STUDENT NAME: _____

STUDENT ID: _____

McGill University
Faculty of Science
School of Computer Science

FINAL EXAMINATION

COMP-250: Introduction to Computer Science - Winter 2008

April 29, 2008
2:00-5:00

Examiner: Prof. Doina Precup

Associate Examiner: Prof. Mathieu Blanchette

Write your name at the top of this page. Answer directly on exam paper. Three blank pages are added at the end in case you need extra space. There are 11 questions worth 100 points in total. The value of each question is found in parentheses next to it. Please write your answer on the provided exam. Partial credit will be given for incomplete or partially correct answers.

SUGGESTIONS: READ ALL THE QUESTIONS BEFORE YOU START! THE NUMBER OF POINTS IS NOT ALWAYS PROPORTIONAL TO THE DIFFICULTY OF THE QUESTIONS. SPEND YOUR TIME WISELY!

GOOD LUCK!

1. [15 points, 3 points each] **True or false?**

   Indicate whether the following statements are true or false. Give a two-line justification for each. Credit will be given only if the justification is correct.

   (a) $f(n) = 1000n + 5$ is $O(n^2)$

   (b) All sorting algorithms are $O(n \log n)$.

   (c) Suppose that some algorithm A has $O(n^3)$ and algorithm B has $O(n \log n)$. Then B is always preferred to A.

   (d) There exists a polynomial-time algorithm for deciding if there is an assignment of truth values to the variables in a Boolean expression such that the expression is true.

   (e) In Java, if class *C extends* class *B*, then any method from class *B* can be called on an object of class *C*.

2. [10 points]**Big-Oh**

   For the following algorithms, state what $O()$ is and explain your answer:

   (a) **Algorithm** f1(n)
   $i \leftarrow 1$
   while $i < n$
   for $j = 1$ **to** i **do**
   print(j)
   $i \leftarrow i + 2$

   (b) **Algorithm** f2(n)
   $i \leftarrow 1$
   while $i < n$
   print(i)
   $i \leftarrow i + 2$
   for $j = 1$ **to** i **do**
   print(j)

   (c) **Algorithm** f3(n)
   if $n = 0$ **return**
   print(n)
   f3($n/100$)

3. [10 points] **Java**

Consider the following piece of Java code. Write next to every printing statement in the main function what its output will be.

```
public class MyParentClass {
   int x;
   public MyParentClass() { x=1; }
   public int get_x() { return x; }
   public void add_to_x(int y) {
      x=x+y;
   }
}

public class MyChildClass extends MyParentClass {
   public MyChildClass() { x=2; }
   public void new_add_to_x(int y) {
      super.add_to_x(y);
      x=x+1;
   }
   public void another_add_to_x(int y) {
      super.add_to_x(y);
      y=10;
   }
}

   public static void main (String[] args) {
      int y=3;

      MyParentClass obj = new MyParentClass();
      System.out.println(obj.get_x());
      obj.add_to_x(y);
      System.out.println(obj.get_x());

      MyChildClass c = new MyChildClass();
      System.out.println(c.get_x());
      c.add_to_x(y);
      System.out.println(c.get_x());
      c.new_add_to_x(y);
      System.out.println(c.get_x());
      c.another_add_to_x(y);
      System.out.println(c.get_x());
      System.out.println(y);
   }
```

4. [10 points] **A simple algorithm**

   You are given an array of n positive integers. The numbers do not appear in any particular order. Write an algorithm that will make the array "wiggly", i.e. arrange the elements in such a way that $a[1] \leq a[2] \geq a[3] \leq a[4]$ etc. In other words, a node with an even index must be greater than or equal to its two neighbors, and a node with odd index must be less than or equal to its neighbors. You may call as a subroutine any algorithm we discussed in class. State and justify what $O()$ is for your algorithm.

5. [5 points] **Sorting and Linked Lists**

   Suppose that you are asked to provide a sorting algorithm for doubly linked lists which works *without copying the list into an array*. What sorting algorithm would you use, and what would $O()$ be? Justify your answer (no pseudocode necessary).

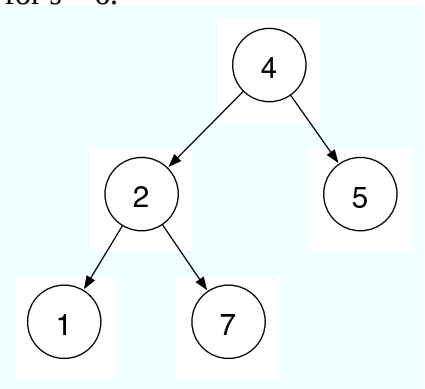
6. [10 points] **Binary trees**

   Consider a BinaryTreeNode data structure with three attributes:

   - content - an integer
   - left - reference to a BinaryTreeNode representing the left child (null if there is no left child)
   - right - reference to a BinaryTreeNode representing the right child (null if there is no right child)

   Write an algorithm in pseudocode that takes as input a BinaryTreeNode n and an integer s. Your algorithm should return true if there is a path from n to the leaves such that the sum of the node content along the path is equal to s, and false otherwise.

