CS570
Analysis of Algorithms
Summer 2011
Exam I

Name: _____

Student ID: _____

____Check if DEN student

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Problem 5	15	
Problem 6	15	
Problem 7	20	
Total	100	

2 hr exam
Close book and notes

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
If single digit multiplication can be done in O(1) time, then multiplying k-digit numbers can be done in O(k logk) time.

Solution: True. Use divide and conquer.

[TRUE/FALSE]
Consider a directed graph G in which every edge has a positive edge weight. Suppose you create a new graph G' by replacing the weight of each edge (w_e) by ($-w_e$). For a given source vertex s, you compute all shortest paths from s in G' using Dijkstra's algorithm. The resulting paths in G' are the longest (i.e., highest cost) simple paths from s in G.

Solution: False. Dijkstra's algorithm may not necessarily return the minimum weight path because this new graph may contain a negative weight cycle.

[TRUE/FALSE]
A spanning tree of a given undirected, connected graph G(E,V) can be found in O(|E|) time, where E is the set of edges and V is the set of vertices in G.

Solution: True. Perform a walk on the graph starting from an arbitrary node.

[TRUE/FALSE]
The recurrence equation T(n)=2T(n/2)+3n has the solution T(n)= $\theta(\lg(n^2) * n)$

T(n)= theta(lg(n^2) * n)
T(n)=2T(n/2)+3.n

Solution: True.
Theta (n lg n^2) = Theta n(lg n)

[TRUE/FALSE]
Given n integers $a_1, \ldots, a_n$, the third smallest number among $a_1, \ldots, a_n$ can be computed in O(n) time.

True. Do a linear scan and remember the three smallest numbers seen so far. Whenever you encounter a new number, one can figure out in constant time, if it should displace any of the current three minimum guys. At the end of the linear scan, output the third smallest number. (The running time is O(n) as the algorithm spends only O(1) time per element.)

[TRUE/FALSE]
Given a connected graph G, with at least two edges having the same cost, there will

be at least two distinct minimum spanning trees in G.

Flase.

[TRUE/FALSE]
Given a graph G(V,E) with more than one minimum spanning tree in this graph and a tree T. If tree T contains all vertices of graph G, and each edge of T belongs to some MST of G, then T is a MST of G.

False.

[TRUE/FALSE]
If function f, g, and h are positive increasing functions with f in O(h) and g in Ω(h), then the function f + g must be in θ(h)

False.

[TRUE/FALSE]
BFS can be used to find the shortest path between any two nodes in a weighted graph.

False.

[TRUE/FALSE]
The Fibonacci heap has a better worst-case time complexity for the decrease_key operation than the binary heap.

True.

2) 10 pts

Suppose we are to receive a set of n jobs, each with a deadline, and we need to schedule them on a single machine. Whenever a job finishes on the machine, we must begin the available job with the earliest deadline. But we may receive the jobs in any order and at any time, and we may not begin a job before we receive it. Present an implementation for this algorithm that runs in O(n log n) total computation time for the scheduling. Argue carefully that your algorithm meets this time bound.

Keep jobs in a Priority Queue with key value = deadline (Assuming a binary heap implenetaionation of the priority queue)
New job, insert into queue O(lg n)
Submit job, extract min from queue O(lg n)

N jobs: Complexity = O (n lg n)

3) 10 pts

Suppose I am given a labeled undirected graph, in the form of an adjacency list. This graph has a vertex for each of the n towns in California, and there is an edge from vertex A to vertex B if you can drive from town A to town B directly, without passing through another town. Each edge is labeled by its length in miles.

We need to place a sign in each town indicating the distance from it to Los Angeles by the shortest route along the edges. Describe an algorithm to find this distance for each town, and give a big-O bound on this algorithm's running time in terms of n.

We carry out the Dijkstra algorithm for the single-source shortest path problem. (You are not required to know the name of this algorithm if you describe it in enough detail to show that you understand it.) We maintain a set of nodes S for which we know the shortest distance. Initially S = {Los Angeles} and the distance from Los Angeles to Los Angeles is 0. We have a priority queue where each element is a town with the best distance found so far (the length of the shortest path that stays in S until its last edge). We repeatedly take the front element F from the queue, add town F to S (noting its distance to Los Angeles for the sign), and then process the edges out of F. Processing an edge (F,G) means looking at the path formed by appending that edge to the path from Los Angeles to F and computing the total distance. If we have no entry yet in the priority queue for G, we make one with the new distance. If there is such an entry, we compare the distance on it with the new distance and do a change-key operation if the new distance is smaller.

We need to carry out n extract-min operations on the queue, one for each element, and up to m processings of edges. Each of these operations takes O(log n) time, so our total time is O((n+m)(log n)). Note that we need to

keep a pointer into the priority queue for each town, so that we can retrieve the queue element to do the change-key if needed. This pointer can also tell us if the endpoint of the new edge is already in S, or not yet in the queue.

4) 10 pts
Consider the following algorithm strangeSort, which sorts n `distinct` items in a list A.

1. If $n \leq 1$, return A unchanged
2. For each item x in A, scan A and count how many other items in A are less than x
3. Put the items with counts less than n/2 in a list B
4. Put the other items in a list C
5. Recursively sort B and C using strangeSort
6. Append the sorted C to the sorted B and return the result

Formulate a recurrence for the running time T(n) of strangeSort on an input list of size n. Solve this recurrence to get the best possible big-O bound on T(n).

