CS570
Analysis of Algorithms
Fall 2006
Exam 1

Name: ________________________
Student ID: ____________________

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

Note: The exam is closed book closed notes.

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

[TRUE/FALSE] True (By definition it is true)
If T(n) is both **O**(f(n)) and **Ω**(f(n)), then T(n) is **Θ**(f(n)) .

[TRUE/FALSE] True
For a graph G and a node v in that graph, the DFS and BFS trees of G rooted at v always contain the same number of edges

[TRUE/FALSE] False (Counter example: Fibonacci Heap)
Complexity of the "Decrease_Key" operation is always O(lgn) for a priority queue.

[TRUE/FALSE] True (See the solution of HW4)
For a graph with distinct edge weights there is a unique MST.

[TRUE/FALSE] True (yes and store all the possible solution in a table)
Dynamic programming considers all the possible solutions.

[TRUE/FALSE] False(The graph a-b-c-d-a with three edges are 1 and the one left is 4)
Consider an undirected graph G=(V, E) and a shortest path P from s to t in G. Suppose we add one 1 to the cost of each edge in G. P will still remain as a shortest path from s to t.

[TRUE/FALSE] True
Consider an undirected graph G=(V, E) and its minimum spanning tree T. Suppose we add one 1 to the cost of each edge in G. T will still remain as an MST.

[TRUE/FALSE] False (Counter example: Shortest Path in a Graph)
Problems solved using dynamic programming cannot be solved thru greedy algorithms.

[TRUE/FALSE] False (union-Find data structure is for Kruskal's algorithm, it is not for the reverse delete. Check the textbook for more details)
The union-Find data structure can be used for an efficient implementation of the reverse delete algorithm to find an MST.

[TRUE/FALSE] False (Prim algorithm Vs Kruskal algorithm, return can be different)
While there are different algorithms to find a minimum spanning tree of undirected connected weighted graph G, all of these algorithms produce the same result for a given G.

2) 10 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 and c are positive constants. You can mark each box with Y (yes) and N (no).

A	B	**O**	**Ω**	**ϴ**
$n^3 + n^2 + n + c$	n^3	Yes	Yes	Yes
2^n	$2^{(n+k)}$	Yes	Yes	Yes
n^2	$n \,.\, 2^{\log(n)}$	No	Yes	No

3) 10 pts

a- What is the minimum and maximum numbers of elements in a heap of height h?

The minimum is 2^(n-1)
The maximum number is 2^n-1

b- What is the number of leaves in a heap of size *n*?
The smallest integer that no less than n/2.
Or
When n is even, the number of leave is n/2;
When n is odd; the number of leaver is (n+1)/2

c- Is the sequence < 23, 7, 14, 6, 13, 10, 1, 5, 17, 12 > a max-heap? If not, show how to heapify the sequence.
Answer: No. The sequence in heapify is
< 23, 7, 14, 17, 13, 10, 1, 5, 6, 12 >, < 23, 17, 14, 7, 13, 10, 1, 5, 6, 12 >

d- Where in a max-heap might the smallest element reside, assuming that all elements are distinct.
The smallest element might reside in any leaves

Grading policy: 2 points for a); 2 points for b); 3 points for c); 3 points for d)

4) 10 pts
Prove or disprove the following:
The shortest path between any two nodes in the minimum spanning tree T = (V, E') of connected weighted undirected graph G = (V,E) is a shortest path between the same two nodes in G. Assume the weights of all edges in G are unique and larger than zero.
False. Consider the graph A-B-C-D-A with weight 1,2,3,4

Grading policy:
-Any one says that the statement is true and try to prove it (of course with wrong proof) may get partial credit up to 3 marks.
- Any one says it is false without correct disproof (counter example) may get partial credit up to 6 marks.
- In Question 5, any one says it is false and give a counter example that has one or more nodes in both G1 and G2 may get partial credit up to 6 marks. Because that is a mistake, nodes in G1 can not exist in G2 and vice versa.
- Any one says it is false and give correct disprove (counter example) get 10 out of 10.

5) 10 pts
Suppose that you divided a graph G = (V,E) into two sub graphs G1 = (V1,E1) and G2 = (V2, E2). And, we can find M1 which is a MST of G1 and M2 which is MST of G2. Then, M1 U M2 U {minimum weight edge among those connecting two graph G1 and G2} always gives MST of G. Prove it or disprove it.
Solution: False.
Counter example: Consider a graph G composed of 4 nodes: A,B,C,D with edge w(A,B)=1,w(B,C)=1, w(B,D)=1, w(C,D)=2. Let V1={A,B} with edge AB, V2={C,D} with edge CD. Then the weight of M1 U M2 U is 4 while the weight of the MST of G is 3.

Grading policy:
-Any one says that the statement is true and try to prove it (of course with wrong proof) may get partial credit up to 3 marks.
- Any one says it is false without correct disproof (counter example) may get partial credit up to 6 marks.
- In Question 5, any one says it is false and give a counter example that has one or more nodes in both G1 and G2 may get partial credit up to 6 marks. Because that is a mistake, nodes in G1 can not exist in G2 and vice versa.
- Any one says it is false and give correct disprove (counter example) get 10 out of 10.

