CSE 326 Final Exam 6/7/07

Name _____ Section _____

Do **not** write your id number or any other confidential information on this page.

There are 14 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, etc. You may use a calculator if you find it helpful, but only for simple arithmetic.

Please wait to turn the page until everyone is told to begin.

Score _____ / 100

1. _____ / 10

2. _____ / 6

3. _____ / 8

4. _____ / 4

5. _____ / 4

6. _____ / 4

7. _____ / 4

8. _____ / 12

9. _____ / 8

10. _____ / 12

11. _____ / 6

12. _____ / 4

13. _____ / 11

14. _____ / 7

Question 1. (10 points) Give a big-O **worst-case upper bound** on the running time of the following operations or algorithms. No explanation is needed, but some may be helpful in assigning partial credit. You should assume a reasonable implementation of any underlying data structures (i.e., if direct access to elements of a list is needed, assume that an array is used instead of a linked list, or assume an adjacency list instead of an adjacency matrix if that makes a graph algorithm faster, etc.)

(a) Find in an AVL tree of size *n*.

(b) Find in a splay tree of size *n*.

(c) Find (set membership) in a union-find uptree that uses weighted unions and path compression, where there are *n* total elements in all of the sets involved.

(d) Compute all shortest paths from a given vertex in a graph with V vertices and E edges using Dijkstra's algorithm. Assume that priority queues are used in addition to the basic graph data structure(s) if these are helpful.

(e) Compute all shortest paths from a given vertex in a graph with V vertices and E edges using Dijkstra's algorithm if no priority queues are used as auxiliary data structures.

Question 2. (6 points) The algorithm zoo. We've discussed many algorithms during the course, including several that illustrate particular design strategies. For each of the following, name (or describe) one algorithm that we've studied (or that you know) that uses the given strategy. Be sure to name or describe a particular algorithm and not just state a problem that can be solved by many different algorithms.

(a) Divide and conquer

(b) Greedy

(c) Dynamic programming

Question 3. (8 points) Circle true or false for each of the following:

(a) Linear probing will always succeed in finding a location to store a value in a hash table if the load factor λ is less than 1.0.

True False

(b) Quadratic probing will always succeed in finding a location to store a value in a hash table if the load factor λ is less than 1.0.

True False

(c) In a properly designed Java class, if two instances `x` and `y` have the property that `x.equals(y)` is true, then for these objects to work properly when stored in collections like maps and sets, it must be true that `x.hashCode()==y.hashCode()`.

True False

(d) In a properly designed Java class, if two instances `x` and `y` have the property that `x.hashCode()==y.hashCode()`, then for these objects to work properly when stored in collections like maps and sets, it must be true that `x.equals(y)`.

True False

Question 4. (4 points) In a hash table that uses *separate chaining* (i.e., each bucket in the hash table points to a list of elements that hash to that bucket), we should rehash the table to preserve fast access to elements when the load factor (λ) gets too large. What is a reasonable value for λ at which rehashing should occur to preserve good performance without spending an excessive amount of time rehashing the table? Give a brief justification for your answer.

Question 5. (4 points) Ben Bitblitter has gone down to Al's Algorithm Emporium to pick up a fancy new algorithm to use in a program that he is working on. There are two algorithms available that would work. Algorithm I guarantees an **amortized** time of $O(\log n)$ per operation. Algorithm II guarantees an **average** time of $O(\log n)$ per operation. Are these two claims equivalent? If not, which one is a better (faster) guarantee? Give a brief justification for your answer.

Question 6. (4 points) Two of your colleagues are having an argument about implementing an effective hash function for character strings. To compute a hash value for a string with n characters $c_1c_2c_3\ldots c_n$, one of your colleagues says that we should treat the characters as integers and compute the value $(\ldots(((c_1*37)+c_2)*37+c_3)*37+\ldots)*37+c_n$. The other colleague says that this is unnecessarily complicated and that we should just add up the character values $c_1+c_2+c_3+\ldots+c_n$. This colleague claims that the second formula is faster to compute and just as good as a hash function.

