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

Autumn 2006 **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!**

Name:____________________________________

Login:____________________________________

Lab TA:__________________________________

Lab day and time:____________________________

Do not write in these boxes.

Problem #	**Possible**	**Score**
1. Miscellaneous	8	
2. Binary Trees	8	
3. Directed Graphs	9	
Total	25	

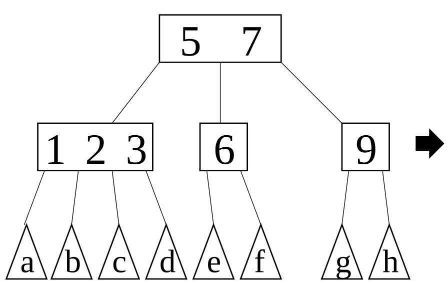
Problem 1. (8 points) **A Miscellany.**

a. (2 points) Suppose you implement a binary search method, and it takes 10 microseconds to search a 1,000,000-int array for a number that turns out not to be in the array. How long do you estimate it would take for your implementation to search a 1,000,000,000,000-int array for a number that's not in the array?

b. (2 points) Explain why $\frac{1}{n-100} \in O(n)$. Don't just reiterate the definition of big-Oh; give specific values of c and N that support the claim. (No need for a complete proof, though.)

c. (2 points) When we remove an item from a 2-3-4 tree, sometimes a node tries to "steal" a key from a sibling by performing a rotation. However, the `remove` algorithm we learned in class only allows a node to steal from an *adjacent* sibling. But there's no fundamental reason why a node couldn't steal a key from a nonadjacent sibling, except that we wanted to keep the algorithm as simple as possible.

Show how we could modify the 2-3-4 tree below so that the "9" node has one more key, the "1 2 3" node has one fewer key, every other node has the same number of keys it had before, and the tree is still a valid 2-3-4 tree containing the same keys it had before. The triangles at the bottom are subtrees, which you cannot restructure (but you can change their parents).



d. (2 points) Fill in the blank. The worst-case running time of the following method, expressed as a function of the input parameter n, is in $\Theta($______________________________$)$. Note that the loop does not use "`i--`"!

```
public static void weirdo(long n) {
  Tree234 tree = new Tree234();   // Construct an empty 2-3-4 tree.

  for (long i = n; i > 1; i = i / 2) {
    tree.insert(i);
  }
}
```

(It's not quite like the running time of any other algorithm you've seen. An explanation of your reasoning might help you get part marks if your answer is wrong.)

Problem 2. (8 points) **Binary Trees.**

a. (2 points) Draw a tree that is *both* a binary search tree and a complete binary tree, and whose keys are the integers 1 through 9 (each appearing once).

b. (2 points) If you perform the operation `remove(5)` on a binary search tree, it will obviously delete 5 from the preorder traversal of the tree. For example, if the preorder traversal was `5 2 1 3 4` before the deletion, then it is `2 1 3 4` after the deletion. But can the operation `remove(5)` ever change the *order* of keys in a preorder traversal? If "yes," give an example. If "no," explain why.

c. (4 points) Show the sequence of swaps by which the method `bottomUpHeap` converts the following array into a heap. Redraw the array once for each swap (in the boxes provided).

(Note: We've provided room for up to seven swaps, the maximum number possible for eight items. That doesn't necessarily mean that this particular example takes seven swaps.)

	7	2	5	9	3	4	1	8

d. (0 points) If you could choose to be a tree,
What kind of tree would you choose to be?

Problem 3. (9 points) **Directed Graphs.**

a. **First**, read the instructions on the next page; then come back here. You may tear the next page off your exam.

```
public class DiGraph {
  int size;                   // Number of vertices in the graph.
  Set[] adj;                  // adj[0]...adj[size - 1] are Sets of edges out.

  // Method that contracts vertex y into vertex x.
  public void contract(int x, int y) {









  }                                              Check here if your answer |   |
}                                                is continued on the back. |___|
```

For the following questions, give the simplest, smallest bounds possible in terms of the number $s =$ `size` of vertices and the number e of edges. Make sure your bounds are correct even when e is much less or much greater than s. (If you skipped part a., assume you have the fastest possible implementation.)

b. The worst-case running time of your `contract` method is in $\Theta($__________________________$)$.

c. If you were to reimplement your `Set` class so it stores the set in a resizable hash table (with the hash table's size proportional to the size of the set) instead of a list, the worst-case running time of your `contract` method would be in $\Theta($__________________________$)$. Assume no hash table chain gets longer than $\mathcal{O}(1)$, and don't count the time for hash table resizing.

d. If you were to reimplement your `Set` class so it stores the set in a 2-3-4 tree (instead of a list), the worst-case running time of your `contract` method would be in $\Theta($__________________________$)$. Assume that computing the union or intersection of two 2-3-4 trees (to produce a new 2-3-4 tree) takes time linear in the total size of the two trees.