6) 20 pts
There are n workers in the factory with heights of $p_1, p_2, \ldots, p_n$, and n working-clothes with height sizes of $s_1, s_2, \ldots, s_n$. The problem is to find best matching strategy such that we minimize the following average differences.

$$\frac{1}{n}\Sigma|p_i - s_i|$$

Present an algorithm to solve this problem along with its proof of correctness.
Solution: Sort the height of workers in increasing order $p_1 \leq p_2 \leq \ldots \leq p_n$
Sort the height size of clothes in increasing order $s_1 \leq s_2 \leq \ldots \leq s_n$
Match h_i with s_i is the best matching strategy such that we minimize the following average differences.

$$\frac{1}{n}\Sigma|p_i - s_i|$$

The algorithm is correct for the problem of minimizing the average difference between the heights of workers and their clothes. The proof is by contradiction. Assume the people and clothes are numbered in increasing order by height. If the greedy algorithm is not optimal, then there is some input p1, . . . , pn, s1, . . . , sn for which it does not produce an optimal solution. Let the optimal solution be T = {(p1, s_(1)), . . . , (pn, s_(n))}, and let the output of the greedy algorithm be G = {(p1, s1), . . . , (pn, sn)}. Beginning with p1, compare T and G. Let pi be the first person who is assigned different cloth in G than in T. Let sj be the pair of cloth assigned to pi in T. Create solution T' by switching the cloth assignments of pi and pj . By the definition of the greedy algorithm, si is equal or greater than sj . The total cost of T' is given by

$$Cost(T') = Cost(T) - \frac{1}{n}(|p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j|)$$

There are six cases to be considered. For each case, one needs to show that $(|p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j|) \geq 0$.

Case 1: $p_i \leq p_j \leq s_i \leq s_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (s_i - p_j) - (s_i - p_i) - (s_j - p_j) &= 0 \end{aligned}$$

Case 2: $p_i \leq s_i \leq p_j \leq s_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (s_i - p_i) - (s_j - p_j) &= \\ 2(p_j - s_i) &\geq 0 \end{aligned}$$

Case 3: $p_i \leq s_i \leq s_j \leq p_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (s_i - p_i) - (p_j - s_j) &= \\ 2(s_j - s_i) &\geq 0 \end{aligned}$$

Case 4: $s_i \leq s_j \leq p_i \leq p_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (p_i - s_j) + (p_j - s_i) - (p_i - s_i) - (p_j - s_j) &= 0 \end{aligned}$$

Case 5: $s_i \leq p_i \leq s_j \leq p_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (p_i - s_i) - (p_j - s_j) &= \\ 2(s_j - p_i) &\geq 0 \end{aligned}$$

Case 6: $s_i \leq p_i \leq p_j \leq s_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (p_i - s_i) - (s_j - p_j) &= \\ 2(p_j - p_i) &\geq 0 \end{aligned}$$

Running time: Sorting takes O(nlog n) and hence the running time is O(nlog n)

Grading policy: Correct description of algorithm 10 pts
Based on the above correct algorithm, present correct complexity 4pts
Correct proof (all six cases) 6pts

7) 20 pts
Given an unlimited supply of coins of denominations $x_1, x_2, \ldots, x_n$, we wish to make change for a value *v*, that is, we wish to find a set of coins whose total value is *v*. This might not be possible: for example, if the denominations are 5 and 10 then we can make change for 15 but not for 12. Give an *O(nv)* algorithm to determine if it is possible to make change for *v* using coins of denominations $x_1, x_2, \ldots, x_n$.

Solution: We solve this question by dynamic programming. Denote OPT(i) as the possibility of paying value i using coins of denominations $x_1, x_2, \ldots, x_n$. Then OPT(i) is true if and only if we can paying $i\text{-}x_1$, or $i\text{-}x_2$, …or $i\text{-}x_n$ using coins of denominations $x_1, x_2, \ldots, x_n$. In other words,
$OPT(i)= OPT(i- x_1) \vee OPT(i- x_2) \vee \ldots OPT(i- x_n)$
Following the recursive relation with initial value $OPT(x_1)=\ldots =OPT(x_n)$=true we compute the value of OPT(v). If OPT(v) is true, then we can change for *v* using coins of denominations $x_1, x_2, \ldots, x_n$, otherwise we can not.

Clearly to compute OPT(v) we need an array of size v, and each time when we compute OPT(i), we do n union operations. Hence the running time is O(nv)

The following pseudo-code would help you understand the solution:
Data structures: A[v,n]: 2-dimensional array with entries T or F;
OPT[1..n]: 1-dimensional array with entries T or F.

```
 for (j=1 to v)
 OPT(j) = F;
 for (j=1 to v)
 for (i=1 to n)
{
    if (j < xi) then A[j,i] = F;
    if (j == xi) then A[j,i] = T;
    if (j > xi) then A[j,i] = OPT(xi) AND OPT(j-xi);
    if (A[j,i] == T) then OPT(j) = T;
}
return OPT(v)
```

Additional Space

Additional Space