| Name:<br><br>Covers chapter 1 – 13, 18, 22-23 | CS3112 Fall 2016 Final<br>California State University, Los Angeles<br>Instructor: Jungsoo Lim |
|---|---|

I pledge by honor that I will not discuss the contents of this exam with anyone.
Signed by _____ Date _____

Part I. (30 pts)

1. (10 pts) Let *f(n)* and *g(n)* be asymptotically positive functions. Prove or disprove each of the following statements.
   a. $f(n) + g(n) = \theta(\max(f(n), g(n)))$

   b. $f(n) = O(g(n)) \quad implies\ g(n) = \Omega(f(n))$

   c. $f(n) = \theta(f\left(\frac{n}{2}\right))$

   d. $f(n) + o(f(n)) = \theta(f(n))$

   e. $f(n) = \theta((f(n))^2)$

2. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + Dn & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

where D and C are constants, what is the order of complexity Θ of the algorithm?

3. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} 2T(n-1) + D & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

where D and C are constants, what is the order of complexity Θ of the algorithm?

4. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} T(n-1) + D\log n & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

Where D and C are constants, what is the order of complexity Θ of the algorithm?

5. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} T(n-2) + Dn^2 & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

Where D and C are constants, what is the order of complexity Θ of the algorithm?

6. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} 4T(n/3) + Dn\log n & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

Where D and C are constants, what is the order of complexity Θ of the algorithm?

7. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} 3T(n/3) + \mathrm{n}\ /\log n & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

Where D and C are constants, what is the order of complexity Θ of the algorithm?

8. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} 7T(n/2) + Dn^2\sqrt{n} & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

Where D and C are constants, what is the order of complexity Θ of the algorithm?

9. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} 9T(n/3 - 2) + Dn/2 & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

Where D and C are constants, what is the order of complexity Θ of the algorithm?

10. (5 pts) Suppose you have an algorithm that operates on a set of data with n elements. If the recurrence formula that computes the time requirement for the algorithm is given by

$$T(n) = \begin{cases} T(n-1) + n & if\ n > 1 \\ C & if\ n = 1 \end{cases}$$

Where D and C are constants, what is the order of complexity Θ of the algorithm?

Part II. (30 pts)

1. (8 pts) How many comparisons are required to sort the array A in ascending order for each of the following sorting algorithm?

   A = {10,9,8,7,6,5,4,3,2,1}

   a. Insertion sort
   b. Selection sort
   c. Quicksort (not randomized quicksort)
   d. Merge sort

2. (8 pts) How many comparisons are required to sort the array B already sorted in ascending order for each of the following sorting algorithm?

   B = {1,2,3,4,5,6,7,8,9,10}

   a. Insertion sort
   b. Selection sort
   c. Quicksort (not randomized quicksort)
   d. Merge sort

3. (4 pts) Suppose that we have numbers between 1 and 1000 in a binary search tree and want to search for the number 363. Which of the following sequences could not be the sequences of nodes examined?
   a. 924,220,911,244,898,258,362,363
   b. 925,202,911,240,912,245,363
   c. 2,399,387,219,266,382,381,278,363
   d. 935,278,347,621,299,392,358,363

4. (10 pts) Show binary tree after inserting 35, 29, 15, 23, 32,10, and 18 into an empty binary tree. Can the binary tree be a Red-Black tree? If so, put square on red node.

5. Show in-order, pre-order, post-order of above tree.

Part III

1. Ordinary multiplication of two big integers takes $n^2$ running time (n addition of n bits). However, by partitioning the integers into high n/2 and low n/2 bits as follows,

$$A = A_h * 2^{n/2} + A_l$$
$$B = B_h * 2^{n/2} + B_l$$

then, $A*B = (A_h * 2^{n/2} + A_l) * (B_h * 2^{n/2} + B_l)$
$= A_hB_h*2^n + A_hB_l*2^{n/2} + A_lB_h2^{n/2} + A_lB_l$

What is a recurrence function of partitioning approach? Solve the recurrence and find time complexity of A*B. Assume power of n takes only one operation using shift left.

