NAME:____________________________STUDENT NUMBER: _________________.

Faculty of Science
FINAL EXAMINATION

COMP-250 A – Introduction to Computer Science
School of Computer Science, McGill University

Examimer: Prof. Mathieu Blanchette December 8th 2005, 14:00-17:00
Associate examiner: Prof. Prakash Panangaden

INSTRUCTIONS
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.
This examination is worth 50% of your final grade.
The total of all questions is 100 points.
The value of each question is found in parentheses next to it.
This is an open book exam though sharing materials with other students is *not* permitted.
No electronic devices, calculator, laptop computer, cell phones, etc. are allowed.
All logs are in base 2.
This exam comprises 14 pages, including the cover page and three blank pages at the end.

**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!**

Grading scheme:

1.	/ 30
2.	/ 12
3.	/ 10
4.	/ 6
5.	/ 12
6.	/ 10
7.	/ 10
8.	/ 10
Total.	/ 100

Question 1. (30 points, 2 points each)

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

a) If $f(n)$ is $O(\ h(n)\ )$ and $g(n)$ is $\Omega(\ h(n)\ )$, then $f(n) \bullet g(n)$ is $\Theta(\ (h(n))^2\ )$.

b) $2^{(2 \log(n))}$ is $O(n)$.

c) To prove that $f(n)$ is $O(\ g(n)\ )$, all one needs to do is to show that there exists a number n_0 such that $f(n_0) \leq c \bullet g(n_0)$, for some constant number c.

d) Although many sorting algorithms have average-case running time $O(n \log(n))$, they all have worst-case running time $O(n^2)$.

e) Suppose that a hash table has K buckets, the buckets are dictionaries implemented with a balanced binary search tree, and the hash table contains a total of N elements. Then, under the best possible choice of hash function, any find(key) operation will take time $O(N/K \log(N/K))$.

f) If it was possible to write a “merge” algorithm that would run in time $O(\sqrt{n})$, then the running time of mergeSort would be $O(\sqrt{n})$.

g) If a hash table contains n buckets, then it can never contain more than n elements.

h) Greedy algorithms are never guaranteed to yield a correct solution.

i) To increase the Google pageRank of your homepage, you should include in your homepage links pointing to many important web sites like yahoo, amazon, mcgill...

j) If someone discovered a polynomial-time algorithm to solve the k-CLIQUE, decision problem, then we would also have a polynomial algorithm for SAT.

k) A decision problem cannot be at the same time NP-Complete and undecidable.

l) Suppose that we are guaranteed that a queue will never contain more than *n* elements. Then, it is possible to implement that queue using an array of size n in such a way that any enqueue and dequeue operations can be performed in constant time.

m) A Monte Carlo algorithm may have a non-zero probability of returning an incorrect result, but this probability can be made as small as desired at the price of longer running time.

n) In a two-player game, a position is winning for player A if and only if all possible moves available to A from that position lead to losing positions for player B.

o) The dynamic programming algorithm for making change is guaranteed to give the optimal number of coins for any possible coin denominations.

Question 2. (12 points)

For each question below, give a 4-line (at most) answer that is as precise as possible.
a) (3 points) Assume that T is a Binary Search Tree. What is the tree traversal algorithm that will print the keys of T in increasing order when it is called on the root of T ?

b) (3 points) Explain how any recursive algorithm can be transformed into an iterative algorithm.

c) (3 points) Give an example of an array of 8 elements on which the QuickSort algorithm seen in class runs as fast as possible. No justifications are required.

d) (3 points) Consider a hash table containing n objects distributed over 101 buckets, each implemented with a linked-list. Then, under the best possible choice of hash function, the "find" operation takes time $O(n/101) = O(n)$, which is the same big-O running time as if we were using a single linked-list to store the n objects. Why then is the hash table generally better?

Question 3. (10 points)

Consider the following problem:

Given: An unsorted array *A* of *n* distinct integers, and an integer *z*
Problem: Return the number of pairs of indices *i* and *j* such that $0 \leq i < j < n$ and such that $A[i]+A[j] = z$.

For example, if A=[3 5 6 1 4 2 5 8] and z = 9, then the algorithm should return the number 4, because A[0]+A[2]=9, A[1]+A[4]=9, A[3]+A[7]=9, A[4]+A[6]=9.
Write an algorithm that solves this problem in worst-case time O(*n* log *n*).
You can use any of the algorithms and data structures seen in class without redefining them.

Algorithm countPairs(*A, n, z*)
Input: An array *A* of *n* integers. An integer *z*.
Output: the number of pairs (*i,j*) such that $0 \leq i < j < n$ and such that $A[i]+A[j] = z$.
/* WRITE YOUR PSEUDOCODE HERE */

Question 4. (6 points)

In Homework #3, you were asked to implement an algorithm evaluate to any parenthesized arithmetic expression of the type (((3+4)*5+(3-(6*9))).

a) (3 Points). What type of check allows you to detect that an expression like ((1+2) is malformed?

b) (3 points). Recall that the Stack class in java as a method called "peek" that returns an object of type Object. Then, assuming that you defined a Stack called operandStack and that you already pushed some objects of type Integer onto operandStack, the following statement is illegal:

   Integer x = operandStack.peek();

   Why does the line above produce a compilation error? Modify the line to fix the error while retaining the meaning.

