Fundamental Algorithms, Final Examination, May 6, 1998

Explain your reasoning for each answer. An answer, such as "$O(N^2)$", or "true", even if correct, may get no credit if there is no explanation. The phrase "sketch an algorithm" asks you to list the steps of the algorithm in some kind of code or pseudocode.

1. A credit card company has 50,000 active accounts, each with a unique 12 digit credit card number. For the month of April, it has 70,000 records of credit card transactions. Each transaction record has a field, `record.number`, which is an array of 12 characters, one for each digit in the credit card number for the transaction. Each record also has a field, `record.amount`, that records the dollar amount of the transaction. We want to write a program that determines the total of the amounts for each account in April. For example, if account number `2345-6789-1223` has two transactions for \$3 and \$50 respectively, then the total for `2345-6789-1223` is \$53.

- **a.** Sketch an algorithm that would do this by hashing the credit card numbers from transaction records. What would be a reasonable size for the hash table? Assume the hash function is `int hf(char num[12])`. Use closed hashing to resolve collisions. What other arrays are needed for this task?
- **b.** Sketch an algorithm that would do this by sorting the account numbers in the transactions. Consider the sorting algorithms insertion-sort, quicksort, and radix-sort. Choose one of these for your algorithm. Explain your choice.

2. In each case, state whether the assertion is true or false. Briefly explain your answer. To prove that something need not always happen, give a counterexample. For example, to prove that a graph need not be connected, just draw a graph that is not connected.

- **a.** Suppose that an algorithm performs $m = n^2$ operations on n objects, and that the amortized cost per operation is $\log(n)$. Then no single operation may take more than $O(n)$ operations.
- **b.** We have a weighted connected undirected graph, G, which has two subgraphs, G_1, and G_2. Each vertex of G is either in G_1 or in G_2. Any edge of G that connects two vertices in G_1 is in G_1, and similarly for G_2. We find a minimum spanning tree for G_1 and another for G_2. We connect these spanning trees by adding the lowest cost edge that connects G_1 to G_2. Is it true that the result is a minimum spanning tree of G?
- **c.** The numbers $a_1, \ldots, a_n$ are integers between 0 and $2n$ *with no integer repeated.* Then $S = a_1 + \cdots + a_n = \Theta(n^2)$. Hint: what are the largest and smallest possible values of S?
- **d.** Every spanning tree of a connected undirected graph has the same number of edges.
- **e.** $e^n = O(n^{\log(n)})$.
- **f.** A root of a DAG is a vertex so that doing DFS from that vertex visits all the vertices of the DAG. Is it true that every connected DAG has a root?
- **g.** We have a weighted graph and have computed the distance from each vertex to a source vertex, s. We now add one new edge. It is possible to determine in $O(1)$ time whether any of the distances decreases.

3. Sketch an algorithm that performs the following operations, which define a "split queue". There is a fraction, f, between 0 and 1, that is specified in the beginning and never changes afterwards. If there are n numbers in the queue, we define m to be the largest integer smaller than fn. The largest m numbers go into the "top queue" and the rest go into the "bottom queue". As n changes, some numbers may have to be moved from one queue to the other. The operations are

- `insert(x)`: puts x into the split queue.
- `delete_top()`: returns and deletes the smallest number in the top queue.
- `delete_bottom()`: returns and deletes the largest number in the bottom queue.

The cost for each operation should be $O(\log(n))$. You may use priority queue operations without saying how they work.

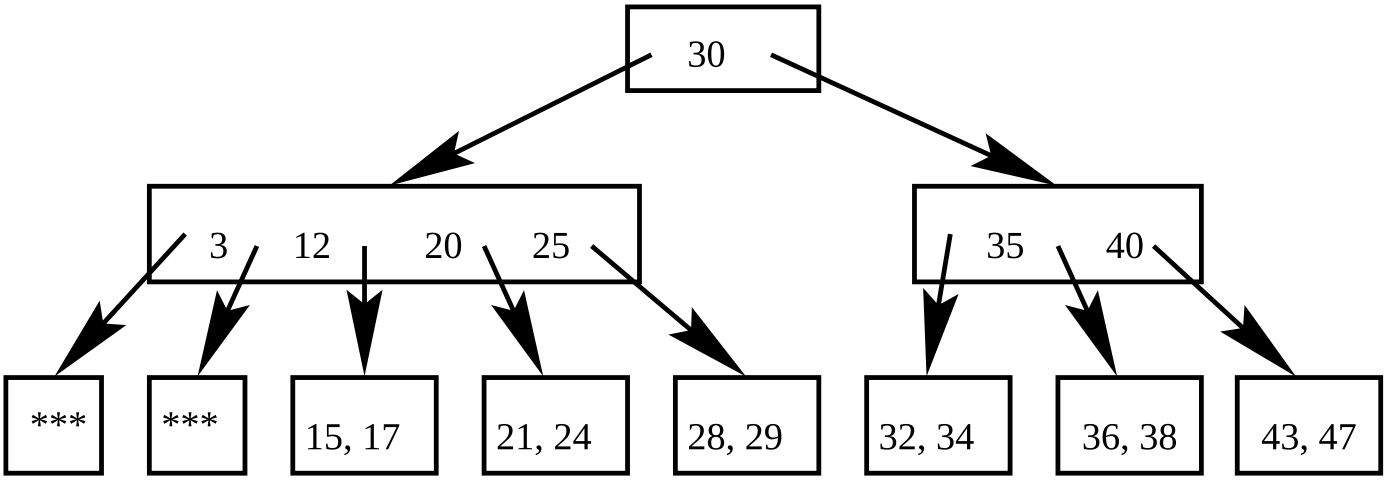
4. Consider the following pseudo C code:

```
float bp( int x, int k ) { /* Solve an optimization problem:
          k = number of levels left.
          x = the position.
          At each stage, you may leave x constant or decrement x by one
          (except that x is not allowed to become negative).  We want to
          optimize our reward after k stages.  There is a multiplier for
          each stage that depends on the position at that stage. */

   extern float p[n]; /* p[x] is the reward at the final stage if you are
                         at level x.       */
   extern float m[n]; /* m[x] is a multiplier that changes the reward if you
                         are at position x and there are some levels left.  */

   if ( k < 0 ) ERROR;
   if ( ( x < 0 ) or ( x > n ) ) ERROR;

   if ( k == 0 ) return p[x]; /* Just get the reward at the last stage.
   if ( x == 0 ) return m(0)*bp(0,k-1); /* Can´t decrement x from 0.

   return max( m(x)*bp(x,k-1), m(x-1)*bp(x-1,k-1) );

}
```

a. How much work is required to execute `bp(n,n)`?

b. Rewrite `bp` using dynamic programming to get the same answer in $O(n^2)$ work.

5. Below is a B tree with parameter $t = 2$. Draw the results of the following operations, each starting with the same tree: Insert 23, Delete 43, Delete 30.



6. An undirected graph is "bipartite" if the vertices can be divided into two subsets, L and R, so that there is no edge connecting any vertex of L to any other vertex of L, and similarly for R. Sketch an algorithm that is a modification of DFS that determines whether a graph, G is bipartite. We may arbitrarily assign the first vertex to the L set. Then, if the graph is bipartite, the assignments of all the other vertices in the connected component are determined.