CS/ENGRD2110: Prelim 1
SOLUTION

10th of March, 2011

```
NAME : __________________________________________________

NETID: _____________
```

- The exam is **closed book and closed notes**. Do not begin until instructed. You have **90 minutes**. Good luck!
- Start by writing your name and Cornell netid on top! There are **10 numbered pages**. Check now that you have all the pages.
- Web, email, etc. may not be used. Calculator with programming capabilities are not permitted. This exam is **individual work**.
- We have **scrap paper** available, so you if you are the kind of programmer who does a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam, just so that we can make sense of what you handed in!
- Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to **fit your answers easily into the space we provided**. Answers that are not concise might not receive full points.
- In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to make the problem more difficult. This does not mean that its good style.

```
POINTS:

Classes, Interfaces, Types, and Stuff        ______ / 19

Recursion                                    ______ / 14

Trees                                        ______ / 22

Asymptotic Complexity                        ______ / 18

Sorting                                      ______ / 13

Lists                                        ______ / 14

                                             ===========

Total                                        ______ /100
```

1 Classes, Interfaces, and Types

1. Answer the following questions with either true or false. No explanation necessary.

   7 pts.

   - Both interfaces and classes define the type hierarchy in Java.
   - Every type in Java has exactly one or zero supertypes.
   - Every type in Java has exactly one or zero subtypes.
   - A cast changes the dynamic type of an object.
   - Upcasts can produce runtime errors.
   - Abstract classes that contain no implementations provide exactly the same functionality as interfaces.
   - The static type of an argument to an overloaded method determines which of the methods is selected.

   SOLUTION:
   yes,no,no,no,no,no,yes
   END SOLUTION

2. Given the interface definitions from above, what are the methods that need to be present in the following class `CA`?

   3 pts.

```
interface IA {
        public IA mA(int a);
}

interface IB extends IA {
        public int mB(int a);
        public Object mBB(int a);
}

interface IC extends IA {
        public IA mC(int a);
}

class CA implements IB {
...
}
```

   SOLUTION:
   mA, mB, mBB
   END SOLUTION

3. Given the interface definitions from above, does the following class definition contain any errors? If yes, what are the errors?

   2 pts.

```
class CB implements IA {
        public IC mA(int a) { return this; }
}
```

SOLUTION:
Yes, it returns an object of type CC, which is not a subtype of IC.
END SOLUTION

4. What output does the following program produce?

4 pts.

```
class CC {
        int x=1;
        public float mD(int a) { return a; }
}

class CD extends CC {
        Integer x=2;
        public float mD(int a) { return x*a;}
}

public class TypeMania {
        public static void main(String[] args) {
                CD x = new CD();
                System.out.println(x.mD(1));
                CC y = (CC)x;
                System.out.println(y.mD(1));
                y.x=3;
                System.out.println(y.mD(1));
        }
}
```

SOLUTION:
2 2 2
END SOLUTION

5. The following program does not compile correctly. Please identify the problem in one sentence.

3 pts.

```
public class Test {
        public static void main(String[] args) {
                System.out.println(max(3,5));
        }

        public int max(int a, int b) {
                if(a > b) {
                        return(a);
                }
```

```
                    else {
                              return(b);
                    }
          }
}
```

SOLUTION:
The method “max” has to be declared as static, since it is called from the staic method main().
END SOLUTION

2 Recursion

1. What is the output of rec(14,4)?

   8 pts.

```
public static void rec(int a, int b) {
        if (a == 0 && b == 0) return;
        if (a >= b) {
                System.out.print (a + " ");
                rec((a/2)-1, b);
                System.out.print (b + " ");
        }
        else {
                System.out.print (a + " ");
                rec(a, b-2);
                System.out.print (b + " ");
        }
        return;
}
```

   SOLUTION:
   14 6 2 2 0 2 2 4 4 4
   END SOLUTION

2. Write a method with the signature `public int max(int[] array)` that returns the maximum value of all the elements in the array. You can assume that the array contains at least one element. Do this without using any loops but use recursion instead.

