CS 61B – Summer 2005 – (Porter)

Final Exam Part 2 – Aug 11, 2005

Do not open until told to begin

This exam is CLOSED BOOK, but you may use 2 letter-sized page of notes that you have created.

Problem 0: (1 point) Please fill out this information, and when told to begin, please put your name **on each page** of the exam.

Your name:	SOLUTION
Your cs61b login:	
Lab time:	
Lab TA's name:	
Name of person to your left:	
Name of person to your right:	

Do not write below here:

Problem	**Score**	**Total possible**
0		1
1 (10 minutes)		9
2 (15 minutes)		25
3 (15 minutes)		20
4 (20 minutes)		25
Total:		**80**

Do not open until told to begin

Good Luck!

Name: ______________________________________

Tree Representation:

Unless otherwise indicated, all tree data types consist of a single instance variable: a reference to the root of the tree. Nodes in the tree are of the BinaryNode class (covered in lecture):

```
class Tree {
     private BinaryNode root;

     // constructors and methods
}

class BinaryNode {
     public Comparable element;
     public BinaryNode left;
     public BinaryNode right;

     // constructors and methods
}
```

Extra Credit (1 point each):

Between 1984 and 1999, Edsger Dijkstra held a professorship at what US University?

The University of Texas at Austin

The Association of Computing Machinery, ACM, is one of the two major computing professional societies. Which UC Berkeley professor is the President of the ACM?

David Patterson

Problem 1 (9 pts) – Family Tree – (10 minutes)

Consider a family tree consisting of Person nodes:

```
public class Person {
        public Person [] children;

        // other methods
}
```

We want to easily and quickly find out whether two people are cousins. What additional instance variables would you add to the Person class to make this task easier? How would you make use of your additional field or fields? You do not have to provide Java code, but give the types of your fields and be specific about how you would make use of them.

If you add a parent reference to each Person object, then you can determine if person A and person B are cousins by comparing A.parent.parent == B.parent.parent. (Cousins share the same grandparents). Two people who have the same parents (A.parent == B.parent) are siblings, not cousins, and so the test would be:

(A.parent.parent == B.parent.parent) && (A.parent != B.parent)

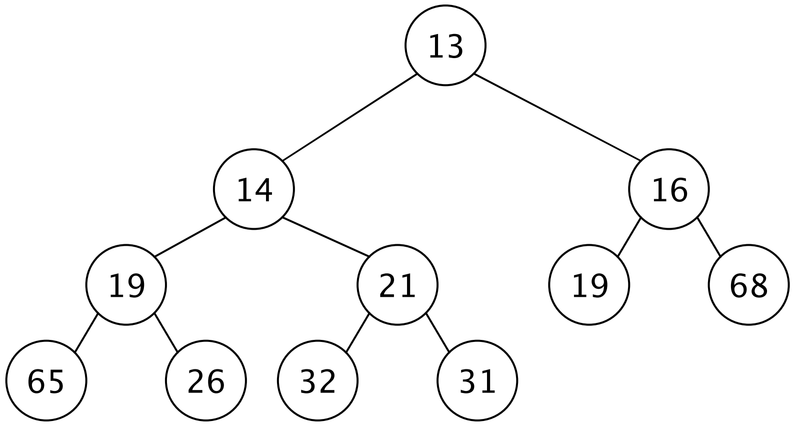
Another solution could add a "Mom" and "Dad" reference to each Person. The comparison would be the same, but you would have more cases (If A.Mom.Mom == B.Mom.Mom...)

