University of Western Ontario  
Department of Computer Science  
Computer Science 1027b Final Exam  
3 hours, April 11th, 2014

PRINT YOUR NAME:
PRINT YOUR SECTION NUMBER:
PRINT YOUR STUDENT NUMBER:
DO NOT TURN THIS PAGE UNTIL INSTRUCTED TO DO SO!

Instructions

• Fill in your name and student number above immediately.
• You have 3 hours to complete the exam.
• For multiple choice questions, circle your answers on this exam paper.
• For other questions, write your answers in the spaces provided in this exam paper.
• The marks for each individual question are given.
• Several interfaces are at the back of the exam.
• There are also pages for rough work at the back of the exam. You may detach them if you wish, but hand them in with the rest of the exam paper.
• Calculators are not allowed!

Mark summary

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/20</td>
<td>/10</td>
<td>/10</td>
<td>/15</td>
<td>/15</td>
<td>/25</td>
<td>/20</td>
<td>/10</td>
<td>/15</td>
<td>/30</td>
<td>/155</td>
</tr>
</tbody>
</table>
Problem 1: true/false (20 marks)

Choose one answer for each question.

1. (1 mark) Given the preorder and the inorder traversals of a tree it is possible to reconstruct the tree.
   (a) true
   (b) false

2. (1 mark) Given the preorder and the postorder traversals of a tree it is possible to reconstruct the tree.
   (a) true
   (b) false

3. (1 mark) Given the inorder and the postorder traversals of a tree it is possible to reconstruct the tree.
   (a) true
   (b) false

4. (1 mark) Heap sort is $O(n^2)$ is the worst case but $O(n)$ is the best case.
   (a) true
   (b) false

5. (1 mark) Bubble sort is $O(n^2)$ is the worst case but $O(n)$ is the best case.
   (a) true
   (b) false

6. (1 mark) Search in a balanced binary tree is $O(\log_2 n)$ in the worst case.
   (a) true
   (b) false

7. (1 mark) Search in a unbalanced binary tree is $O(n^2)$ in the worst case.
   (a) true
   (b) false

8. (1 mark) Two or more methods with same name can be differentiated on the basis of their parameters data type and number of parameters. This is called **overriding**.
   (a) true
   (b) false

9. (1 mark) A method in a subclass can have the same number and type of parameters as a method of the same name in a superclass?
   (a) true
   (b) false

10. (1 mark) The `charAt()` method can be used to obtain a substring at specified index in a string?
    (a) true
    (b) false
11. (1 mark) The more comments in a program, the faster the program runs.
   (a) true
   (b) false

12. (1 mark) Garbage collection in Java is, when all references to an object are gone, the memory used by the object is automatically reclaimed.
   (a) true
   (b) false

13. (1 mark) The following statement mystery.thing(blab) in a Java program that compiles and executes means that thing is a private variable?
   (a) true
   (b) false

14. (1 mark) The iterator() is used to obtain an iterator to the start of the collection.
   (a) true
   (b) false

15. (1 mark) Whenever a subclass needs to refer to its immediate super class it uses the keyword super.
   (a) true
   (b) false

16. (1 mark) The private variables of a class can only be accessed by methods in that class
   (a) true
   (b) false

17. (1 mark) The protected variables in a class can only be access by methods in that class or any of its super class.
   (a) true
   (b) false

18. (1 mark) Interfaces can be used to fully abstract a class from its implementation.
   (a) true
   (b) false

19. (1 mark) An infinite loop occurs when a recursive method does not have a base case.
   (a) true
   (b) false

20. (1 mark) Janitors working for Sun MicroSystems are required to throw away any Microsoft documentation on Java found in the employees’ offices.
   (a) true
   (b) false
Problem 2 (10 marks)

In each of the following situations, use big-O notation to express the amount of work being done in terms of $n$.

1. (2 marks) We execute the following code segment

   ```java
   for (int i = 1; i < n/2; i+=2)
     for (int j = 1; j < n/3; j+=3)
       System.out.println(i+j);
   ```

   Answer: $O(n^2)$

2. (2 marks) We execute the following code segment

   ```java
   for (int i = 1; i < n/2; i*=2)
     for (int j = 1; j < n/3; j*=2)
       System.out.println(i+j);
   ```

   Answer: $O((\log_2(n))^2)$

3. (2 marks) We insert an integer into a sorted list of integers using binary search.

