CS570

Analysis of Algorithms

Fall 2010

Exam I

Name: ______________________

Student ID: __________________

____Monday ____Friday ____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 to an algorithm is required please limit your description to within 150 words, anything beyond 150 words will not be considered.

1) 20 pts

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

[TRUE]

The number of spanning trees in a fully connected graph with *n* vertices goes up exponentially with respect to *n* .

[FALSE]

BFS can be used to find the shortest path between any two nodes in a weighted graph.

[FALSE]

DFS can be used to find the shortest path between any two nodes in a non-weighted graph.

[TRUE]

While there are different algorithms to find a minimum spanning tree of an undirected connected weighted graph *G*, all of these algorithms produce the same result for a given graph with unique edge costs.

[TRUE]

If $T(n)$ is $\Theta(f(n))$, then $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

[TRUE]

The array [20 15 18 7 9 5 12 3 6 2] forms a max-heap.

[TRUE]

Suppose that in an instance of the original Stable Marriage problem with n couples, there is a man M who is first on every woman's list and a woman W who is first on every man's list. If the Gale-Shapley algorithm is run on this instance, then M and W will be paired with each other.

[TRUE]

The complexity of the recursion given by $T(n) = 4T(n/2) + cn^2$, for some positive constant c, is $O(n^2 \log n)$.

[FALSE]

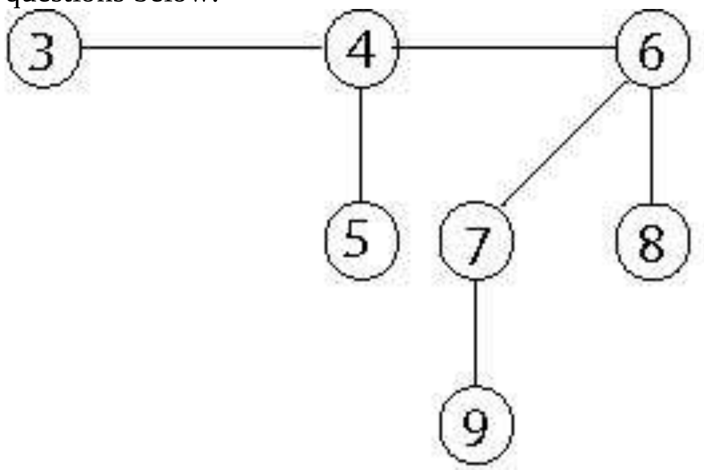
Consider the interval scheduling problem. A greedy algorithm, which is designed to always select the available request that starts the earliest, returns an optimal set A.

[FALSE]

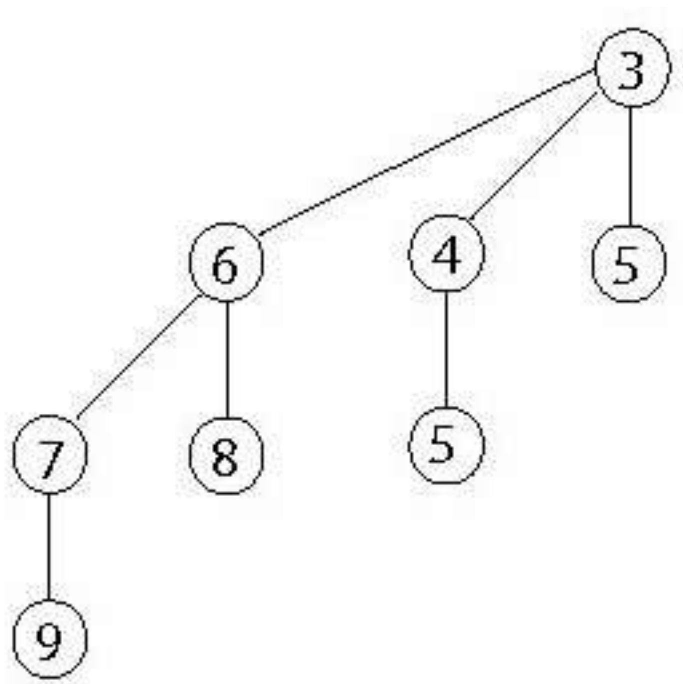
Any divide and conquer algorithm will run in best case $\Omega(n \log n)$ time because the height of the recursion tree is at least $\log n$.

2) 15 pts

You are given the below binomial heap. Show all your work as you answer the questions below.



a- Insert a new node with key value 5. Show the resulting tree and intermediate steps if any. Is the resulting heap also a binary heap and/or a Fibonacci heap?