Question 5. (12 points)

Consider the following algorithm that operates on the node of a linked list.

```
Algorithm ExamPrint(node n)
Input: a node n
Output: Prints something...
if ( n ≠ null ) then print n.getValue()
if ( n.getNext() ≠ null ) then {
      ExamPrint( n.getNext() )
}
if ( n.getNext() ≠ null AND n.getNext().getNext() ≠ null ) then {
      ExamPrint( n.getNext().getNext() )
}
```

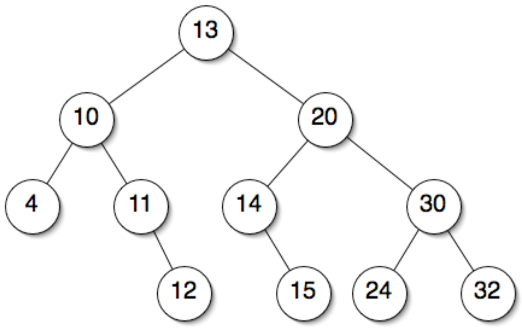
a) (3 points). What does the ExamPrint algorithm print when it is called on the head of the linked list: $5 \rightarrow 1 \rightarrow 3 \rightarrow 4$?

b) (4 points). Let *T(n)* be the total number of items printed when ExamPrint is called on the head of a list that contains *n* elements. Write a recurrence for *T(n)*.

c) (5 points). Prove formally that $T(n)$ is $\Omega((3/2)^n)$.

Question 6. (10 points)

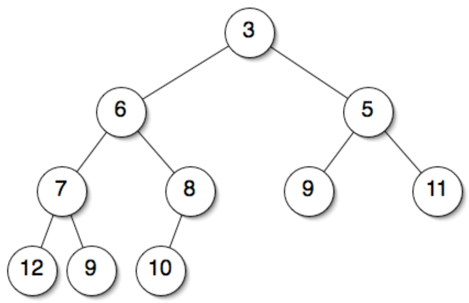
Consider the following binary search tree:



a) **(2 points)** Draw the binary search tree after an element with key 31 has been inserted.

b) **(3 points)** Draw the binary search tree after *remove*(13*)* has been executed. Start from the original tree, not the tree you got in (a).

Consider the following heap:



c) **(2 points)** Draw the heap after an element with key 4 has been inserted.

d) **(3 points)** Draw the heap after *removeMin*() has been executed. Start from the original heap, not the one you got in (c).

Question 7. (10 points)

Let *T* be a binary search tree that contains a set of keys with distinct integers. Assume that you have a method subtreeSize(treeNode *n*) that returns the number of nodes in the subtree rooted at *n*, including *n* itself. Assume at any call to subtreeSize(treeNode *n*) takes time O(1).

Problem: Write an algorithm that computes the number of nodes with a key greater or equal to a given integer *k*. Your algorithm should run in worst-case time O(*h*), where *h* is the height of the binary search tree (but you don't need to prove it).

Algorithm nbNodesGreaterEqual(treeNode *n*, int *k*)
Input: A treeNode *n* and an integer *k*
Output: The number of nodes with key greater or equal to *k* in the subtree rooted at *n*.
/* WRITE YOUR PSEUDOCODE HERE */

Question 8. (10 points)

An undirected connected graph *G* is said to be *acyclic* if it contains no cycles.
Assume that you are given a vertex *u* from *G*.
Question: Write an algorithm that checks whether the graph to which *u* belongs is acyclic. The only information you have about *G* is that it is connected and that it contains vertex *u*.
Use the following standard graph ADT methods if needed.

- *getNeighbors*(vertex *v*) returns the list of vertices that are the adjacent to vertex *v*. It is ok for you to write something like: for each vertex *w* in *getNeighbors*(*v*) do ...
- boolean *getVisited*(vertex *v*) returns TRUE if and only if vertex *v* has been marked as visited.
- *setVisited*(vertex *v*, boolean *b*) sets the visited status of vertex *v* to *b*.

Notes:

- If your "isAcyclic" algorithm needs to take more than just vertex u as argument, feel free to add other arguments. In that case, also specify what initial values should be given to these arguments when calling your isAcyclic method.
- You can add other members to the vertex class or define other methods if needed, but please specify their type and meaning.

Algorithm isAcyclic(vertex *u, ...*)
Input: a vertex *u* from the graph, and other arguments if you want...
Output: Returns true if the graph is acyclic, and false otherwise
/* WRITE YOUR PSEUDOCODE HERE */

(use next page if needed)

Page left blank intentionally. Use it if you need extra space.

Page left blank intentionally. Use it if you need extra space.

Page left blank intentionally. Use it if you need extra space.