University of California at Berkeley
Department of Electrical Engineering and Computer Sciences
Computer Science Division

Spring 2009 **Jonathan Shewchuk**

CS 61B: Midterm Exam II

This is an open book, open notes exam. Electronic devices are forbidden on your person, including cell phones, iPods, headphones, laptops, and PDAs. Turn your cell phone off and leave it and all electronics at the front of the room, or risk getting a zero on the exam. **Do not open your exam until you are told to do so!**

Name:_____

Login:_____

Lab TA:_____

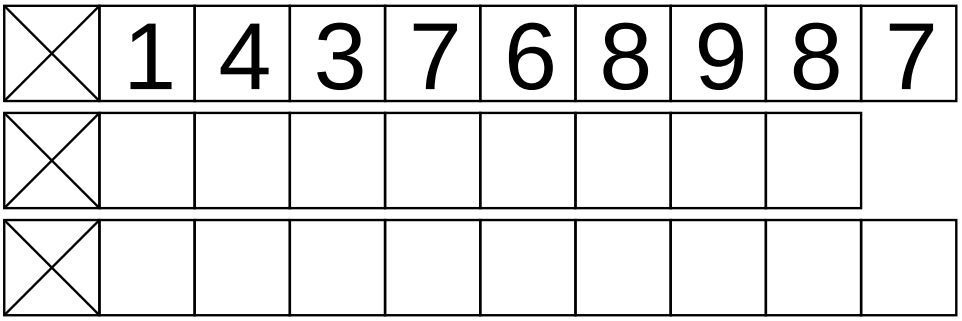
Lab day and time:_____

Do not write in these boxes.

Problem #	**Possible**	**Score**
1. Miscellaneous	12	
2. Trees	6	
3. Graph Adjacency	7	
Total	25	

Problem 1. (12 points) **A Miscellany.**

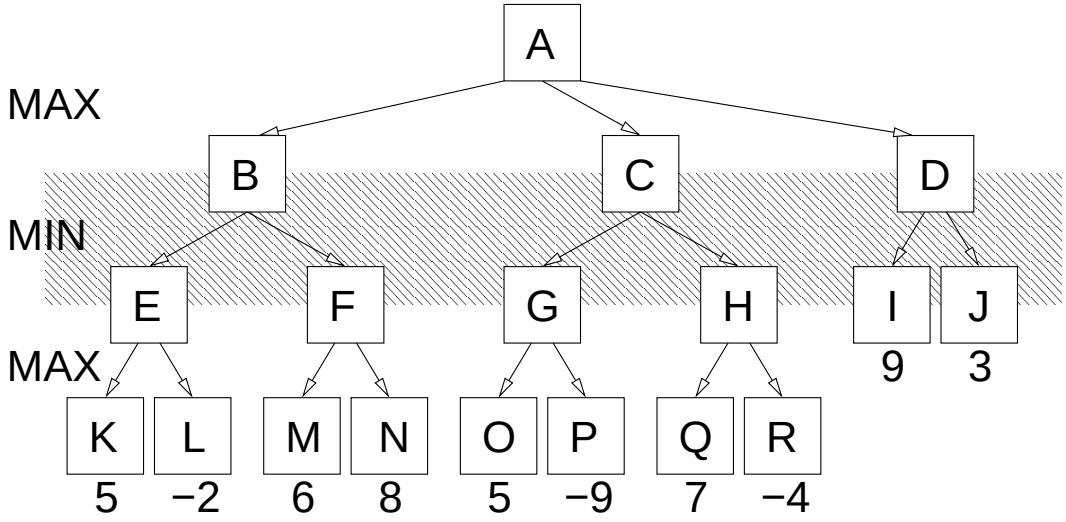
a. (3 points) Draw the following binary heap after `removeMin()`, then again after `insert(2)`.



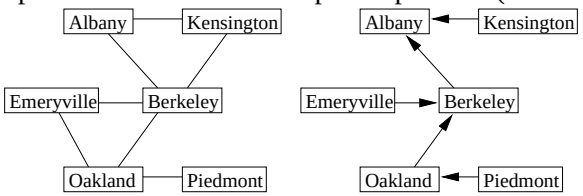
b. (1 point) Each of the exceptions `XException`, `YException`, `ZException`, `BException`, `CException`, `DException`, and `EException` is declared so it "`extends Exception`". Which exception, if any, propagates out of the following code? _____.

```
public static void z() throws Exception {
  try {
    throw new XException();
  } catch (YException y) {
    throw new ZException();
  } catch (Throwable t) {
    try {
      throw new BException();
    } catch (RunTimeException c) {
      throw new CException();
    }
    throw new DException();
  } catch (BException b) {
    throw new EException();
  }
}
```

c. (3 points) The leaf nodes of the following game search tree are scored as indicated. Cross out all the nodes that will be pruned (i.e. not visited) if alpha-beta pruning is used. Assume children are explored from left to right.



d. (1 point) Recall that when we perform a breadth-first search of a graph (like the graph at below left), we can build a tree as follows: every time we traverse an edge (u, v) to visit a vertex v for the first time, we set u to be the parent of v. The set P of parent pointers (below right) forms a tree.



