CS570
Analysis of Algorithms
Spring 2007
Exam 1

Name: ______________________
Student ID: __________________

	Maximum	Received
Problem 1	14	
Problem 2	6	
Problem 3	15	
Problem 4	20	
Problem 5	15	
Problem 6	20	
Problem 7	10	

Note: The exam is closed book closed notes.

1) 14 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]False
Function $f(n)= 5n^3 2^n + 6n^2 3^n$ is $O(n^3 2^n)$.

[TRUE/FALSE] True
$n\log^3 n$ is NOT $O(n\log n)$

[TRUE/FALSE] True
In an undirected graph, the shortest path between two nodes lies on some minimum spanning tree.

[TRUE/FALSE] False
Max base heap can be converted into Min based heap by reversing the order of the elements in the heap array.

[TRUE/FALSE] False
Kruskal's algorithm for finding a MST of a weighted, undirected graph is a form of a dynamic programming technique.

[TRUE/FALSE] True
In the case of applying Dijkstra's algorithm for dense graph, Fibbonacci implementation is the best one among Binary, Binomial, and Fibbonacci.

[TRUE/FALSE] False
If all of the weights from a connected and weighted graph are distinct, then distinct spanning trees of the graph have distinct weights.

2) 6pts

What is the worst-case complexity of the each of the following code fragments?

a)
```
for (i = 0; i < 2N; i++) {
      sequence of statements
     }
for (j = 0; j < 3M; j++) {
    sequence of statements
     }
O(M+N)
```

b)
```
for (i = 0; i < N; i++) {
          for (j = 0; j < 2N^2; j++) {
              sequence of statements
          }
      }
      for (k = 0; k < 3M; k++) {
          sequence of statements
      }
      O(N^3+M)
```

c)
```
for (i = 0; i < N^3; i++) {
          for (j = i; j < 2lg(M); j++) {
               sequence of statements
          }
      }
      for (k = 0; k < M; k++) {
               sequence of statements
             }
```

$O(N^3 \log M+M)$

3) 15 pts
The maximum spanning tree problem is to find a spanning tree of a graph whose edge weights have the largest sum possible. Give an algorithm to find a maximum spanning tree and give a proof that the algorithm is correct.
Solution: We can reduce finding the maximum spanning tree problem into finding minimum spanning tree problem. The reduction is as follows: Let w(e) denote the weight of edge e in the graph and let M be the maximum weight of all the edges in G. Now consider re-weighting each edge e in G as M-w(e). Now consider two spanning tree of G, T and T' and note that they both contain exactly the same number of edges, i.e., n-1 edges, where G has n nodes. The key point is that $\sum_{e \in T} w(e) \geq \sum_{e' \in T'} w(e')$ if and only if

$$\sum_{e \in T}[M - w(e)] = (n-1)M - \sum_{e \in T} w(e) \leq (n-1)M - \sum_{e' \in T'} w(e') = \sum_{e' \in T'}[M - w(e')]$$

Hence, if we solve the minimum spanning tree problem using the new edge weights, the optimal tree found will be the maximum spanning tree using the original weights. Therefore we can reduce the maximum spanning tree problem into the minimum spanning tree problem. Hence we can use either Kruskal's algorithm or Prim's algorithm after the reduction and the running time is

O(mlog n)

4) 20 pts
Suppose you are given two set *A* and *B*, each containing *n* positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the *i*-th element of set *A*, and let b_i be the *i*th element of set *B*. You then receive a payoff of $\sum^{n}_{i=1} b_i \log a_i$. Give an algorithm that will maximize your payoff. Prove that your algorithm maximizes the payoff, and state its running time.

Algorithm: Sort A and B in increasing order and the payoff is maximized

Proof: Suppose $\{a_1,\ldots,a_n\}$ and $\{b_1,\ldots b_n\}$ is an optimal solution but ($a_i>a_j$) and ($b_i<b_j$). Then switch a_i and a_j we can get a better solution. The reason is as follows:

$$a_i > a_j, b_i < b_j \Leftrightarrow a_i^{b_j-b_i} > a_j^{b_j-b_i} \Leftrightarrow a_i^{b_j} a_j^{b_i} > a_j^{b_j} a_i^{b_i} \Leftrightarrow \log(a_i^{b_j} a_j^{b_i}) > \log(a_j^{b_j} a_i^{b_i})$$
$$\Leftrightarrow b_j \log a_i + b_i \log a_j > b_j \log a_j + b_i \log a_i$$

which contradicts that $\{a_1,\ldots,a_n\}$ and $\{b_1,\ldots b_n\}$ is an optimal solution