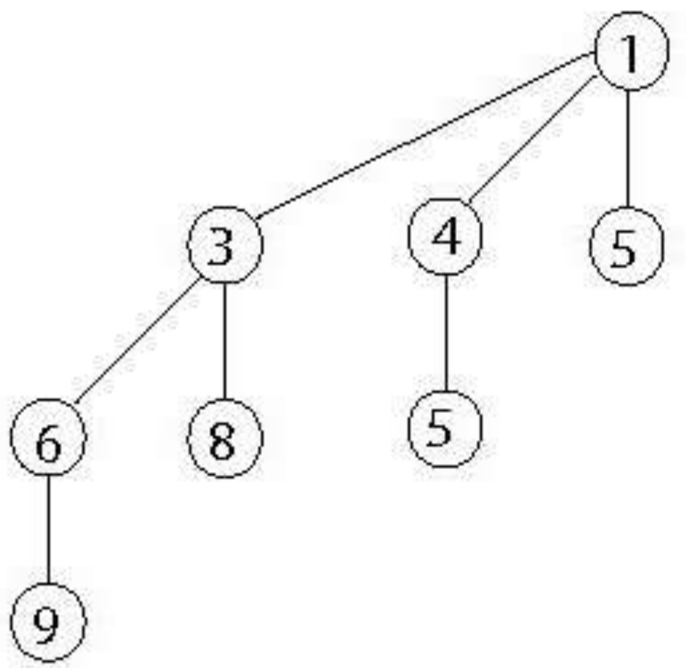
This is also a Fibonacci heap, but not binary heap.

b- Analyze the complexity of your insertion algorithm.

O(logn)

c- Now decrease the key of node 7 to 1. Is the minimum-heap property violated? If so, rearrange the heap. Show the resulting tree and intermediate steps if any.

When key value is changed to 1, min-heap property is violated since 1 is the smallest key value and it has to be at the root node. The rearrangement of key values results in the following heap.



d- Analyze the complexity of the operation in C.

O(logn)

3) 15 pts

A polygon is convex if all of its internal angles are less than 180° (and none of the edges cross each other).We represent a convex polygon as an array *V[1...n]* where each element of the array represents a vertex of the polygon in the form of a coordinate pair *(x, y)*. We are told that *V[1]* is the vertex with the minimum *x* coordinate and that the vertices *V[1...n]* are ordered counterclockwise. You may also assume that the *x* coordinates of the vertices are all distinct, as are the *y* coordinates of the vertices.

a- Give a divide and conquer algorithm to find the vertex with the maximum *x* coordinate in *O(log n)* time.

Note that for each $1 \le i < n$ either $V[i] < V[i+1]$ or $V[i] > V[i+1]$ (Such an array is called a unimodal array). The main idea is to distinguish these two cases:

1. if $V[i] < V[i+1]$, then the maximum element of $V[1..n]$ occurs in $A[i+1..n]$.
2. In a similar way, if $V[i] > V[i+1]$, then the maximum element of $V[1..n]$ occurs in $V[1..i]$.

This leads to the following divide and conquer solution (note its resemblance to binary search):

```
1 a, b ← 1, n
2 while a < b
3   do mid ← ⌊(a + b)/2⌋
4        if V[mid] < V[mid + 1]
5             then a ← mid + 1
6        if V[mid] > V[mid + 1]
7             then b ← mid
8 return V[a]
```

The precondition is that we are given a unimodal array $V[1..n]$. The postcondition is that $V[a]$ is the maximum element of $V[1..n]$. For the loop we propose the invariant "The maximum element of $V[1..n]$ is in $V[a..b]$ and $a \le b$".

When the loop completes, $a \ge b$ (since the loop condition failed) and $a \le b$ (by the loop invariant). Therefore $a = b$, and by the first part of the loop invariant the maximum element of $V[1..n]$ is equal to $V[a]$.

We use induction to prove the correctness of the invariant. Initially, $a = 1$ and $b = n$, so, the invariant trivially holds. Suppose that the invariant holds at the start of the loop. Then, we know that the maximum element of $V[1..n]$ is in $V[a..b]$. Notice that $V[a..b]$ is unimodal as well. If $V[mid] < V[mid+1]$, then the maximum element of $V[a..b]$ occurs in $V[mid+1..b]$ by case 1. Hence, after $a \leftarrow mid+1$ and b remains unchanged in line 4, the maximum element is again in $V[a..b]$. The other case is symmetric.