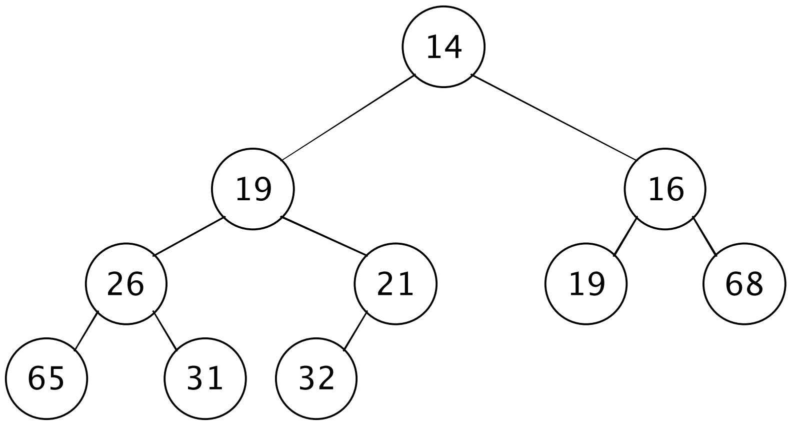
Name: ____

Problem 2 (25 pts) – Heaps – (15 minutes)

In this problem all heaps are assumed to contain the smallest items on top.

Show the heap that results from calling removeMin() on the following heap:





Removing the minimum item from a heap with **N** elements (given no other information) requires, on the average,

a. a constant number of comparisons
b. order log N comparisons
c. order N comparisons
d. order N log N comparisons
e. order N^2 comparisons

Finding the *largest* item in that heap requires at worst

a. a constant number of comparisons
b. order log N comparisons
c. order N comparisons
d. order N log N comparisons
e. order N^2 comparisons

Name: ____

Describe an algorithm that uses a heap of size k+1 to collect the k largest elements in an unsorted list of length N in time proportional to N log k. Your description should be in detail sufficient for another CS 61B student to recognize how your solution should be implemented in Java. You DO NOT HAVE TO WRITE CODE, but *be specific.*

Insert the first k items from the list into the heap. Now, for the rest of the unsorted list, we process each entry E as follows. Compare E to the top element of the heap. If it is smaller, we throw it away, and go to the next element. If it is larger, then we call removeMin() on the heap, then insert E. Continue this process until you have processed each element of the list. The heap now contains the largest k elements.

Problem 3 (20 pts) – Balanced Trees – (15 minutes)

A programmer on your staff presents you with the following algorithm to create a balanced binary search tree out of a singly linked sorted list of N elements in order N operations. The algorithm assumes that each node in the list contains only the list element and the reference to the next list element, and that each node in the tree contains the corresponding tree element and two child references. You can assume each element of the list is unique.

Algorithm

1. Find (in the obvious way) the node of the sorted list that contains the median element, and make it the root of the balanced tree. The median element of a list of size N is the N/2th element of that list.
2. Make a balanced tree—using this algorithm—out of the list of elements that precede the median, and make that tree the left subtree of the root.
3. Make a balanced tree—using this algorithm—out of the list of elements that follow the median, and make that tree the right subtree of the root.
4. Return a reference to the root of the tree.

Is the programmer's claim correct? That is, does the algorithm described above run in linear time, and does it produce a balanced binary search tree? Justify your answer.

It produces the correct tree, but the time isn't linear. What is the running time?

Finding the median of a sorted list (knowing its length is N) takes N/2 comparisons. So to find the median of the list takes N/2 comparisons. Finding the median of each of those lists is N/4 comparisons. Then N/8, etc., This can be done log N times. Thus we find this algorithm takes O(N log N) time.

Draw a tree with 8 nodes that has both the Binary Search tree property and the AVL property:

There are a variety of answers for this problem. A balanced binary search tree will work.

Problem 4 (25 pts) – Negative edge-weight Graphs – (20 minutes)

A) Give an example when Dijkstra’s algorithm gives the wrong answer in a graph with a negative edge but no negative-cost cycle (make your graph contain at least four nodes, signify one node as the “source”, and make sure that each node is reachable from the source)

This was a homework problem. I also gave an example in class. There are also examples in the book.

B) Show the result of applying the Bellman-Ford algorithm to your graph. Clearly mark the final distance computations for each node, and show the steps. To show the steps, it is sufficient to list the contents of the “in consideration” queue at each step

See the handout I gave in class that shows the steps in Bellman-Ford.