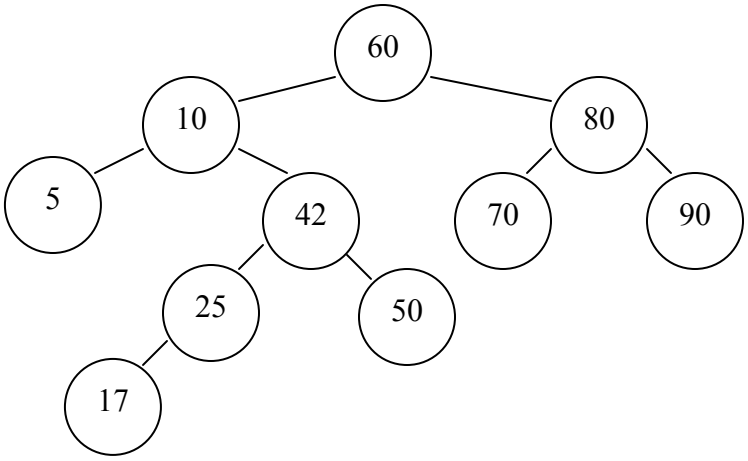
Which formula is better as a hash function and why?

Question 7. (4 points) Quicksort has an expected sorting time of $O(n \log n)$ provided that the pivot value is chosen with some care.

(a) What needs to be true about the choice of pivot value to ensure that quicksort has an expected time of $O(n \log n)$?

(b) Give one effective strategy for picking pivot values that will ensure that the condition needed in part (a) will usually be met to ensure that quicksort runs in $O(n \log n)$ time.

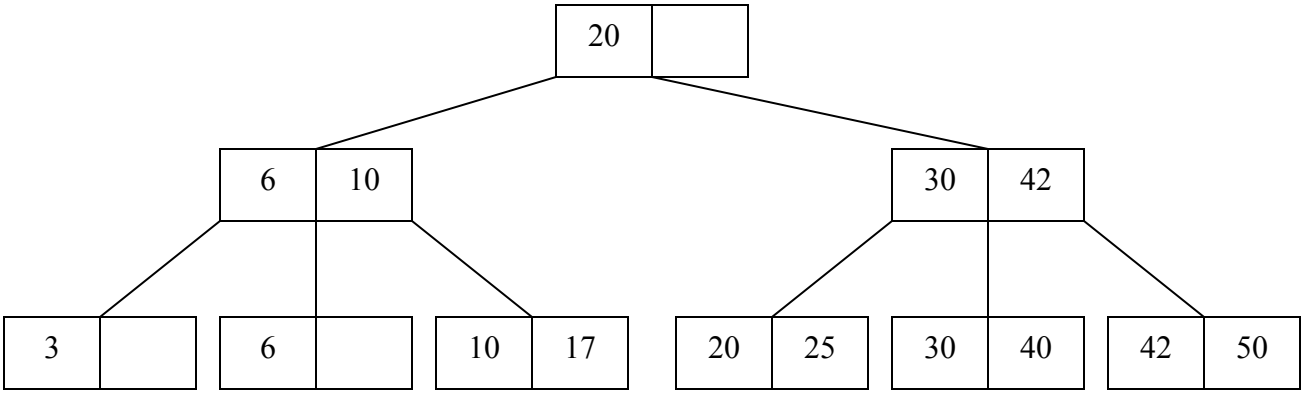
Question 8. (12 points) Consider the following splay tree.



(a) Draw the modified splay tree that results after we perform a find operation on 17 in the above tree.

(b) One of your colleagues claims it would simplify and improve things if, instead of the complex zig-zag and zig-zig rotations in a classic splay tree, we just used a sequence of single rotations to move the found node to the root in a splay operation. Can we make this change and still expect to get amortized $O(\log n)$ time for splay tree operations? Give a brief justification for your answer

Question 9. (8 points) Consider the following *M*-ary B-tree, where $M = 3$ and the number of data items in a leaf $L = 2$.



