CS 3510: Design and Analysis of Algorithms	Georgia Tech
Exam 1	
Instructor: Shahrokh Shahi	*Summer 2022*

GT Username:	**Full Name:**

Instructions:

1. Write your **name** and **GT username** on each page very clearly. Then, complete the exam.
2. This exam is closed-book, and collaboration is NOT permitted.
3. You are allowed to use **one sheet of notes**, i.e., both sides of a letter-sized paper, during the exam.
4. No calculator is required.
5. You have **80 minutes** to complete this exam.
6. It is recommended to read all the questions before starting. Please read the questions carefully. Misunderstanding the question is not a valid excuse for losing points.
7. If you find it necessary, make reasonable assumptions but make sure to state them clearly.
8. You can use the back of each sheet as scratch paper.
9. Write your solution is the space provided. In case you need more space, you can use back of the same sheet, and make a notation on the front of the sheet.
10. The exam has 50 points in total.

Good luck!

Number	Problem	Points	Grade
1	Asymptotic Notations	12	
2	Master Theorem	10	
3	Divide-and-Conquer: Tri-Merge-Sort	8	
4	Divide-and-Conquer: Binary Search	10	
5	Dynamic Programming	10	

GT Username:	**Full Name:**

1 Asymptotic Notations [12 pts]

(a) (5 pts) For each pair of functions f and g, choose one of $f \in O(g), f \in \Theta(g), f \in \Omega(g)$ that best describes their relative asymptotic growth. No justification is required.

- $f = \log(n^3)$, $g = 100\log(n)$
- $f = n^4$, $g = (n \log n)^3 + n^2$
- $f = n^{1000}$, $g = 1.5^n$
- $f = 2^n$, $g = (\frac{5}{2})^n$
- $f = (n+3)^3$, $g = 100n^3 - n$

(b) (3 pts) Give the mathematical definition of $f(n) \in \Omega(g(n))$ and provide an example.

(c) (4 pts) Assume you have functions f and g, such that $f(n) \in O(g(n))$. For the following statement, tell whether it is true or false, and give a proof (if it is true) or a counterexample (if it is false).

`if` $g(n) \in O(h(n))$`, then` $f(n) \in O(h(n))$.

GT Username:	Full Name:

2 Master Theorem [10 pts]

2.1 Solve the following recurrence relations using the Master Theorem and give the tightest bound in terms of Θ. Also, state whether the computational cost is dominated at the leaves, the root or equally distributed at all levels of the corresponding recursion tree.

(a) (3 pts) $T(n) = 9T(n/3) + \Theta(n^2)$

(b) (3 pts) $T(n) = 5T(n/4) + \Theta(\frac{1}{2}n + 5)$

2.2 (4 pts) For the following divide-and-conquer program, give the recurrence relation describing the running time and apply the Master Theorem to calculate the running time.

```
def func(n):
    if n==0: stop

    func(n/3)

    func(n/3)

    do_something in O(n)
```

GT Username:	**Full Name:**

3 Divide-and-Conquer: Tri-Merge-Sort [8 pts]

We would like to build a more advanced version of the `Merge-Sort` algorithm in which at each step the array will be divided into three sub-arrays. Answer the following questions:

(a) (4 pts) The merging step in the `Merge-Sort` algorithm (discussed in lectures) combines two sorted sub-arrays in linear time. Now, suppose we have three sorted sub-arrays A, B, and C, each of length $n/3$, and we want to merge them into a single sorted array S of length n containing all elements of these three sub-arrays. Design an algorithm that can combine these three sorted sub-arrays in linear time. Describe your algorithm in words or pseudocode.

(b) (4 pts) Consider the `Tri-Merge-Sort` algorithm in which the given array is divided into three equal length sub-arrays. Each sub-array is sorted recursively, and then, the three sorted sub-arrays are combined using your linear time algorithm from part (a). Provide the recurrence relation describing the running time of the `Tri-Merge-Sort` and apply the Master Theorem to obtain the time complexity of this algorithm.

GT Username:	**Full Name:**

4 Divide-and-Conquer: Binary Search [10 pts]

Given a sorted array $A = [a_1, a_2, \ldots, a_n]$ including all the integers in the range $\{1, 2, \ldots, n-1\}$ exactly once, expect for one of them which appears twice. Design a divide and conquer algorithm to find the only repeated element.

Example 1: repeated element = 3

$$A = [1, 2, 3, 3, 4, 5]$$

Example 2: repeated element = 1

$$A = [1, 1, 2, 3, 4, 5, 6, 7]$$

(a) (5 pts) Explain your algorithm in words, and justify its correctness.

GT Username:	**Full Name:**

(b) (3 pts) Provide the pseudocode describing your algorithm.

(c) (2 pts) Analyze the running time of your algorithm using the Master Theorem.

GT Username:	**Full Name:**

5 Dynamic Programming: Maximum Paired Sum [10 pts]

Consider an array of n integer numbers $A = [a_1, a_2, \ldots, a_n]$. Design an algorithm to find the maximum `Adjacent-Pair-Product-Sum`, which is defined as the largest value that can be obtained by multiplying adjacent elements in the array and then add them together. Each element can be paired with at most one of its immediate neighbors, but it is also allowed to be left alone.

Example 1: $A = [1, 2, 3, 1]$
Maximum `Adjacent-Pair-Product-Sum` $= 1 + (2 \times 3) + 1 = 8$

Example 2: $A = [2, 2, 1, 3, 2, 1, 2, 2, 1, 2]$
Maximum `Adjacent-Pair-Product-Sum` $= (2 \times 2) + 1 + (3 \times 2) + 1 + (2 \times 2) + 1 + 2 = 19$.
But another `Adjacent-Pair-Product-Sum` that is not optimal is $2+(2\times1)+(3\times2)+1+(2\times2)+1+2 = 18$.

(a) (1 pt) Compute the largest `Adjacent-Pair-Product-Sum` of $A = [1, 4, 3, 2, 3, 4, 2]$

(b) (4 pts) Discuss the optimal substructure of the `Adjacent-Pair-Product-Sum` problem, and give the recurrence relation including the base case(s).
You can define $OPT[i]$ as the largest Adjacent-Pair-Product-Sum of the first i elements, $a_1, \ldots, a_i$.

GT Username:	**Full Name:**

(c) (3 pts) Give the pseudocode of a bottom-up or top-down implementation of the dynamic programming algorithm using the recurrence relation from part (a).

(d) (2 pts) Analyze the time and space complexity of your algorithm.