Clearly if $n \leq 1$ the list is already sorted, and thus line 1 returns the sorted version of the list as desired. Assume now that strangeSort sorts all lists of length n/2, with smaller items first. Consider the operation of strangeSort on a list of size n. In line 2, each item is assigned a number from 0 through n-1, and in line 3 the smallest n/2 items are put in B. The two lists B and C thus each have n/2 items, so by the inductive hypothesis the recursive calls sort B and C correctly. Since every item in B is smaller than every item in C, the append operation in line 6 creates a sorted list of n elements. So the inductive step is complete, assuming that n is a power of 2.

Line 1 means that $T(1) = O(1)$. Step 2 requires n scans of the entire list of n elements and so takes $O(n^2)$ time. Lines 3 and 4 each move n/2 items and so take O(n) time. Line 5 makes two calls to strangeSort with arguments of size n/2 and so takes time 2T(n/2). Line 6 also takes O(n) time. The total time is thus $2T(n/2) + O(n^2) + O(n) = 2T(n/2) + O(n^2)$.

This recurrence was solved to $T(n) = O(n^2)$ in the book -- we repeat the derivation here:

- $T(n) = 2T(n/2) + O(n^2)$
- $T(n) \leq 2T(n/2) + cn^2$ (for some c)
- $T(n) \leq 4T(n/4) + 2c(n/2)^2 + cn^2$
- $T(n) \leq 8T(n/8) + 4c(n/4)^2 + 2c(n/2)^2 + cn^2$
- ...

- $T(n) \leq nT(1) + cn^2[1/n + 2/n + ... + 1/4 + 1/2 + 1]$
- $T(n) \leq O(n) + 2cn^2$
- $T(n) = O(n^2)$

They can also use the Mater theorem to show that the complexity is $O(n^2)$.

5) 15 pts
Part of the preference list of 4 men (denoted by A, B, C and D) and 4 women (denoted by a, b, c and d) are given below. The content of the box (A,a), which is (1,3), means that man A ranks woman a as his first choice, and woman a ranks man A as her third choice.

Men/Women	a	b	c
A	(1,3)	(2,2)	(3,1)
B	(3,1)	(1,3)	(2,2)
C	(2,2)	(3,1)	(1,3)

a) List ALL possible stable matchings of these men and women.

(A,a), (B,b), (C,c), (D,d)

(A,b), (B,c), (C,a), (D,d)

(A,c), (B,a), (C,b), (D,d)

b) If GS algorithm is used. And men propose. What is the resulting stable matching?
(A,a), (B,b), (C,c), (D,d)

c) If GS algorithm is used. And women propose. What is the resulting stable matching?
(A,b), (B,c), (C,a), (D,d)

6) 15 pts
You are given a sorted array of n positive integers
A[1] < A[2] < ... < A[n]. Your job is to determine if there exists an array index k, such that k = A[k].

Give a detailed divide and conquer algorithm for the Boolean function that returns FALSE if no such k can be found, and returns TRUE if such an index exists. The complexity of this function must be O(log n).

Answer:

```
Function (A, n)

    if (i == floor (n/2))

        return TRUE

    if (n == 1) and (A[i]! = i)

        return FALSE

    if (A[i] < i)

        return Function(A[i + 1 : n], n − i)

    if (A[i] > i)

        return Function(A[1 : i], i)
```

Proof: The algorithm is based on Divide and Conquer. Every time we break the array into two halves.
If the middle element i satisfy A[i] < j, we can see that for all j < i, A[j] < j. This is because A is a sorted
array of DISTINCT integers. To see this we note that A[j + 1] − A[j] >= 1 for all j. Thus in the next round
of search we only need to focus on A[i + 1 : n] Likewise, if A[i] > i we only need to search A[1 : i] in the next
round. For complexity T (n) = T (n/2) + O(1) Thus T (n) = O(log n)

7) 20 pts

Suppose you are given of an ordered list of n words. The length of the i-th word is w_i , that is the i-th word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you cannot reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter.

a) The penalty for having a line of length K is L−K. The total penalty is the sum of the line penalties. The problem is to find a layout that minimizes the total penalty. You need to justify correctness and analyze the running time of your algorithm.

Answer:

For i= 1 to n

Place the ith word on the current line if it fits
else place the ith word on a new line

endfor

Proof:

Assume there is any other solution T , and call the output of the greedy algorithm G. First we can see
that G has no more lines than T . Because by the end of each line the number of words written down

in solution G is no less than the number of words written down in solution T . Indeed, let Nsi denote
the number of words written down in first i lines of solution s. It is obvious that $N_{G1} >= N_{T1}$. Let
j be the smallest number such that $N_{Gj} < N_{Tj}$ then word $N_{Tj-1}+1$ to word N_{Tj} can be placed in
one line. However, in solution G, words $N_{Gj-1}+1 (\geq N_{Tj-1})$ to word N_{Tj} cannot be placed in one
line. This is not possible.

This means that of all possible layouts, G given the one with the least number of lines. Also we note
that for a list of words of total length S, a layout with N lines always results in a total penalty of
N L − S. Therefore G minimize the total penalty by achieving smallest number of N .

This algorithm runs in O(n) time.

b) If the penalty for having a line of length K is $(L - K)^2$, can your algorithm given in part a) still find a layout that minimizes the total penalty? If you answer YES, give a complete proof. If you answer NO, give a specific counter example.

Answer: Suppose the line width is 100. The input is 3 words with length 50, 49, 2. If we do greedy
algorithm, the penalty is 1 + 98^2 . However, if we put the second and third word in the second line, the
penalty is 50^2 + 49^2 < 1 + 98^2 .