Banner number: Name:

Midterm Exam

CSCI 3110: Design and Analysis of Algorithms

June 23, 2014

Group 1		Group 2		Group 3		$\sum$
Question 1.1		Question 2.1		Question 3.1	☐	
Question 1.2		Question 2.2		Question 3.2	☐	
$\sum$		$\sum$		$\sum$		

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.
- **You are not allowed to use a cheat sheet.**
- 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 8 pages, including this title page. Notify me immediately if your copy has fewer than 8 pages.**

Question 1.1 (Asymptotic growth of functions) 9 marks

$O(f(n))$, $\Omega(f(n))$, $\Theta(f(n))$, $o(f(n))$, and $\omega(f(n))$ are sets of functions.

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

$O(f(n))$ is the set of all functions $g(n)$ that satisfy

$$g(n) \le c \cdot f(n) \quad \textit{for two constants } c > 0 \textit{ and } n_0 \ge 0 \textit{ and all } n \ge n_0.$$

b. Define the set $\Theta(f(n))$.

$\Theta(f(n))$ is the set of all functions $g(n)$ that satisfy

$$c_1 \cdot f(n) \le g(n) \le c_2 \cdot f(n) \quad \textit{for three constants } c_1 > 0,\ c_2 > 0, \textit{ and } n_0 \ge 0 \textit{ and all } n \ge n_0.$$

c. Define the set $\omega(f(n))$.

$\omega(f(n))$ is the set of all functions $g(n)$ such that, for all $c > 0$, there exists a constant $n_0 \ge 0$ such that

$$g(n) \ge c \cdot f(n) \quad \textit{for all } n \ge n_0.$$

Question 1.2 (Running time of algorithms) 9 marks

a. Define what the **worst-case running time** of an algorithm is.

The worst-case running time of an algorithm is a function $T(\cdot)$ defined as follows: Let $\mathscr{I}_n$ be the set of all inputs of size n and let $T'(I)$ be the running time of the algorithm on input I. Then $T(n) = \max\{T'(I) \mid I \in \mathscr{I}_n\}$.

b. Define what the **average-case running time** of an algorithm is.

The average-case running time of an algorithm is a function $T(\cdot)$ defined as follows: Let $\mathscr{I}_n$ be the set of all inputs of size n and let $T'(I)$ be the running time of the algorithm on input I. Then $T(n) = \frac{1}{|\mathscr{I}_n|} \sum_{I \in \mathscr{I}_n} T'(I)$.

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

a Order the following functions by increasing order of growth:

$$(\lg n)^2 \quad n^{1/\lg n} \quad \frac{n}{\lg n} \quad 4^{\lg n} \quad \sqrt{n}$$

$$n^{1/\lg n} = 2 \quad (\lg n)^2 \quad \sqrt{n} \quad \frac{n}{\lg n} \quad 4^{\lg n} = n^2$$

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 $f_1(n), f_2(n), \dots, f_5(n)$, prove that $f_2(n) \in o(f_3(n))$.

The second function is $(\lg n)^2$*, the third function is* $\sqrt{n}$*. We need to prove that* $\lim_{n\to\infty} \frac{(\lg n)^2}{\sqrt{n}} = 0$*, which hold if and only if* $\lim_{n\to\infty} \frac{\lg n}{n^{1/4}} = 0$*. To prove the latter, we apply l'Hôpital's rule:*

$$\lim_{n\to\infty} \frac{\lg n}{n^{1/4}} = \lim_{n\to\infty} \frac{1/(n \ln 2)}{n^{-3/4}/4} = \lim_{n\to\infty} \frac{4}{n^{1/4} \ln 2} = 0$$

because $\lim_{n\to\infty} \frac{4}{\ln 2} = \frac{4}{\ln 2}$ *and* $\lim_{n\to\infty} n^{1/4} = \infty$.

c Prove that $n \in o(n^2)$ without using limits.

We need to prove that, for all $c > 0$*, there exists an* $n_0 \geq 0$ *such that, for all* $n \geq n_0$*,* $n \leq cn^2$*, which is equivalent to* $\frac{1}{c} \leq n$*. So,* $n_0 = \frac{1}{c}$.

Question 2.2 (Correctness proofs) **10 marks**

Euclid's algorithm can be used to find the greatest common divisor (GCD) of two integers:

```
EUCLID(a, b)
    // You may assume that both numbers are non-negative and that at least one is strictly positive.
1   if $a < b$
2       swap $a$ and $b$
3   if $b == 0$
4       return $a$
5   return EUCLID($a - b, b$)
```

Prove that this algorithm is correct, that is, that it does indeed return the GCD of the two numbers.

We prove the claim by induction on $n = a + b$.

If $n = 1$, then we have $a = 0$ and $b = 1$ or $a = 1$ an $b = 0$. In both cases, the algorithm returns $\max(a, b) = 1$, which is the correct answer.