Suppose we consider a complete graph $G = (V, E)$ with 1,000 vertices. (A complete graph has every possible edge; for any vertices $u, v \in V$, the edge (u, v) is in the set E of edges.) If we form a tree $T = (V, P)$ by performing breadth-first search on G starting from a vertex w, what will be the indegree of w in T? (The *indegree* of w is the number of edges of T whose destination is w.)

e. (1 point) If we form a tree T by performing *depth*-first search on G starting from a vertex w, what will be the indegree of w in T?

f. (3 points) If $f(n) \in O(g(n))$, and $h(n) \in \Omega(n \cdot g(n))$, can $f(n)$ be in $\Theta(h(n))$?

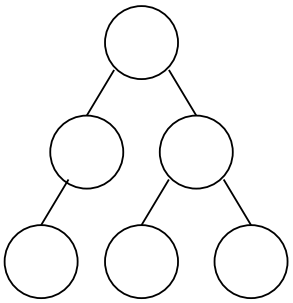
(Assume all three functions are strictly positive for $n > 0$.)

If "yes," give an example of functions f, g, and h for which all three are true.

If "no," explain why, using the definitions of O, Ω, and Θ to demonstrate the impossibility.

Problem 2. (6 points) **Trees.**

a. (1 point) Fill in the following tree so its postorder traversal is `K O I N C R`.

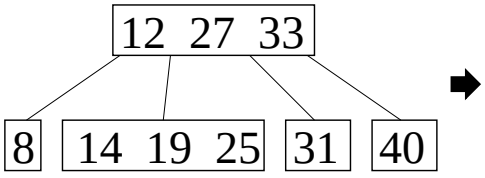


b. (1 point) Suppose that you run the following code on a 2-3-4 tree that initially contains n entries. Recall that `first()` is the method that returns the entry with the smallest key (but doesn't remove it). Recall also that a 2-3-4 tree can store multiple entries with the same key.

```
for (i = 0; i < r; i++) {
  insert(first().key, first().value);
}
```

Expressed as a function of the initial tree size n and the number r of loop iterations, the worst-case running time of this loop is in $\Theta($_____$)$.

c. (2 points) Draw the following 2-3-4 tree after execution of the operation `insert(13)`.



d. (2 points) Suppose you have a binary search tree that is *perfectly* balanced; it has exactly $n = 2^{d+1} - 1$ nodes, where d is its depth. It is stored in an array by level numbering (though it's *not* a heap). The tree contains n numerical keys with no duplicates.

Describe (in English) how to find the ith-largest key **as quickly as possible**. What is the asymptotic running time of your algorithm in terms of n?

Problem 3. (7 points) **Graph Adjacency Running Times.**

Let $G = (V, E)$ be a directed graph, and let u be a vertex in V, where

$\mathbf{n} = |V|$ is the number of vertices,
$\mathbf{e} = |E|$ is the number of edges, and
$\mathbf{d}$ is the outdegree of u (in other words, the number of vertices v such that $(u, v) \in E$, where E is the set of edges in the graph).

We want to support four operations on graphs:

insert(u, v). Add a new edge (u, v) to the graph. One may insert an edge that's already in the graph, in which case the graph does not change.
remove(u, v). Remove an edge (u, v) from the graph.
member(u, v). Is edge (u, v) in the graph?
printList(u). Print (on the screen) every edge whose source is u, in any order.

Each vertex is represented by an integer in the range $0 \ldots \mathbf{n} - 1$. We will consider five different ways to store the edges (E), including an adjacency matrix, two variations on adjacency lists, an edge list, and a dictionary:

Adjacency matrix. As discussed in class, the entire set of edges is represented as an $\mathbf{n} \times \mathbf{n}$ array of booleans.
Adjacency list with sorted arrays. For each vertex v, we store a sorted array of vertices w for which $(v, w) \in E$. This array is **not** like a row of an adjacency matrix: missing edges do not take up any space. However, the array has empty space at the end so it never needs to be resized. Duplicate items are forbidden in any one list.
Adjacency list with balanced search trees. For each vertex v, we maintain a 2-3-4 tree containing every vertex w such that $(v, w) \in E$. Duplicate items are forbidden in any one search tree.
One sorted array. Instead of having a separate array for each vertex, we have one array that stores every edge $(v, w) \in E$. The edges are sorted in order by v, with ties broken by w. The array has empty space at the end so it never needs to be resized.
Dictionary. All of the edges (for *every* source vertex) are stored in a single hash table (with chaining). The hash table's size is in $\Theta(\mathbf{e})$. Assume it never needs to be resized, and that there are only $O(1)$ collisions in any one bucket.

Assuming you implement each data structure as intelligently as you can, what is your tightest (smallest) bound on the **worst-case** running time for each operation, expressed in terms of $\mathbf{n}$, $\mathbf{e}$, and $\mathbf{d}$?

	insert(u, v) or **remove**(u, v)	**member**(u, v)	**printList**(u)
Adjacency matrix			
Sorted arrays			$\Theta(\mathbf{d})$
Balanced search trees			
One sorted array			
Dictionary			