CS570
Analysis of Algorithms
Summer 2010
Exam I

Name: ______________________
Student ID: __________________

____Check if DEN student

	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

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

[[✓ TRUE]/FALSE]
Suppose G is a graph with n vertices and $n^{1.5}$ edges, represented in adjacency list representation. Then depth-first search in G runs in $O(n^{1.5})$ time.

[[✓ TRUE]/FALSE]
There are multiple stable matching solutions when A prefers B, B prefers C, C prefers D and D prefers A.

[[✓ TRUE]/FALSE]
To find the minimum element in a max heap of n elements, it takes O(n) time

[[✓ TRUE]/FALSE]
A DFS tree is a spanning tree

[[✓ TRUE]/FALSE]
An array that is sorted in non-decreasing order is a binary min-heap

[[✓ TRUE]/FALSE]
Every edge in a MST tree is an edge of at least one shortest path in the graph

[TRUE/[✓ FALSE]]
Every edge in a shortest path is an edge of at least one MST

[TRUE/[✓ FALSE]]
The difference between Dijkstra's algorithm and Prim's algorithm is that Dijkstra's has a relaxation step but Prim's doesn't

[TRUE/[✓ FALSE]]
For a sparse graph, the asymptotic complexity of Krsukal's algorithm is higher than that of Dijkstra's when using a Fibonacci heap

[TRUE/[✓ FALSE]]
In the divide and conquer approach the sizes of sub-problems at each level are exactly the same

2) 15 pts

a- Arrange the following functions in increasing order of asymptotic complexity. If $f(n)=\Theta(g(n))$ then put f=g. Else, if $f(n)=O(g(n))$, put $f < g$.
$4n^2$, $\log_2(n)$, 20n, 2, $\log_3(n)$, n^n, 3^n, nlog(n), 2n, 2^{n+1}, log(n!)

(10 points) $2< \log_3(n)=\log_2(n)<2n=20n < \log(n!)=n\log(n)< 4n^2<2^{n+1}<3^n<n^n$

For 1)
a. Confused about "=" and "<" (each one missing 2 points)
b. wrong orders (each pair missing 2 points)
c. Missing item: less than the total number of 11 (each one missing 2 points)

b- Find the complexity of the following function:

```
int countPrime(int n)
{
    counter = 0;
    for(i=101; i<n; i+=2) {
        isPrime = true;
        for(j=3; j<i/2; j+=2)) {
                if(i%j == 0) {
                isPrime = false;
                break;
                }
        }
     }
return counter;
}
```

$O(n^2)$

For 2)
a. if O(n) – (Get 0 point).
b. if only write down: n^2 or $\Theta(n^2)$ – (Get 2 points)

3) 15 pts

You are given an array of n positive numbers A[1], A[2],…, A[n]. Give a divide and conquer algorithm to find indices i<j such that A[j]/A[i] is maximized. Your algorithm should run in O(n) time.

We divide the array recursively into two halves until it only contain two elements. When combine the results there are three conditions:

- Both i and j are in the first half
- Both i and j are in the second half
- i comes from the first half and j comes from the second half

Thus we need to record the max and min value of each sub array and return them to the upper level for comparison.
Note that it is crucial to describe how to obtain the max and min value of each sub array in O(1) time in the combine step. Otherwise it is not clear how they can be collected. This including 1) recursively get the max/min value through comparison and 2) return them to the upper level.

```
F(B[1..n], n)
    If n==2
        m1=min(B(1), B(2));
        m2=max(B(1), B(2));
        res=B(2)/B(1);
        return (m1,m2,res);
    else:
        (min1,max1, res1)=F(B[1..n/2], n/2);
        (min2,max2, res2)=F(B[n/2+1..n], n/2);
        m1=min(min1, min2);
        m2=max(max1, max2);
        res=min(res1, res2, max2/min1);
        return(m1, m2, res);
end
```

$T(n)=2T(n/2)+O(1)$ Thus $T(n)=O(n)$

4) 15 pts
Often there are multiple shortest paths between two nodes of a graph. Modify Dijkstra's algorithm so that it computes the shortest path and tracks the number of distinct shortest paths from a start node s to all nodes, on a graph with positive weights.