   Answer: $O(n)$ - integers may need to be shifted, on average $n/2$ shifts.

4. (2 marks) We visit each internal node in a binary search tree using a `preOrder` method.

   Answer: $O(n)$ - we need to visit each mode once and determine if it is a leaf or internal node

5. (2 marks) We visit each leaf node in a binary search tree using a `postOrder` method.

   Answer: $O(n)$ - we need to visit each mode once and determine if it is a leaf or internal node
Problem 3 (10 marks)

In all following cases, write a code segment whose running time is as required. Use a `System.out.println(1)` statement or something like that as the statement to be executed the required number of times.

1. (2 marks) $O((\log_2 n)^2)$ using 2 nested loops:
   
   One possible answer:
   ```java
   for(int i=0;i<n;i=i*2)
   for(int j=0;j<n;j=j*2)
      System.out.println(1)
   ```

2. (2 marks) $O(2^n)$, using a recursive method and no loop:
   
   One possible answer:
   ```java
   static void func(int n){
      if (n <= 0)
         System.out.println(1);
      else{
         System.out.println(n);
         func(n-1)+func(n-2);
      }
   }
   ```

3. (2 marks) $O(n^2)$, using five nested loops.
   
   One possible answer:
   ```java
   for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      for (int k=0; k <100000; k++)
         for (int l=0; l <100000; l++)
            for (int m=0; m <100000; m++)
               System.out.println(1);
   ```

4. (2 marks) $O(n^5)$, using a single loop.
   
   One possible answer:
   ```java
   for(int i = 0; i < Math.pow(n,5);i++)
      System.out.println(1);
   ```

5. (2 marks) $O(n)$, using only a recursive function.
   
   ```java
   static void func(int n){
      if (n <= 0)
         System.out.println(1);
      else{
         System.out.println(n);
         func(n-1);
      }
   }
   ```
Problem 4 (15 marks)

This question is related to this year’s assignment 4. We wish to write a method to determine if one region is completely inside another. By this, we mean the second region is completely enclosed by the first region’s boundary. We can check this using the adjacent method on assignment 4 by doing the following for region \( r \) and this region:

1. check if \( r \) and this region are adjacent and, if true,

2. check if every additional region \( r_i \) that is adjacent to this region is NOT adjacent to region \( r \).

The adjacent method to be used here is the one on assignment 4. That is, \( r_1.\text{adjacent}(r_2) \) is true is regions \( r_1 \) and \( r_2 \) are adjacent to each other and false otherwise. Write a Boolean Java method \( \text{insideRegion}(r) \) that does this. That is, \( r_i.\text{insideRegion}(r,\text{regionQueue}) \) returns true if \( r \) is inside region \( r_i \) and false otherwise. regionQueue is a queue of all the regions found to date. Note: use queues not lists when answering this question.

```java
public Boolean insideRegion(Region r, QueueADT<Region> regionQueue) {
    if(this.adjacent(r)==false) return false;
    for(int i=0;i<regionQueue.size();i++)
    {
        Region reg=regionQueue.dequeue();
        regionQueue.enqueue(reg);
        if(r!=reg && this.adjacent(reg) && r.adjacent(reg)) return false;
    }
    return true;
}
```
Problem 5 (15 marks)

Consider the following Java program:

class Problem5_2014 {

    public static class A extends B {
        public void method2() {
            System.out.print("a 2 ");
            method1();
        }
    }

    public static class B extends C {
        public String toString() {
            return("b");
        }

        public void method2() {
            System.out.print("b 2 ");
            super.method2();
        }
    }

    public static class C {
        public String toString() {
            return "c";
        }

        public void method1() {
            System.out.print("c 1 ");
        }

        public void method2() {
            System.out.print("c 2 ");
        }
    }

    public static class D extends B {
        public void method1() {
            System.out.print("d 1 ");
            method2();
        }
    }
}


public static void main(String[] args) {

    C[] elements = {new A(), new B(), new C(), new D()};

    for (int i = 0; i < elements.length; i++) {
        System.out.println(elements[i]);
        elements[i].method1();
        System.out.println();
        elements[i].method2();
        System.out.println();
        System.out.println();
    }
}
} // Problem5_2014

Hand trace this program (show what is printed) below:

    b
c 1
a 2 c 1

    b
c 1
b 2 c 2

    c
    c 1
    c 2

    b
d 1 b 2 c 2
b 2 c 2
Problem 6 (25 marks)

We consider the selection sorting algorithm for int's. We want to write a recursive implementation of it, using only arrays.