To complete the proof, we need to show that the second part of the invariant $a \le b$ is also true. At the start of the loop $a < b$. Therefore, $a \le \lfloor (a+b)/2 \rfloor < b$. This means that $a \le mid < b$ such that after line 4 or line 5 in which a and b get updated $a \le b$ holds once more.

The divide and conquer approach leads to a running time of $T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$.

b- Give a divide and conquer algorithm to find the vertex with the maximum *y* coordinate in *O(log n)* time.

After finding the vertex $V[max]$ with the maximum x-coordinate, notice that the y-coordinates in $V[max], V[max+1], \ldots, V[n-1], V[n], V[1]$ form a unimodal array and the maximum y-coordinate of $V[1..n]$ lies in this array. Thus the divide and conquer solution in part a can be used to find the vertex with the maximum y-coordinate. The total running time is $\Theta(\log n)$.

4) 15 pts

You are given a weighted directed graph $G = (V,E,w)$ and the shortest path distances $\delta(s, u)$ from a source vertex s to every other vertex in G. However, you are not given $\pi(u)$ (the predecessor pointers). With this information, give an algorithm to find a shortest path from s to a given vertex t in $O(|V| + |E|)$ time.

Start at u. Of the edges that point to u, at least one of them will come from a vertex v that satisfies $\delta(s, v)+w(v, u)= \delta(s, u)$. Such a v is on the shortest path. Recursively find the shortest path from s to v.

This algorithm hits every vertex and edge at most once, for a running time of $O(|V| + |E|)$.

5) 15 pts

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems of size *n* by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size *n* by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size *n* by dividing them into nine subproblems of size *n/3*, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
$T(n) = 5\ T(n/2) + c\ n$
Applying master theorm, $a=2$, $b=5$, $f(n)=c\ n$, $degree(f(n))=1$
Since $log_2\ 5 > 1$, $T(n) = O(n^{log_a\ b}) = O(n^{log 2\ 5})$

Algorithm B solves problems of size n by recursively solving two subproblems of size n-1 and then combining the solutions in constant time.
$T(n) = 2\ T(n-1) + c = 2^2\ T(n-2) + 2c + c = (\ 2^n -1\ )\ c$
$T(n) = O(2^n)$

Algorithm C solves problems of size n by dividing them into nine subproblems of size n/3, recursively solving each subproblems and then combining the solution in $O(n^2)$ time.
$T(n) = 9\ T(n/3) + c\ n^2$
Applying master theorm, $a=3$, $b=9$, $f(n)= c\ n^2$, $degree(f(n))=2$
Since $log_3\ 9 = 2$, $T(n) = O(n^2\ log\ n)$

From above three algorithms, we can see that time complexity of the third algorithm is best. Thus, we will choose algorithm C.

6) 20 pts

a- Suppose we are given an instance of the Shortest Path problem with source vertex s on a directed graph G. Assume that all edges costs are positive and distinct. Let P be a minimum cost path from s to t. Now suppose that we replace each edge cost c_e by its square root, $c_e^{1/2}$, thereby creating a new instance of the problem with the same graph but different costs.
Prove or disprove: P still a minimum-cost s - t path for this new instance.

The statement can be disproved by giving a counterexample as follows.
$G=(V, E)$; $V=\{s, a, t\}$; $E=\{(s, a), (a, t), (s, t)\}$;
$cost((s, a))=9$; $cost((a, t))=16$; $cost((s, t))=36$.
It is obvious that the minimum-cost $s - t$ path is $s - a - t$.

By replacing each edge cost c_e by its square. root, $c_e^{1/2}$, the costs become:
$cost((s, a))=3$; $cost((a, t))=4$; $cost((s, t))=6$.
Now the the minimum-cost $s - t$ path is $s - t$, not $s - a - t$ anymore.

b- Suppose we are given an instance of the Minimum Spanning Tree problem on an undirected graph G. Assume that all edges costs are positive and distinct. Let T be an MST in G. Now suppose that we replace each edge cost c_e by its square root, $c_e^{1/2}$, thereby creating a new instance of the problem (G') with the same graph but different costs.
Prove or disprove: T is still an MST in G'.

The statement is true due to that replacing each edge cost c_e by its square root, $c_e^{1/2}$ does not change the order of the cost, i.e. for positive real numbers a and b, if a is greater than b, $a^{1/2}$ *is greater than* $b^{1/2}$.