   Hint: Feel free to create a helper methods.

   6 pts.

   SOLUTION:

   END SOLUTION

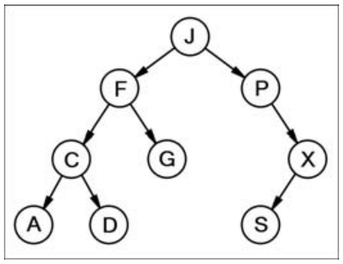
```
        public static int max(int[] array) {
                return max(array,array.length-1);
        }

        public static int max(int[] array, int end) {
                if(end == 0) return(array[0]);
                else {
                        int maxInPrefix = max(array, end-1);
                        if(maxInPrefix < array[end])
                                return(array[end]);
                        else {
                                return(maxInPrefix);
                        }
                }
        }
```

3 Trees

1. Give the preorder, inorder, and postorder traversal of the following tree.

9 pts.



SOLUTION:
Preorder: JFCADGPXS
Inorder: ACDFGJPSX
Postorder: ADCGFSXPJ
END SOLUTION

2. Show the binary search tree (BST) after inserting the following sequence of elements. The lexicographical ordering should be used to construct the tree.

7 pts.

M, G, A, Q, N, B, Z

SOLUTION:
Here is the pre-order traversal of the tree after the sequence of insertions: M(G(A(null,B),null),Q(N,Z))
END SOLUTION

3. Derive an efficient method for deleting a given element (denoted by X) from a BST, so that the resulting tree after deletion is again a BST. There are three cases to be considered depending on where in the tree the element is located. Explain in English (or through a diagram showing how the tree is modified) what needs to be done in each case. Do NOT write Java code.

6 pts.

SOLUTION:

- X is a leaf (has no children): In this case, simply delete the node by setting the pointer to it to NULL.
- X has exactly one child: In this case, make the child of X the child of X's parent.
- X has two children: In this case, locate the maximum node (call it M) in the left subtree of X and replace X's value with M's value. Then delete M, which is either a leaf or has only one child. Alternatively, one can also replace X with the minimum node in the right subtree of X.

END SOLUTION

4 Asymptotic Complexity

1. Answer the following questions with either true or false. No explanation necessary.

   6 pts.

   - Binary search in a sorted array with n elements has average-case time complexity $O(log(n))$.
   - Binary search in a sorted array with n elements has worst-case time complexity $O(log(n))$.
   - Finding an element in a BST with n elements has worst-case time complexity $O(log(n))$.
   - There might be an algorithm for sorting an array of integers with n elements in time $O(n)$.
   - Inserting n elements into an unsorted linked list can be done in worst-case time $O(n)$.
   - Checking whether an element is contained in a general tree can always be done in average-case time $O(h)$, where h is the height of the tree.

   SOLUTION:
   true, true, false, false, true, false
   END SOLUTION

2. Give the mathematical definition of "$f(n)$ is $O(2^n)$".

   5 pts.

   SOLUTION:
   There exists constants c and N such that for all $n \geq N$ it holds that $f(n) \leq c * 2^n$.
   END SOLUTION

3. Prove that $(3 * n^2 + 15 * n * log(n) - n)$ is $O(n^2)$.

   4 pts.

   SOLUTION:
   $(3 * n^2 + 15 * n * log(n) - n) \leq n * (3 * n + 15 * log(n) - 1) \leq n * (3 * n + 15 * log(n)) \leq n * (3 * n + 15 * n) = 18 * n^2$
   With c=18 and N=1 it therefore holds that $(3 * n^2 + 15 * n * log(n) - n) \leq 18 * n^2 \leq c * n^2$.
   END SOLUTION

4. Prove that 3^n is NOT $O(2^n)$.

   3 pts.