Solution: The number of shortest paths to a node v will depend on the number of path u (which is prev(v)). If this is a new shortest path, then the number of paths to u is the number of paths to v (since there is only one path from u to v). If the shortest path via u is the same as the existing shortest path to v, then the number of paths to v is incremented by the number of paths to u.

```
if dist(v) > dist(u) + l(u,v)
{
        numpaths(v) = numpaths(u)
        dist(v) = dist(u) + l(u,v)
}
if dist(v) = dist(u) + l(u,v)
        numpaths(v) += numpaths(u)
```

Proof:

By contradiction or point out the sub-shortest path is part of the whole shortest path (the property of using greedy algorithm)

Complexity: The same with Dijsktra algorithm

With a binary heap, the algorithm requires O((|E|+|V|)log|V|) time (which is dominated by O(|E|log|V|).

Grading Distribution:

1) Using Dijsktra algorithm framework (4 points)
2) Solution is correct, not necessary the same as above. (6 points.)
3) Remember there is a need to show proof or analyze complexity: (1 points)
4) Correct Proof: (2 points)
5) Correct Complexity: (2 points)

If you only keep track of the number of shortest paths between node v and another single node instead of all the other nodes, and the algorithm is right, you only can 4 points from the solution part.

5) 15 pts
A stable roommate problem with 4 students a, b, c, d is defined as follows. Each student ranks the other three in strict order of preference. A matching is defined as the separation of the students into two disjoint pairs. A matching is stable if no two separated students prefer each other to their current roommates. Does a stable matching always exist? If yes, give a proof. Otherwise give an example roommate preference set where no stable matching exists.

A stable matching need not exist. Consider the following list of preferences. Let a; b and c all have d last on their list. Say a prefers b over c, b prefers c over a and c prefers a over b. In any matching, one of a; b; c should be paired with d and the other two with each other. Now, d's roommate and the one for whom d's roommate is the _rst choice prefer to be with each other. Thus no stable matching exists in this case.

6) 20 pts
Suppose that we are given a graph G=(V,E) and one of its minimum spanning trees, say T=(V, E'). If the weight of an edge $e \in E$ is reduced, describe an algorithm to compute a minimum spanning tree of the modified graph.

Let w(.) denote the weight of an edge in the original graph and likewise $w_{'}(.)$ in the modified graph. Let w'(e) = w(e) - a.
If e belongs to E', then clearly T is a minimum spanning tree for the modified graph(Output T)
If e does not belong to E' , then adding the edge e to the tree T results in a cycle (call it C). Let e' be the heaviest edge in the cycle. If w'(e) = w'(e'), T is an MST for the modified graph (so output T).
If e is not e', then T' := (V; _E'+e-e') is an MST for the modified graph(Output T').

Proof:
The only case that is not obvious is when $T_{'}$ is output. We shall prove this by contradiction. Assume that T_0 is not an MST for the modified graph in this case. That is there exists a tree T_{opt} such that $w'(T_{opt}) < w'(T)$.
Clearly e has to be in T_{opt}. Thus $w(T_{opt}) = a + w'(T_{opt})$
Also $w'(T_{opt}) < w'(T') = w'(T) - w(e') + w'(e) = w(T) - w(e') + w(e) - a$
The above statements together imply that
$w(T_{opt}) < w(T) - w(e') + w(e) < w(T)$
This contradicts the fact the T is an MST for the original graph.

Implementation and Running Time: Once e is added to T, the maximum edge e' in the cycle C can be found as follows. Let e = (u, v) and let P be the unique path in T between u and v. The path P (and the maximum weight edge in the path) can be found using BFS in O(|V|). Then we just have to compare the maximum weight edge in P with e to find the maximum weight edge in C. The total running time is thus O(|V|).

Additional Space

Additional Space