Banner number: Name:

# Midterm Exam

## CSCI 3110: Design and Analysis of Algorithms

June 24, 2015

| Group 1 | | Group 2 | | Group 3 | | $\Sigma$ |
|---|---|---|---|---|---|---|
| Question 1.1 | | Question 2.1 | | Question 3.1 | ☐ | |
| Question 1.2 | | Question 2.2 | | Question 3.2 | ☐ | |
| $\Sigma$ | | $\Sigma$ | | $\Sigma$ | | |

**Instructions:**

- The questions are divided into three groups: Group 1 (18 marks = 36%), Group 2 (20 marks = 40%), and Group 3 (12 marks = 24%). You have to answer **all questions in Groups 1 and 2** and **exactly one question in Group 3**. In the above table, put a check mark in the **small** box beside the one question in Group 3 you want me to mark. If you select 0 or 2 questions in Group 3, I will mark neither.
- Provide your answer in the box after each question. If you absolutely need extra space, use the backs of the pages; but try to avoid it. Keey your answers short and to the point. If you need scrap paper, you can also use the backs of the pages.
- **You are not allowed to use a cheat sheet.**
- **Make sure your answers are clear and legible. If I can't decipher an answer or follow your train of thought with reasonable effort, you'll receive 0 marks for your answer.**
- If you are asked to design an algorithm and you cannot design one that achieves the desired running time, design a slower algorithm that is correct. A correct and slow algorithm earns you 50% of the marks for the algorithm. A fast and incorrect algorithm earns 0 marks.
- When designing an algorithm, you are allowed to use algorithms and data structures you learned in class as black boxes, without explaining how they work, as long as these algorithms and data structures do not directly answer the questions.
- **Read every question carefully before answering. In particular, do not waste time on an analysis if none is asked for, and do not forget to provide one if it is required.**
- **Do not forget to write your banner number and name on the top of this page.**
- **This exam has 9 pages, including this title page. Notify me immediately if your copy has fewer than 9 pages.**

## Question 1.1 (Asymptotic growth of functions) 9 marks

a. Define the set $o(f(n))$.

b. Running times of algorithms are always positive functions, that is, $T(n) > 0$ for all $n$. Use the definition of $o$-notation given above to show that, for two positive functions $f(n)$ and $g(n)$, $f(n) \in o(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

## Question 1.2 (Running time of algorithms) 9 marks

Let $A$ and $B$ be two algorithms that solve the same problem $P$. Assume $A$'s average-case running time is in $\Theta(n)$ while its worst-case running time is in $\Theta(n^2)$. $B$'s average-case running time is in $\Theta(n \lg n)$, as is its worst-case running time. The constants hidden by the $\Theta$-notation are much smaller for $A$ than for $B$ and $A$ is much easier to implement than $B$. Now consider a number of real-world scenarios where you would have to solve problem $P$. State which of the two algorithms would be the better choice in each of the following scenarios and briefly justify your answer.

a. The inputs are fairly small.

b. The inputs are big and fairly uniformly chosen from the set of all possible inputs. You want to process a large number of inputs and would like to minimize the total amount of time you spend on processing them all.

c. The inputs are big and heavily skewed towards $A$'s worst case. As in the previous case, you want to process a large number of inputs and would like to minimize the total amount of time you spend on processing them all.

d. The inputs are of moderate size, neither small nor huge. You would like to process them one at a time in real time, as part of some interactive tool for the user to explore some data collection. Thus, you care about the response time on each individual input.

## Question 2.1 (Asymptotic growth of functions) 10 marks

a. Order the following functions by increasing order of growth:

$$n \lg n \quad n^{\lg n} \quad (\lg n)^n \quad 2^{\frac{\lg n}{2}} \quad n^2$$

b. Prove that you have arranged the second and third functions in the sorted sequence in the right order; that is, if the sorted sequence is $f_1(n), f_2(n), \ldots, f_5(n)$, prove that $f_2(n) \in o(f_3(n))$.

c. Prove that $f(n) = 3n^2 + n \lg n - 10n \in \Theta(n^2)$ by providing constants $0 < c_1 < c_2$ and $n_0 \geq 0$ such that $c_1 n^2 \leq f(n) \leq c_2 n^2$ for all $n \geq n_0$.

## Question 2.2 (Correctness proofs) 10 marks

Here's yet another minimum spanning tree algorithm:

```
MST($G$)
  $T := G$
  while $T$ contains a cycle
    do Choose an arbitrary cycle $C$ in $T$
       Remove the heaviest edge in $C$ from $T$
  return $T$
```

Prove that this algorithm does indeed produce a minimimum spanning tree of $G$. You may assume that $G$ is connected and that there are no two edges in $G$ that have the same weight. Your proof needs to show that the algorithm terminates, that the result is a spanning tree of $G$, and that it has minimum weight among all spanning trees of $G$.

## Question 3.1 (Graph algorithms) 12 marks

Given a graph $G$ whose edges have positive weights, a *shortest path* between two vertices $u$ and $w$ is a path from $u$ to $w$ whose edges have minimum total weight among all paths from $u$ to $w$. The *distance* $\operatorname{dist}(u, w)$ between $u$ and $w$ is the length of such a shortest path. The *diameter* of $G$ is the maximum distance between any two vertices in $G$: $\operatorname{diam}(G) = \max_{u,v \in G} \operatorname{dist}(u, v)$. In general, computing the diameter of $G$ boils down to computing all pairwise vertex distances and reporting the largest one. If $G$ is a tree, however, things get quite a bit simpler.

Develop an algorithm that computes the diameter of a tree $T$ whose edges have positive weights. The running time of your algorithm should be in $O(n)$, where $n$ is the number of vertices of $T$. Describe your algorithm, argue that its running time is in $O(n)$, and prove that it does indeed compute the diameter of $T$.

**Extra space for Question 3.1**

## Question 3.2 (Greedy algorithms) 12 marks

In this question, you are given a set $C = \{1, 5, 10, 25\}$ of coin denominations, that is, pennies, nickels, dimes, and quarters. Your goal is to design an algorithm which, for any input value $V$ in cents, outputs the minimum number of coins with denominations in $C$ whose total value is exactly $V$. For example, to represent the value $V = 37$, we would choose one quarter, one dime, one nickel, and two pennies, a total of 5 coins. Your algorithm should take $O(|C|) = O(1)$ time. An $O(V + |C|)$-time algorithm is also acceptable.

Describe your algorithm, argue briefly that it does indeed achieve the desired running time, and prove formally that it outputs the minimum number of coins needed to represent a given value $V$, for any value $V$ it is given as input.

**Extra space for Question 3.2**