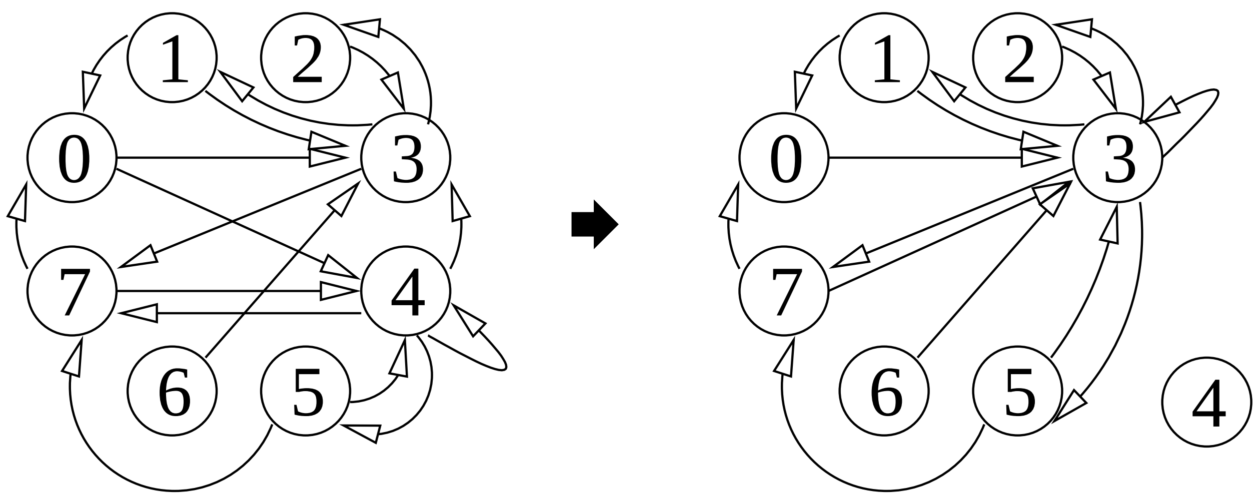
Hint: if your implementation runs as fast as it should, the answers to parts b–d are all different.

Suppose you implement a directed graph class, `DiGraph`, using an adjacency list. The vertices are the integers $0 \ldots$ `size` $- 1$. `DiGraph` represents the adjacency list as an array of `Set`s from Homework 5. Recall that your `Set` implementation stores numbers in a sorted linked list. `Set`s support the following methods.

```
public Set()                      // Constructs an empty Set.
public void insert(int i)         // Insert i into this Set.
public boolean remove(int i)      // Remove i from this Set.
public void union(Set s)          // Assign this = (this union s).
public void intersect(Set s)      // Assign this = (this intersect s).
```

Recall that neither `union` nor `intersect` change `s`. Note that we've added a new method `remove`. If the integer `i` is in the `Set`, `remove` removes `i` from the `Set` and returns `true`. Otherwise, it returns `false` (without changing the `Set`). You may access `Set`s only through these methods.

Write a method to *contract* an edge with vertices `x` and `y`—in other words, to combine the two vertices into one vertex `x`. After an edge contraction, all the edges that pointed into or out of vertex `y` now point into or out of vertex `x` instead, and no edge is incident on `y`. This figure illustrates contracting the vertices 3 and 4, so that 4 becomes disconnected from the rest of the graph.



If an edge contraction creates multiple copies of an edge, only one is stored. (Your `Set` class will handle this for you automatically, since a set can't contain two copies of the same item.) If the graph had one (or more) of the edges (`x`, `y`), (`y`, `x`), or the self-edge (`y`, `y`) before the contraction, then vertex `x` gains a self-edge after, as the figure shows.

You will be judged on the speed of your code, as well as its correctness.