   SOLUTION:
   If 3^n were $O(2^n)$, then there must be some c so that for any $n \geq N$ it holds that $3^n \leq c * 2^n$.
   Solve for c: $3^n \leq c * 2^n \Leftrightarrow (\frac{3}{2})^n \leq c$.
   Since the needed c grows with n, there is always some large n that violates $3^n \leq c * 2^n$ for any fixed c.
   END SOLUTION

5 Sorting

1. The following algorithm (in abbreviated pseudo code) sorts an array of n integers. In Big-O notation, what are the numbers of pair-wise comparisons that this algorithm makes in the best case, the expected case, and in the worst case? Give the tightest Big-O statement you can make.

   6 pts.

```
void lameSort(int[] arr) {
  for (int i = 1; i < arr.length; ++i) {
    for(int j = 0; j < arr.length - i; ++j) {
      if (arr[j] > arr[j+1]) {
        'swap arr[j] and arr[j+1])'
      }
    }
  }
}
```

   SOLUTION:
   Best-case: $O(n^2)$
   Expected-case: $O(n^2)$
   Worst-case: $O(n^2)$
   END SOLUTION

2. Explain (in English) how you could improve the sorting algorithm from above so that it has best-case time complexity of $O(n)$? What is this best case?

   5 pts.

   SOLUTION:
   Keep track whether there were any swaps in the previous iteration if the inner loop. If there were none, the array must be sorted and the algorithm can terminate. The new algorithm will take only a single pass through the array in case it is already sorted.
   END SOLUTION

3. Yes or No: Is the sorting algorithm from above stable? No explanation necessary.

   2 pts.

   SOLUTION:
   Yes. The algorithm only swaps two elements if one is strictly greater than the other. That means that two equal elements are never swapped and therefore never change their order.
   END SOLUTION

6 Lists

1. In the following code, what does the method ”mystery” do? Also explain briefly when such a method would be useful?

   8 pts.

```
class ListNode {
        ListNode next;
        String value;
}

public class List {
        ListNode head;

        boolean mystery(String s){
                if(head == null) return false;
                if(head.value.equals(s)) return true;
                ListNode p = head;
                while(p.next != null){
                        if(p.next.value.equals(s)){
                                ListNode q = p.next.next;
                                p.next.next = head;
                                head = p.next;
                                p.next = q;
                                return true;
                        }
                        else
                                p = p.next;
                }
                return false;
        }

        //other list methods not shown
}
```

   SOLUTION:
   mystery returns false if s is not in the list. If it is, mystery moves it to the front and returns true. The method would be useful if we call it many times and some of the strings we pass to it are searched for more frequently than the others. Then those frequently searched for elements would be found faster in the list. (It can be shown that moving searched items to then front is at least half as good as the best static list).
   END SOLUTION

2. Add a method `deleteMax()` to the class `List` from above that finds the maximal element in the list and deletes all its occurences from the list.

   6 pts.

   Reminder: A string `s` is greater than a string `t` if `s.compareTo(String t) > 0`.

   SOLUTION:

END SOLUTION

```
    private String findMax() {
        ListNode maxN = head;
        for(ListNode n = head; n != null; n=n.next) {
                if(maxN.value.compareTo(n.value) < 0) {
                        maxN = n;
                }
        }
        return maxN.value;
    }

    private void delete(String d) {
        // remove d if at front of list
        while(head != null && d.equals(head.value)) {
                head = head.next;
        }
        // delete d if later in list
        ListNode n = head;
        while(n != null && n.next != null) {
                if(d.equals(n.next.value)) {
                        n.next = n.next.next;
                }
                else {
                        n=n.next;
                }
        }
    }

    public void deleteMax() {
        if(head != null) {
                delete(findMax());
        }
    }
```

Alternatively, you can reuse “mystery” for the deletion. Call mystery with the maximal element and if it returns true, delete the first element of the list. Repeat until mystery returns false.