CS570
Analysis of Algorithms
Fall 2008
Exam I

Name: _____

Student ID: _____

____Monday Section ____Wednesday Section ____Friday Section

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	20	
Problem 5	20	
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] F
Given graph *G* and a Minimum Spanning Tree *T* on *G*, you could find the (weighted) shortest path between arbitrary pair *u, v* in *V(G)* using only edges in *T*.

[TRUE/FALSE] F
$V^2 \log V = \theta (E \log E^2)$ whether the graph is dense or sparse.

[TRUE/FALSE] T
If DFS and BFS returns different trees, then the original graph is not a tree.

[TRUE/FALSE] T
An algorithm with the running time of $n * 2^{\min(n \log n, 10000)}$ runs in polynomial time.

[TRUE/FALSE] T
We may need to run Dijkstra's algorithm to compute the shortest path on a directed graph, even if the graph doesn't have a cycle.

[TRUE/FALSE] F
Given a graph that contains negative edge weights, we can use Dijkstra's algorithm to find the shortest paths between any two vertexes by first adding a constant weight to all of the edges to eliminate the negative weights.

[TRUE/FALSE] F
If *f(n)* and *g(n)* are asymptotically positive functions, then
$f(n)+g(n) = \theta(\min\{f(n),g(n)\})$.

[TRUE/FALSE] F
For a stable matching problem such that *m* ranks *w* last and *w* ranks *m* last, *m* and *w* will never be paired in a stable matching.

[TRUE/FALSE] F
Breadth first search is an example of a divide-and-conquer algorithm.

[TRUE/FALSE] T
Kruskal's algorithm for finding the MST works with positive and negative edge weights.

2) 10 pts

a) Arrange the following in the increasing order of asymptotic growth. Identify any ties.

$lg\ n^{10}$, 3^n , $lg\ n^{2n}$, $3n^2$, $lg\ n^{lg\ n}$, $10^{lg\ n}$, $n^{lg\ n}$, $n\ lg\ n$

$lg\ n^{10}$, $lg\ n^{lg\ n}$, $lg\ n^{2n}$, $n\ lg\ n$, $3n^2$, $10^{lg\ n}$, $n^{lg\ n}$, 3^n

b) Analyze the complexity of the following loops:

i- x = 0
for i=1 to n
x= x + lg n
end for

O(n)

ii- x=0
for i=1 to n
for j=1 to lg n
x = x * lg n
endfor
endfor

O(nlgn)

iii- x = 0
k = “some constant”
for i=1 to max (n, k)
x= x + lg n
end for

O(n)

iv- x=0
k = “some constant”
for i=1 to min(n, k)
for j=1 to lg n
x = x * lg n
endfor
endfor

O(lgn)

3) 10 pts

a) For each of the following recurrences, give an expression for the runtime T (n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

$T(n) = T(n/2) + 2^n$
$\theta(2^n)$

$T(n) = 16T(n/4) + n$
$\theta(n^2)$

$T(n) = 2T(n/2) + n \log n$
Master Theorem does not apply.

$T(n) = 2T(n/2) + \log n$
$\theta(n)$

$T(n) = 64T(n/8) - n^2 \log n$
Master Theorem does not apply.

b) Suppose we have a divide and conquer algorithm which at each step takes time n, and breaks the problem up into $n^{1/2}$ subproblems of size $n^{1/2}$ each. Find the runtime $f(n)$ such that $T(n) = \theta(f(n))$, given the following recurrence. $T(n) = n^{1/2}\, T(n^{1/2}) + n$

They have to show the full recursion tree.
At the root we spend cn
At the next level we have $n^{1/2}$ nodes each taking c$n^{1/2}$ so again, at this level we spend cn time
Same goes with every other level. There are lgn levels in the tree because at every we take the square root of n. So the total run time is $\theta(nlgn)$

4) 20 pts

A key to customer service is to avoid having the customer wait longer than necessary. That's your philosophy anyway when you open a coffee shop shortly after graduation. This philosophy and your CS background have made you wonder about the order in which your server should serve customers whose orders have different levels of complexity. For example, if customer one's order will take 3 minutes while customer two's will take 1 minute, then serving customer one first results in 7 minutes of waiting (3 for customer one and 4 for customer two) while serving customer two first results in only 5 (1 for customer two and 4 for customer one).

(a) Phrase this problem more formally by introducing notation for the service time and precisely define the problem's objective.

