CS325: Midterm Exam

Name and ONID number: _____

Instructions:

- This exam is closed book and closed note. No calculator or any other electronic devices.
- If you use the back of the page, please clearly mark it out so that it won't be missed in grading.
- Partial points will only be awarded if your work is shown.
- These formulae may be useful to you:

$$\log_b x = \log_k x / \log_k b$$

$$\log_b x = 1 / \log_x b$$

$$a^{\log_b x} = x^{\log_b a}$$

Question	Points	out of
1		10
2		14
3		10
4		16
BONUS		5
TOTAL		50

Questions:

1. (10 pts) In each of the following cases, indicate if $f = O(g)$, $f = \Omega(g)$, $f = \Theta(g)$. Mark Y for yes and N for no.

	$f(n)$	$g(n)$	$O(\leq)$	$\Omega(\geq)$	$\Theta(=)$
a.	$2^n \log n \quad \leq$	2.5^n	*Y*	*N*	*N*
b.	$0.2 + 0.2^2 + \cdots + 0.2^n \quad \leq$	n	*Y*	*N*	*N*
c.	$1.1^n \quad \geq$	$n^{1.1}$	*N*	*Y*	*N*
d.	$(3n)^2 \quad =$	$(n + \log n)^2$	*Y*	*Y*	*Y*
e.	$n^{\log_2 8} \quad \geq$	$n^{\log_4 8}$	*N*	*Y*	*N*
f.	$\sqrt{n} \quad \geq$	$(\log_4 n)^2$	*N*	*Y*	*N*

2. Analyze the running time of each of the following bits of pseudocode in big O notation. Note that if the run time is $\theta(n^2)$, answering $O(n^2)$ will only get partial credit. Here we assume each multiplication and each addition take constant time. Be careful to notice all the differences between the different pseudocode bits.

   (a) (3 pts)

```
x = 0
for i = 1 ... n
    for j = 1 ... n
        for k = 1 ... n
            x = x + i × j × k
```

   $\Theta(n^3)$
   2 pts for $O(n^3)$.

   (b) (3 pts)

```
x = 0
for i = 1 ... n-1
    for j = i+1 ... n
        for k = i ... j
            x = x + i × j × k
```

   $\theta(n^3)$
   2 pts for $O(n^3)$.

   (c) (4 pts)

```
x = 0
for i = 1 ... p
    for j = 1 ... q
        for k = 1 ... i × j
            x = x + i × j × k
```

   $\theta(p^2q^2)$
   3 pts for $O(p^2q^2)$.

   (d) (4 pts)

```
f(A[1,...,n]):
    if n = 0
        return 1
    else
        return A[1] × f(A[3,...,n])
```

   $\theta(n)$
   3 pts for $O(n)$.

3. (10 pts) Prove by contradiction that if the cheapest edge in a graph G is unique, then it must be part of every minimum spanning tree of G.

   Assume this is not true and there exists an MST T that does not contain this unique cheapest e. Adding e to T will create a cycle. If we remove any edge on this cycle that is not e, we will end up with a cheaper spanning tree, contradicting to the assumption that T is a MST.

   Some common mistakes:

   - *argue that Kruskal's algorithm will select e and construct an MST that contains e. But this does not prove that e is in every MST.*
   - *argue that one can get a cheaper tree by creating a cut of the graph for which e is the lightest, then identifying the edge in T that cross the same cut and replace it with e. The issue with this proof is that there is no guarantee that after replacement we have a tree. See the lecture notes on proof for cut property. The buggy proof had the exact same problem.*

4. (15 pts) In this problem, we will design an algorithm for making changes. As input, we are given $d_1, d_2, ..., d_m$, the denomination of the coins, and K, the target amount to make change for. The goal is to find the minimum number of coins needed to make the target amount K. For example, to make 80 cents of change using US coins, we have $K = 80$, $d_1 = 25$, $d_2 = 10$, $d_3 = 5$ and $d_4 = 1$, and we will need 3 quarters and 1 nickel.

   Please design a dynamic programming algorithm for solving this problem in $O(mK)$ time. You can assume that the denominations are such that we can always make the change whatever amount it is.

   Hint: please consider the following subproblem $L(k)$, which is the minimum number of coins that are required to make change k.

   (a) (6 pts) Define a recursion for computing $L(k)$.

   (b) (2 pts) What are the base cases for $L(k)$?

   (c) (5 pts) Give a pseudocode for the dynamic programming algorithm.

   (d) (3 pts) Analyze the running time of your algorithm.

(a)

$$L(k) = 1 + \min_{i=1,...,m, d_i \leq k} L(i)$$

To make change k, we just need to pick one coin from all possible coins (all i s.t. $d(i) \leq k$), each choice will lead to a smaller problem, choose among them the one uses the least number of coins.

(b) $L(0) = 0$. *To make change* 0, *we need* 0 *coin.*

(c)

```
L(0) = 0
for k = 1 ... K
    mincount= ∞
    for i = 1 ... m
        if d(i) ≤ k:  mincount=min(L(i),mincount)
    L(k)=1+mincount return L(K)
```

(d) The algorithm has two loops. K outer loops and m inner loops, leading to $\theta(Km)$ run time.

This page is left blank.

BONUS (5 pts) For the same change making problem, we will now consider a greedy algorithm that at each step chooses the coin with the largest denomination that does not exceed the target amount. For example, if we are using the US coins, we will use as many quarters as possible first, and then move on to dime, then nickel, and lastly penny. Whether this greedy algorithm can produce a solution with the minimum number of coins depends on the denominations $d_1, d_2, ..., d_m$, as well as the target K. Please give an example where the greedy algorithm does not give an optimal solution. Your example should explicitly provide the denominations and a target such that the greedy algorithm fails to return an optimal solution.

If you try to create a counter example using the US coin system, you will not succeed because the denominations of US coins ensure that greedy is optimal. Consider the following strange denominations $1, 8, 10$. *To make change for* 16*, greedy will use one* 10 *and* 6 *ones, whereas the optimal will only use two* 8.