x

CS570
Analysis of Algorithms
Fall 2009
Exam I

Name: _____

Student ID: _____

____Monday ____Friday 2-5 ____Friday 5-8 ____DEN

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

2 hr exam
Close book and notes

If a description of an algorithm is required, please limit your description within 200 words, anything beyond 200 words will not be considered. Proof/justification or complexity analysis will only be considered if the algorithm is either correct or provided by the problem.

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

[TRUE]
Given a min-heap with *n* elements, the time complexity of select the smallest element from the *n* elements is O(1).

[FALSE] But we give credit to both true and false
Given a min-heap with *n* elements, the time complexity of select the second smallest element from the *n* elements is O(1).

[FALSE] Algorithms (e.g., bubble sort) with $O(n^2)$ time complexity could cost O(n) in some instances.
Given a problem with input of size *n*, a solution with O(n) time complexity always costs less in computing time than a solution with $O(n^2)$ time complexity.

[FALSE]
By using a heap, we can sort any array with *n* integers in O(n) time.

[TRUE] m is at most n(n-1)
For any graph with *n* vertices and *m* edges we can say that $m = O(n^2)$.

[FALSE] m could be n(n-1)
For any graph with *n* vertices and *m* edges we can say that O(m+n) = O(m).

[FALSE] Consider the following counter-example:
G(V, E). V={A,B,C}, (A,B)=2, (A,C)=1, (B,C)=1. P=AB is the shortest path between A and B, but it is not in the MST.
Given the shortest path P between two nodes A and B in graph G(V,E), then there exists a minimum spanning tree T of G, such that all edges of path P is contained in T.

[FALSE] Consider the following counter-example:
w1 prefers m1 to m2; w2 prefers m2 to m1; m1 prefer w1 to w2; m2 prefer w2 to w1
Consider an instance of the Stable Matching Problem in which there exists a man *m* and a woman *w* such that *m* is ranked last on the preference list of *w* and *w* is ranked last on the preference list of *m*, then in every stable matching *S* for this instance, the pair (*w*, *m*) belongs to *S*.

[FALSE] A graph could have multiple MSTs.
While there are many algorithms to find the minimum spanning tree in a graph, they all produce the same minimum spanning tree.

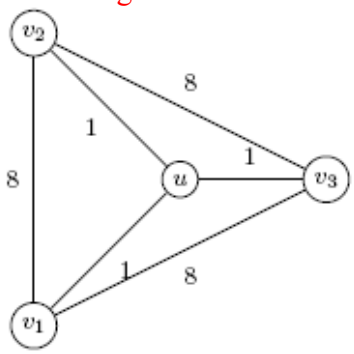
[TRUE]
A BFS tree is a spanning tree

2) 15 pts

Here is a divide-and-conquer algorithm that aims at finding a minimum spanning tree. Given a graph $G = (V, E)$, partition the set V of vertices into two sets $V1$ and $V2$ such that $|V1|$ and $|V2|$ differ by at most 1. Let $E1$ be the set of edges that are incident only on vertices in $V1$, and let $E2$ be the set of edges that are incident only on vertices in $V2$. Recursively solve a minimum spanning tree problem on each of the two subgraphs $G1 = (V1, E1)$ and $G2 = (V2, E2)$. Finally, select the minimum-weight edge in E that crosses the cut $(V1, V2)$, and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

This algorithm fails. Take the following simple graph:



Never mind how the algorithm first divides the graph, the MST in one subgraph (the one containing u) will be one edge of cost 1, and in the other subgraph, it will be one edge of cost 8. Adding another edge of cost 1 will give a tree of cost 10. But clearly, it would be better to connect everyone to u via edges of cost 1.

3) 15 pts

Suppose you are given two sets 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 ith element of set A, and let b_i be the ith element of set B. You then receive a payoff of $\prod_{i=1}^{n} a_i^{b_i}$. Give an algorithm that will maximize your payoff. Prove that your algorithm maximizes the payoff, and state its running time.

Solution:

Algorithm:
Sort A and B into monotonically decreasing order.

Proof:
Consider any indices i, j, p, q such that $i < j$ and $p < q$, and consider the terms ai^{bp} and aj^{bq}. We want to show that it is no worse to include these terms in the payoff than to include ai^{bq} and aj^{bp} , that is, $ai^{bp}\ aj^{bq} \geq ai^{bq}\ aj^{bp}$.

Since A and B are sorted into monotonically decreasing order and $i < j$, $p < q$, we have $ai \geq aj$ and $bp \geq bq$. Since ai and aj are positive and $bp - bq$ is nonnegative, we have $ai^{bp-bq} \geq aj^{bp-bq}$.

Multiplying both sides by $ai^{bq}\ aj^{bq}$ yields $ai^{bp}\ aj^{bq} \geq ai^{bq}\ aj^{bp}$.

Since the order of multiplication does not matter, sorting A and B into monotonically increasing order works as well.

Complexity:
Sorting 2 sets of n positive integers is O(nlogn).

4) 15 pts

Indicate, for each pair of expressions (A, B) in the table below, whether A is O, Ω or Θ of B. Assume that k ≥ 1, and ε > 0 are constants. Answer "yes" or "no" in each box in the following form. No explanations required.