Draw the modified tree that results if 35 is inserted in the above B-tree.

Question 10. (12 points) The uptrees used to represent sets in the union-find algorithm can be stored in two *n*-element arrays: one giving the parent of each node (or -1 if the node has no parent), and the other giving the number of items in a set if the node is the root (representative node) of a set. For example, we can represent a collection of sets containing the numbers 1 through 14 as follows:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
up	-1	1	2	5	2	-1	1	6	8	6	-1	8	11	11
weight	6	-	-	-	-	5	-	-	-	-	3	-	-	-

(a) Draw a picture of the uptrees represented by the data in the above arrays.

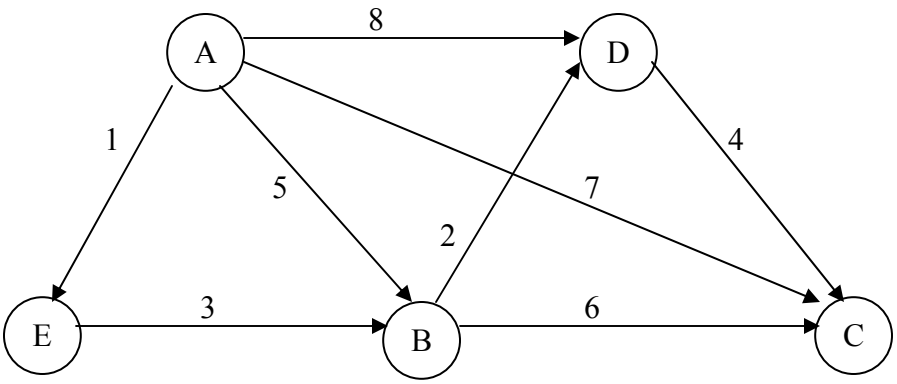
(b) Now, **draw** a new set of uptrees and **update** the data arrays as needed to show the results of executing the following two set operations:

```
union(find(13), find(8));
find(4);
```

You should assume that the find operations use **path compression** and that the union operation uses **union-by-size**. Remember to update the numbers in the data array tables at the top of the page as well as draw the updated trees below.

Das Graph

The following graph is the subject of the next several questions.



(Feel free to tear out this page and use it when answering the questions on the following pages.)

Remainder of this page intentionally left blank.

Question 11. (6 points) Give both the adjacency matrix and adjacency list representations of the above graph. Be sure to specify which is which.

Question 12. (4 points) Give two different valid topological orderings of the nodes in the above graph if possible. If fewer than two different orderings exist, list any that do exist and give an informal argument why there are fewer than two orderings possible.

Question 13. (11 points) (a) Step through Dijkstra's algorithm to calculate the single-source shortest paths from A to every other vertex. You only need to show the final table, but to help us award partial credit, you should show your steps in the table below. Cross out old values and write in new ones as the algorithm proceeds.

In addition, list the vertices here in the order in which Dijkstra's algorithm marks them as known:

Known vertices (in order discovered): _____ _____ _____ _____ _____

Vertex	Known	Distance	Path
A			
B			
C			
D			
E			

(b) What is the shortest weighted path from A to D in the graph, as computed above by Dijkstra's algorithm?

(c) What is the length (weighted cost) of the shortest path you gave as your answer to part (b)?

Question 14. (7 points) Now suppose that the edges in the graph used in the previous questions were undirected, but with the same weights as given above, and assume we run Kruskal's algorithm on this undirected graph to compute a minimum spanning tree. Write down the edges in the order they are considered by Kriskal's algorithm. If the edge is part of the minimum spanning tree found by the algorithm, write it down in the first list of edges that form the MST. Write down the other edges considered by the algorithm in the order they were considered in the second list. Assume that the algorithm terminates as soon as the MST has been found.

In the lists, use (x,y) to indicate an edge connecting vertices x and y.

Edges that form part of the MST, in order:

Other edges considered, but not included in the MST, if any, in order: