrive at the time complexity.

**[5 Marks]**

**c)** Write a snippet of Java code or pseudo-code which performs the same task but which is ***asymptotically faster*** (not a constant factor faster) in time complexity. Analyze the complexity of your algorithm in Big-O notation.

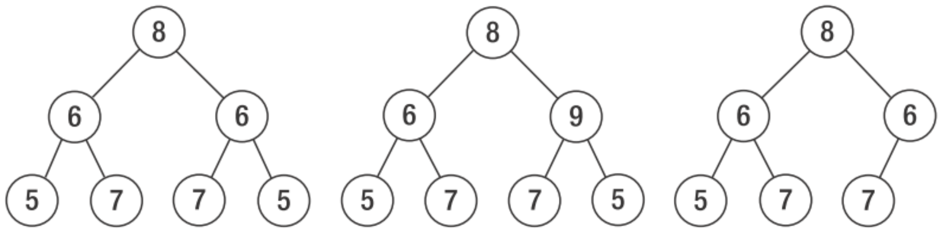
**[10 Marks]**

## **Question 2:** Recursion and Binary Tree

**a)** Provide the definition of a binary tree. **[3 Marks]**

**b**) Discuss how to use top-down approach to develop an algorithm using recursion.
Discuss which four important questions to answer, and how to develop a recursion algorithm using the above four questions - illustrate with one simple example. **[5 Marks]**

**c**) Given a pointer to the root of a binary tree, write a function to verify whether a binary tree is symmetrical. A tree is symmetrical if its mirrored image looks the same as the tree itself.



For example, above are three binary trees. The first tree is symmetrical but the rest are not.

You should not use any library functions but can assume that the key at a tree node can be obtained using the `t.data` property and the children of a node `t` using the `t.left` and `t.right` properties. If there is no child, `t.left` and `t.right` store `null` value.

```
public class Node {
      public Node left;
      public Node right;
      public int data;
      Node(int newData) {
            left = null;
            right = null;
            data = newData;
      }
}
```

Use the following template:

```
public boolean isSym() {
      return isSymSubTree(root.left, root.right);
}

public boolean isSymSubTree (Node left, Node right) {
      // some code here
      // ...
      // some code here
}
```

**[7 marks]**

**Question 3:** Hashtable and Anagrams

**a)** Explain what a Hashtable is, and why a good hash function is important.

**[5 marks]**

**b)** Two strings (unbounded sequences of letters) are anagrams of each other if the letters from one string can be rearranged to form the other string. For examples, “silent” and “listen”, “nameless” and “salesmen”, “treason” and “senator” are all pairs of anagrams.

Implement a java program that runs in O(n) time to verify whether two strings (each contains n letters) are a pair of anagrams. Analyse the Big-O complexity of your algorithm.

Hint: two words are anagrams if all characters in both strings occur same number of times.

```
> java StringAnagrams silences salesmen
Not anagrams!
> java StringAnagrams nameless salesmen
Anagrams!
```

You can assume the following API is provided:

| | |
|---|---|
| `HashMap<Character, Integer> times` | - a java hashtable |
| `times.containsKey(ch)` | - to query if a character (ch) exists in a hashtable |
| `times.get(ch)` | - to query the associated frequency of ch |
| `times.put(ch, freq)` | - to update the associated frequency of ch to freq |

Use the following template:

```
import java.util.HashMap;
public class StringAnagrams {
     public static void main (String[] args) {
          String str1 = args[0];
          String str2 = args[1];
          if (str1.length() != str2.length()) {
               System.out.println("Not anagrams!");
               return;
          }
          // ... Code here...
          // ...
          // ... Code here...
     }
}
```

**[10 marks]**

---- END ----