$$T(n) = 4T(n/2) + Cn$$

$2^{nd}$ try,

$A*B = A_hB_h*2^n + [(A_h+ A_l)(B_h + B_l) - A_hB_h - A_lB_l]2^{n/2} + A_lB_l$

Since $(A_h+ A_l)(B_h + B_l) = A_hB_l + A_hB_h + A_lB_l + A_lB_h$

Now, what is the recurrence function of $2^{nd}$ try? Solve the recurrence and find time complexity of A*B.

$$T(n) = 3T(n/2) + cn$$
$$T(n) = N^{\log_2 3}$$

Part IV (40)

1. (10) Suppose we want to find all prime numbers less than or equal to n, we can use the following efficient algorithm instead of brute force algorithm:

**Find-All-Primes(n)**

*Let A be an array of Boolean values, indexed by integers 2 to n, initially all set to true.*

```
for p = 2 to √n
if A[p] is true
    j = p2
    while j < n
        A[j] = false
        j =j + p
Output: all p such that A[p] is true.
```

Overview of Find-All-Primes(n)

1. Create an array of Boolean indexed consecutive integers from 2 through *n*: (2, 3, 4, ..., *n*) and initialize them to be true.
2. Initially, let *p* equal 2, the smallest prime number.
3. Enumerate the multiples of *p* by counting to *n* from 2*p* in increments of *p*, and mark them in the array as false (these will be 2*p*, 3*p*, 4*p*, ...; the *p* itself should not be marked).
4. Find the first number greater than *p* in the array that is not marked as false. If there was no such number, stop. Otherwise, let *p* now equal this new number (which is the next prime), and repeat from Step 3.
5. When the algorithm terminates, the numbers remaining not marked to be false in the array are all the primes below *n*.

A. (5) What is the running time of Find-All-Primes algorithm?
(Hint: It has been proven that there are approximately $i/\log i$ primes for i.)

B. (5) The algorithm requires an array of size n. As n grows, the size of array grows as well. Thus, the algorithm is not scalable. How can we mitigate this issue?

2. (10) The greatest common divisor (GCD) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers. When GCD is equal to 1, the two numbers are called relatively prime or coprime. We can use the following efficient algorithm to find GCD between two integers.

```
GCD (m,  n)
        if m mod n == 0
                return n
        else
                return GCD (n, m mod n)
```

Overview of GCD (m, n)

1. Divide m by n and get the remainder r. If r == 0, **return n** as the GCD of m and n.
2. Replace **m by n** and replace **n by r**. Return to step 1.

A. (5) What is the recurrence of GCD algorithm?

$T(n) =$

Hint: assuming m >= n, m%n < m/2

1. If n <=m/2, then m%n < m/2 since the remainder is always less then n.
2. If n > m/2, then m%n = m – n < m/2.
3. Thus, m%n < m/2 (Each recursive call, the input size will be reduced by half at the minimum.

B. (5) Solve the above recurrence and find the running time of GCD algorithm.

3. (10 pts) Design an algorithm how to multiply two complex numbers a + bi and c + di using only three multiplications of real numbers. Your algorithm should take a, b, c, and d as input and produce the real component a*c – b*d and the imaginary component a*d + b*c separately.

MULTIPLY-COMPLEX-NUMBER(a, b, c, d)

4. (10) Suppose that you are given a sequence of n elements to sort. The input sequence consists of n/k subsequences, each containing k elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length n is to sort the k elements in each of the n/k subsequences. Design an algorithm that sorts the list in O(n log k) running time.

Extra Credit

1. (10) In an undirected graph, a path $< v_0, v_1, \ldots, v_k >$ form a **cycle** if $k > 0$, and $v_0 = v_k$. In addition, a cycle is **simple** if all edges on a path are distinct and $v_0, v_1, \ldots, v_k$ are distinct. Design an O(V) algorithm that determines whether or not a given undirected graph G = (V, E) contains a simple cycle.

2. (10) The **diameter** of a tree T = (V, E) is defined as $\max_{u, v \in V} \delta(u, v)$, that is, the largest of all shortest-path distance in the tree. Design an O(V + E) algorithm to compute the diameter of a tree.