(6 marks) Write a method public static void swap(int[] data, int i1, int i2) that exchanges the contents of data[i1] and data[i2]. For instance, if array data is initially (12, 43, 56, 87), i1 is 3 and i2 is 0, data should be (87, 43, 56, 12) after calling the method. You can assume that i1 and i2 are not out of bounds.

```java
public static void swap(int[] data, int i1, int i2){
    int tmp = data[i1];
    data[i1] = data[i2];
    data[i2] = tmp;
}
```

(9 marks) Write a method public static int indexMinimum(int[] data, int start) which returns the index of the smallest element among data[start], data[start+1], . . . , data[data.length-1]. For instance, if data is (12, 0, 35, 20) and start is 2, the result should be 3. You can assume that start is greater than or equal to 0, and less than data.length.

```java
public static int indexMinimum(int[] data, int start){
    int index = start;
    for (int i = start+1; i < data.length; i++)
        if (data[i] < data[index])
            index = i;
    return index;
}
```
(7 marks) Write a recursive method `public static void sortRec(int[] data, int start)` that sorts the portion of `data` between indices `start` and `data.length-1` (inclusive). If `start ≥ data.length`, don’t do anything. Otherwise, find the index of the minimum element between indices `start` and `data.length-1`, exchange it with `data[start]` and do a recursive call on the portion of `data` starting at `start+1`. Note: this method returns nothing; it modifies the array `data`. Use `indexMinimum` in your solution.

```java
public static void sortRec(int[] data, int start){
    if (start >= data.length)
        return;
    int index = indexMinimum(data, start);
    swap(data, start, index);
    sortRec(data, start+1);
}
```

(3 marks) Write a method `public static void sort(int[] data)` that sorts the whole array, using `sortRec`.

```java
public static void sort(int[] data){
    sortRec(data, 0);
}
```
Problem 7 (20 marks)

Consider the following binary tree:

(7a) (3%) Give the inorder traversal of this tree:

6472513

(7b) (3%) Give the preorder traversal of this tree:

1246753

(7c) (3%) Give the postorder traversal of this tree:

6745231

(7d) (3%) Give the level order traversal of this tree:

1234567
Consider the mystery traversal 6753. Write a recursive method that produces this output

```java
public static void mystery(BinaryTreeNode<Integer> node) {
    if (node == null)
        return;
    // a leaf node
    if (node.getLeft() == null && node.getRight() == null)
        System.out.print(node.getElement());
    else{
        mystery(node.getLeft());
        mystery(node.getRight());
    }
}
```
Problem 8 (10 marks)

Write a method `size` that gives the number of nodes in a `BinaryTreeNode` (you can only use the getters we give).

```java
public static int size(BinaryTreeNode<Integer> node) {
    if (node == null)
        return 0;
    return 1 + size(node.getLeft()) + size(node.getRight());
}
```
Problem 9 (15 marks)

We say that a binary tree is strict if every node has either 0 or 2 children.

(3 marks) Is the tree of Problem 7 strict?
   yes

(12 marks) Write a method isStrict that returns whether the input tree is strict (you can only use the getters we give).

   public static boolean isStrict(BinaryTreeNode<Integer> node) {
       if (node == null)
           return true;
       if (node.getLeft() == null && node.getRight() != null)
           return false;
       if (node.getLeft() != null && node.getRight() == null)
           return false;
       return isStrict(node.getLeft()) && isStrict(node.getRight());
   }
**Problem 10 (15 + 15 marks)**

Suppose we have a class `OrderedList` of `Integer`'s sorted in increasing order, with the following methods:

```java
public class OrderedList{
    public OrderedList();
    public Integer getFirst(); // returns the first element, the list doesn’t change
    public void removeFirst(); // removes the first element
    public void setFirst(Integer s); // sets the first element, shifts all other elements
    public boolean isEmpty();
}
```

For instance, starting from the list (2, 3, 7), `getFirst` would return 2, `removeFirst` would change the list to (3, 7) and `setFirst(0)` would modify the list to make it (0, 3, 7).