   For example, for the tree below, when called with a reference to node 4, your algorithm should return true for s= 9 and false for s= 6.

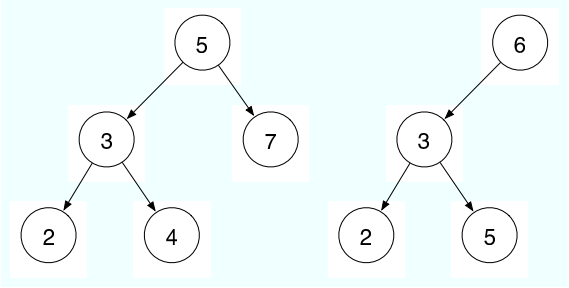


   Algorithm hasPathSum (BinaryTreeNode n, int s)
   Input: A BinaryTreeNode n and an integer s
   Output: true if there is a path from root to leaves in the tree such that the sum of the nodes in the path is s, false otherwise

7. [10 points] **Binary search trees**

   Recall that in a binary search tree, at every node, all elements to the left of the node have a smaller key, and all elements to the right of a node have a larger key. Write in pseudocode an algorithm which, given a binary search tree with at least two nodes, will return the element with the node with the *second largest* key in the tree.

   For example, for both of the trees below, the algorithm should return 5.



   Your algorithm should run in $O(h)$, where h is the height of the tree

   Algorithm findSecondMax(BinarySearchTreeNode n)
   Input: A binary search tree node, representing the root of a BST
   Output: A the node in the binary search tree whose key is second largest

8. [5 points] **Heaps**

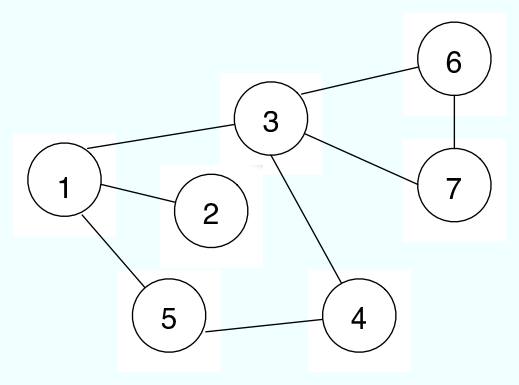
   In the heaps we studied in class, one could retrieve the minimum element in $O(1)$. Now suppose that instead you want to retrieve the *maximum* element in $O(1)$. What properties should such a heap have?

9. [5 points] **Hash tables**

   Suppose you are hired to build an application in which you need to store data about millions of company customers in a hash table. You have no idea what would be a good hashing function, but you look on the internet and find two promising candidates. However, you have no idea which function would work best. Describe an approach which can take advantage of both functions. You do not need to write pseudocode, but your description should be precise.

10. [10 points] **Search in graphs**

Consider the graph in the figure below.



(a) [4 points] Suppose that you are using breadth-first search to find a path from vertex 1 to vertex 7. Show the state of the queue after each node expansion, assuming nodes are enqueued in increasing order. What path do you find?

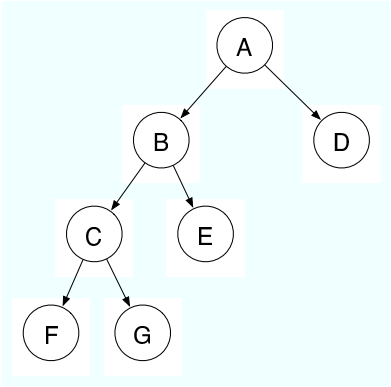
(b) [4 points] Suppose that you are using depth-first search for the same task. Show the node expansion, assuming that successors of the same node are investigated in increasing order. What path do you find?

(c) [2 points] Based on this example, what is the main advantage of depth-first search? What is the main advantage of breadth-first search?

11. [10 points] **Proof by induction**

Recall that a binary tree is called *proper* if every node has either 2 or 0 descendents. Suppose that you have a proper binary tree with n internal nodes. Let $PI(n)$ be the sum of the length of the paths between the root and all the internal nodes, and $PL(n)$ be the sum of the length of the paths between the root and all the leafs.

For example, in the tree below, A, B and C are internal nodes, $PI(n) = 0+1+2 = 3$ and $PL(n) = 1+2+3+3 = 9$.



Prove by induction that $PL(n) - PI(n) = 2n$.

Page left intentionally blank in case you need extra space

Page left intentionally blank in case you need extra space

Page left intentionally blank in case you need extra space