A	B	O	Ω	Θ
$\log^k n$	n^ε	Yes	No	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No
$n^{\log m}$	$m^{\log n}$	Yes	Yes	Yes

The explanation is not required:

(1) $\log(\log n)=O(\log n)$ => $k\log(\log n)=O(\varepsilon\log n)$ => $\log(\log^k n)=O(\log n^\varepsilon)$ => $\log^k n=O(n^\varepsilon)$

(2) $-1<=\sin n<=1$, so we can not determine which one is larger;

(3) $n^{\log m}=\Theta(n^{\lg m})$, $m^{\log n}=\Theta(m^{\lg n})$, let $m=10^x$, and $n=10^y$, then $n^{\lg m}=m^{\lg n}=2^{xy}$.

5) 15 pts

Suppose you are given a number x and an array A with n entries, each being a distinct number. Also it is known that the sequence of values A[1];A[2]; ... ;A[n] is unimodal. In other words for some unknown index p between 1 and n, we have
A[1] < A[2] < ... < A[p] and A[p] > A[p + 1] > ... > A[n].
Give an algorithm with running time O(log n) to find out if x belongs to A, if yes the algorithm should return the index j such that A[j] = x. You should justify both your algorithm and the running time.

Solution: The idea is to first find out p and then break A into two separated sorted arrays, then use binary search on these two arrays to check if x is belong to A.

Let FindPeak() be the function of finding the peak in A . Then FindPeak(A [1 : n]) works as follows:
Look at A [n / 2], there are 3 cases:
(1) If A [n / 2 - 1] < A [n / 2] < A [n/2+1], then the peak must come strictly after n / 2. Run FindPeak(A [n / 2 : n]).
(2) If A [n / 2 - 1] > A [n / 2] > A [n/2+1], then the peak must come strictly before n / 2. Run FindPeak(A [1 : n / 2]).
(3) If A [n / 2 - 1] < A [n / 2] > A [n/2+1], then the peak is A [n / 2], return n/2.

Now we know the peak index(p value). Then we can run binary search on A [1 : p] and A [p +1 : n] to see if x belong to them because they are both sorted.

In the procedure of finding p, we halve the size of the array in each recurrence. The running time T(n) satisfies T (n) = T (n/2) + O(1) . Thus T (n) = O (log n). Also both binary search has running time at most O (log n), so total running time is O (log n).

Explanations:
Note that trying to get x in one shot (in one divide and conquer recursion) usually does not work. Binary search certainly cannot work because the array is not sorted. Modified binary search can hardly work either. The problem is that in each round you need to abandon half the array. However, this decision is hard to made. For example, the array is [1 2 3 4 5 -1], x=-1. When divide we have mid=3. We find A[mid-1]<A[mid]<A[mid+1]. A common but wrong practice here is to continue search only in the A[1:mid]. We can see clearly x=-1 is in the second half of the array. On the other hand you cannot throw away the first half of the array either. The counter example is [1 2 3 4 5 -1] where x=1.

One other mistake is to search p in a linear fashion. This take O(n) in the worst case.

6) 20 pts

Given a string *S* with *m* digits (digits are from 1 to 9), you are required to delete *n* (*n*<*m*) digits from *S*. After the operation, you have a string *S*' with *m*-*n* digits, let *N* denote the number that *S*' represents. Design a greedy algorithm to minimize *N*. For example, *S*="456298111", *n* = 3, after deleting 4, 5 and 6, you get the minimal N=298111. Prove the correctness of your algorithm and analyze its time complexity.

Solution:

The greedy strategy: every step we delete a digit, make sure the number represented by the rest of the digits is minimal. To achieve this, in each step, we do the following operation:
Find the first *i* such that S[*i*]>S[*i*+1], then we delete S[i]; if such *i* does not exist, which means the digits are in increasing order, then we simply delete the last digit. (*)
We call the position i "peak".

Proof by induction

(1) We first prove (*) can achieve minimal *N* when n = 1. We have
S' = $a_1a_2\ldots a_{i-1}a_{i+1}\ldots a_{n-1}$
Without loss of generality, suppose we delete S[j] rather than S[i], the we have
$S_1 = a_1a_2\ldots a_{j-1}a_{j+1}\ldots a_{n-1}$ (j!=i), and N_1 is the number represented by S_1.
If j < i, since $a_j < a_{j+1}$, we have $N < N_1$; if j > i, since $a_{i+1} < a_i$, we have $N < N_1$.
Therefore, $N < N_1$ for all j!=i.

(2) Next, we assume (*) can achieve minimal N when n=k, now we prove (*) can achieve minimal N when n=k+1.
Suppose the first deletion, we delete p', then for the rest of k deletions, we need to do (*) k times (deleting the first k peaks) in order to get the minimal result, which is denoted by N';
Let N denote the number generated by deleting the first k+1 peaks, p1, p2, …, p_{k+1}, in S.
If p'=pi (1<=i<=k+1), then N'=N; if p'<p1 or p_i<p'<p_{i+1} or p'>p_{k+1}, similar to (1), we can prove N' > N; Therefore, we show that N<=N'.

(3) With (1) and (2), we prove the correctness of the algorithm.

Complexity:
O(n), but O(mn) is also acceptable.