Running time: Sorting takes O(nlog n). Hence the running time of our algorithm is O(nlog n)

5) 15 pts

Given a connected graph G = (V,E) with positive edge weights and two nodes s,t in V, prove or disprove:

a. If all edge weights are unique, then there is a single shortest path between any two nodes in V.

b. If each edge's weight is increased by *k*, the shortest path cost will increase by a multiple of *k*.

c. If the weight of some edge *e* decreases by *k*, then the shortest path cost will decrease by at most *k*.

a) False. Counter Example: (s,a) with weight 1, (a,t) with weight 2 and (s,t) with weight 3. There are two shortest path from s to t though the edge weights are unique.

b) False. Example: suppose the shortest path *s →t* consist of two edges, each with cost 1, and there is also an edge *e=(s,t)* in G with cost(e)=3. If now we increase the cost of each edge by 2, *e* will become the shortest path (with the total cost of 5).

c) True. For any two nodes s,t, assume that $P_1,\ldots,P_k$ are all the paths from s to t. If e belongs to P_i then the path cost decrease by k, otherwise the path cost unchanged. Hence all paths from s to t will decrease by at most k. As shortest path is among them, then the shortest path cost will decrease by at most k.

Grading policy: 5 points for each item. 2 points for TRUE/FALSE part and 3 points for prove/disprove part.

6) 20 pts
Sam is shocked to find out that his word processor has corrupted his text document and has eliminated all spacing between words and all punctuation so his final term paper looks something like: "anawardwinningalgorithmto…". Luckily he has access to a Boolean function *dictionary( )* that takes a string *s* and returns true if *s* is a valid word and false otherwise. Considering that he has limited time to turn in his paper before the due date, find an algorithm that reconstructs his paper into a sequence of valid words with no more than O(n^2) calls to the *dictionary( )* function.

Solution: We denote that the string as s[1],….,s[n]. For the sub string s[1],….,s[m], we construct the table P as follows: if s[1],….,s[m] is composed of a sequence of valid words, then P[m]=true; otherwise P[m]=false. The recursive relation is as follows:

$$P[0] = true$$

$$P[m] = \vee_{0 \le i \le m-1} (P[i] \wedge dictionary \ \ (s[i+1],..., \ s[m]))$$

If P[m] is true, we denote q[m]=j where j is such that $P[j] \wedge dictionary(s[j+1],..., s[m])$ is true. By the definition of P[m], it is obvious that such a j must exist when P[m] is true. To reconstruct the paper into a sequence of valid words, we just need to output n, q[n],q[q[n]],…,0 as segmentation points.

Running time: The length of table of P and q is n and each step when we compute P[m] we need at most m calls of function dictionary(). Note the $m \le n$. Hence the running time is bounded by O(n^2)

Remark: Actually this question is exactly the same as the first problem in HW4: Problem 5 in Chapter 6 if we set $quality(Y_{i+1}, k) = 0$ if $dictionary(Y_{i+1}k) = \ false$ and $quality(Y_{i+1}, k) = 1$ if $dictionary(Y_{i+1}k) = true$.

7) 10 pts

An $n \times n$ array A consists of 1's and 0's such that, in any row of A, <u>all the 1's come before any 0's in that row.</u> Give the most efficient algorithm you can for counting the number of 1's in A.

Algorithm: For each row i, we use binary search to find the index of last element of 1 in the array $A_i[1,\ldots.n]$. Suppose the index is a_i. Then the sum of $a_i(0<i<n+1)$ is the number of 1's in A.

Proof: Assume that in i-th row k is the index of last element of 1. Then we have the property that $a_{ij}=1$ for $j<k+1$ and $a_{ij}=0$ for $j>k$ as all 1's come before any 0's in each row. During the binary search, if $a_{im}=1$, then we must have $k\geq m$; $a_{im}=0$, then we must have $k<m$. By doing this, we can finally find the value of k for each row.

Running time: There are n rows and for each row the running time to find the last element which is 1 by binary search is log n. Hence the running time of our algorithm is O(nlog n)

Grading policy: 5 points for algorithm. 3 points for proof and 2 points for running time.

Additional Space

Additional Space