Please refer to solution to Q4

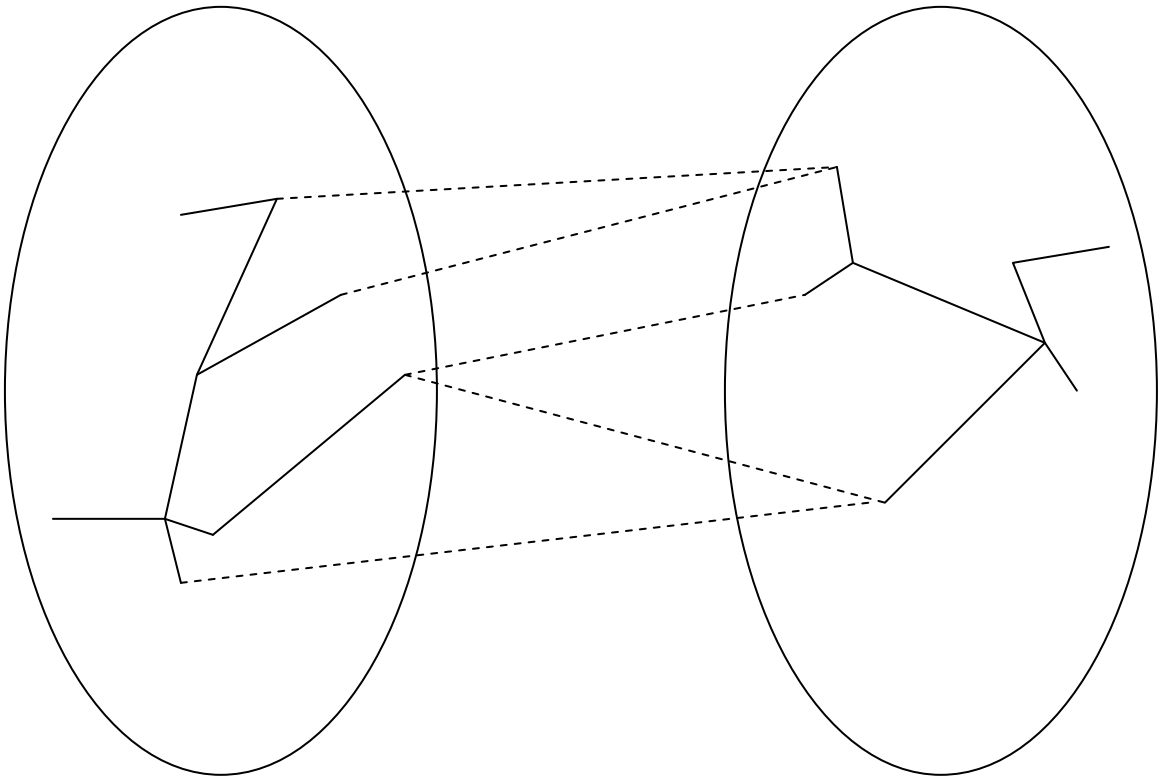
(b) Give a greedy algorithm and use an interchange argument to prove that it works for a single server as long as no new customers arrive.

(c) Show how your algorithm can fail if new customers arrive while the others are being served. (For this problem, assume that you cannot stop serving a customer once you start; coffee equipment is not designed for context switching.)

(d) Show that your algorithm does not necessarily find the best solution if the coffee shop has two servers.

5) 20 pts

Assume we have two graphs G1 = (V1, E1) and G2 = (V2, E2). Also assume that we have T1 which is a MST of G1 and T2 which is MST of G2. Now consider a new graph G = (V, E) such that V = V1 ∪ V2 and E = E1 ∪ E2 ∪ E3 where E3 is a new set of edges that all cross the cut (V1, V2). Following is an example of what G might look like.



The dashed edges are E3, the solid edges in G1 (on the left) are T1, and the solid edges in G2 (on the right) are T2.

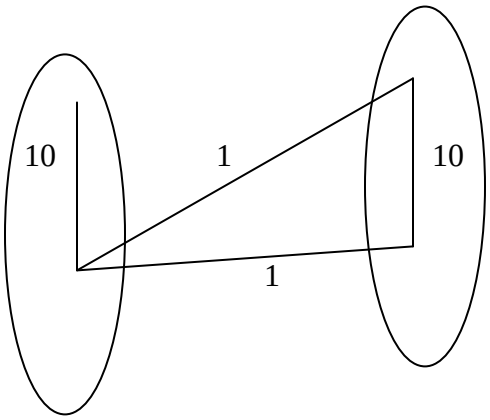
Now assume we want to fine a MST of the new graph. Consider the following algorithm:

Maybe-MST(T1, T2, E3)
e_{min} = a minimum weight edge in E3
T = T1 U T2 U { e_{min}}
return T

Prove or disprove this algorithm (space provided on next page)

Additional space

Can show a simple counter example.



6) 20 pts

For an unsorted array x of size n, give a divide and conquer algorithm that runs in *O(n)* time and computes the maximum sum *M* of two adjacent elements in an array.
$M = Max\ (x_i + x_{i+1});\ i=1\ to\ n-1$

Divide: divide problem into 2 equal size subproblems at each step
Conquer: solve the subproblems recursively until the subproblem become trivial to solve. In this case problem size of 2 is trivial. In that case the solution is the sum of the 2 numbers.
Combine: We need to merge the solutions to the two subproblems S1 and S2. At each step we need to evaluate the following 3 cases:
Case 1: Max is in S1 (subproblem to the left)
Case 2: Max is in S2 (subproblem to the right)
Cases 3: Max is on the border of S1 and S2
To achieve this. In addition to finding the Max and sending it up the recursion tree at each step we need to send the two numbers at the ends of the subarrays S1 and S2 at each step. So using the following convention:
F1 = first element of S1
L1 = last element of S1
MAX1 = maximum sum of two adjacent elements in S1
F2 = first element of S2
L2 = last element of S2
MAX2 = maximum sum of two adjacent elements in S2

Then we will combine the two subproblems as follows:

```
If (L1+F2 > MAX1 and L1+F2 > MAX2 ) then
    F = F1
    L = L2
    MAX = L1+F2
Else if (MAX1 > MAX2) then
    F = F1
    L = L2
    MAX = MAX1
Else
    F = F1
    L = L2
    MAX = MAX2
endif
```

Additional Space

Additional Space