We want to insert an `Integer s` in an `OrderedList` called `list`, using *only* the methods above. We use the following algorithm. To insert `s` in `list`, we can suppose that `list` is not empty (if it is empty, just insert `s` in it). Then, we can extract the first element of `list`, call it `t`, and remove it from the list. If `s ≤ t`, set `t` as the first element of `list`, and then `s`. Otherwise, insert `s` in `list`, then set `t` as the first element.

(15 marks) Write a method `public static void insert(Integer s, OrderedList list)` that implements this algorithm. The algorithm must modify the input list. You can assume that `list` is not null.

```java
public static void insert(Integer s, OrderedList list){
    if (list.isEmpty()){
        list.setFirst(s);
        return;
    }

    Integer t = list.getFirst();
    list.removeFirst();

    if (s <= t){
        list.setFirst(t);
        list.setFirst(s);
    } else{
        insert(s, list);
        list.setFirst(t);
    }
}
```
**Bonus** question (15 marks) We want to *merge* two *OrderedLists*, while maintaining the order. Suppose for instance that `list1` is `(0, 2, 3, 9)` and `list2` is `(2, 4, 10)`: the output should be `(0, 2, 2, 3, 4, 9, 10)`. Write a recursive method for this. It may or may not modify the input lists.

```java
public static OrderedList merge(OrderedList list1, OrderedList list2)
{
    if (list1.isEmpty())
        return list2;
    Integer t = list1.getFirst();
    list1.removeFirst();
    OrderedList list = merge(list1, list2);
    insert(t, list);
    return list;
}
```
Interfaces and classes

public interface StackADT<T>{
    // Adds one element to the top of this stack.
    public void push (T element);

    // Removes and returns the top element from this stack.
    public T pop();

    // Returns without removing the top element of this stack.
    public T peek();

    // Returns true if this stack contains no elements.
    public boolean isEmpty();

    // Returns the number of elements in this stack.
    public int size();

    // Returns a string representation of this stack.
    public String toString();
}

class Stack<

public interface QueueADT<T>{
    // Adds one element to the rear of this queue.
    public void enqueue (T element);

    // Removes and returns the element at the front of this queue.
    public T dequeue();

    // Returns without removing the element at the front of this queue.
    public T first();

    // Returns true if this queue contains no elements.
    public boolean isEmpty();

    // Returns the number of elements in this queue.
    public int size();

    // Returns a string representation of this queue.
    public String toString();
}

class Queue<

17
public interface ListADT<T> extends Iterable<T>{
    // Removes and returns the first element from this list.
    public T removeFirst();

    // Removes and returns the last element from this list.
    public T removeLast();

    // Removes and returns the specified element from this list.
    public T remove(T element);

    // Returns a reference to the first element in this list.
    public T first();

    // Returns a reference to the last element in this list.
    public T last();

    // Returns true if this list contains the specified target element.
    public boolean contains(T target);

    // Returns true if this list contains no elements.
    public boolean isEmpty();

    // Returns the number of elements in this list.
    public int size();

    // Returns an iterator for the elements in this list.
    public Iterator<T> iterator();

    // Returns a string representation of this list.
    public String toString();
}

public interface Iterator<T>{
    // Returns true if the iterator has more elements.
    boolean hasNext();

    // Returns the next element in the iterator.
    T next();
}

public class BinaryTreeNode<T>{
    protected T element;
    protected BinaryTreeNode<T> left, right;

    // the getters you will need
    public T getElement();
    public BinaryTreeNode<T> getLeft();
    public BinaryTreeNode<T> getRight();
}
Rough work 1/4
Rough work 2/4
Rough work 3/4
Rough work 4/4