If $n > 1$, we distinguish three cases:

If $a = 0$, then $b > 0$ and the algorithm returns b. Since $b \mid b$, $b \mid 0$, and $c \nmid b$ for all $c > b$, this is the correct answer.

If $b = 0$, then $a > 0$ and, by the same argument as in the previous case, the algorithm correctly returns a.

Finally, if $a > 0$ and $b > 0$, observe that the algorithm swaps a and b if necessary to ensure that $a \geq b$. This guarantees that $a - b \geq 0$ and $b > 0$. Since $(a - b) + b = a < a + b$ (because $b > 0$), we can apply the inductive hypothesis to show that EUCLID$(a - b, b)$ *returns the GCD of $a - b$ and b. Thus, it remains to prove that $\gcd(a - b, b) = \gcd(a, b)$.*

Let $x = \gcd(a - b, b)$. Then $b = b'x$ and $a - b = a'x$ for some $a', b' \in \mathbb{N}$. Thus, $a = b + (a - b) = b'x + a'x = (a' + b')x$, that is, $x \mid a$. This shows that $\gcd(a, b) \geq \gcd(a - b, b)$.

Conversely, if $x = \gcd(a, b)$, then $b = b'x$ and $a = a'x$ for some $a', b' \in \mathbb{N}$, so $a - b = a'x - b'x = (a' - b')x$, that is, $x \mid (a - b)$. This shows that $\gcd(a, b) \leq \gcd(a - b, b)$.

Together, these two inequalities prove that $\gcd(a, b) = \gcd(a - b, b)$. This finishes the proof.

Question 3.1 (Graph algorithms) **12 marks**

You are given a graph G each of whose edges is either red or blue and an integer parameter $k \geq 0$. Your task is to develop an algorithm that decides whether G has a spanning tree with exactly k red edges. (The algorithm does not have to find such a spanning tree. This can be done and is not particularly hard but requires more time to figure out than you have in this exam.) The running time of your algorithm should be $O(n+m)$, where n is the number of vertices of G and m is the number of edges of G. To make your task easier, here are the three parts of the answer you have to figure out:

(a) Argue that G has a spanning tree with exactly k red edges if and only if it has a spanning tree with at most k red edges and it has a spanning tree with at least k red edges.

(b) Argue that a minimum spanning tree of a graph whose edges have weight 0 or 1 can be found in $O(n+m)$ time.

(c) Use the linear-time minimum spanning tree algorithm for 0/1 edge weights to decide whether G has two spanning trees as in (a).

(a) Clearly, if G has a spanning tree T with exactly k red edges, it also has a spanning tree with at most k red edges and a spanning tree with at least k red edges, namely T.

Now assume there exist two spanning trees T_1 and T_2, one with $k_1 \leq k$ red edges and the other with $k_2 \geq k$ red edges. Let $\{e_1, e_2, \dots, e_q\}$ be the set of edges of T_2 that do not belong to T_1. We obtain a sequence $T_1 = T'_0, T'_1, \dots, T'_q = T_2$ of spanning trees as follows: We define $T'_0 = T_1$. For $0 < i \leq q$, we add e_i to T'_{i-1}. This creates a cycle C in $T'_{i-1} \cup \{e_i\}$. Since T_2 contains no cycle, there must exist an edge f in C that does not belong to T_2. We choose such an edge f and define $T'_i := T'_{i-1} \cup \{e_i\} \setminus \{f\}$. Since C is the only cycle in $T'_{i-1} \cup \{e_i\}$ and removing f breaks this cycle, T'_i is a spanning tree of G. Now observe that $T_1 = T'_0$ has at most k red edges, $T_2 = T'_q$ has at least k red edges, and for all $0 < i \leq q$, T'_i contains exactly one edge that does not belong to T'_{i-1}, that is, the number of red edges in T'_{i-1} and T'_i can differ by at most 1. Thus, there must exist an index i such that T'_i has exactly k red edges.

(b) The cost of Prim's algorithm can be divided into $O(n+m)$ plus the cost of the $O(m)$ priority queue operations the algorithm performs. If every edge has weight 0 or 1, and thus every priority queue entry has priority 0 or 1, we can use a simple priority queue implementation consisting of two doubly-linked lists, one for priority-0 entries, one for priority-1 entries. An INSERT *operation appends the inserted element to the end of the corresponding list, which takes $O(1)$ time. A* DECREASEKEY *operation moves the element from one list to the other, which also takes $O(1)$ time. A* DELETEMIN *operation returns the first element in the priority-0 list unless this list is empty, in which case it returns the first element in the priority-1 list. This also takes $O(1)$ time. Thus, each priority queue operation takes $O(1)$ time and Prim's algorithm takes $O(n+m)$ time using this priority queue.*

(c) To test whether there exists a spanning tree with at most k red edges, we give weight 0 to all blue edges and weight 1 to all red edges. There exists such a spanning tree if and only if the MST with respect to these edge weights has weight at most k. To test whether there exists a spanning tree with at least k red edges, we give weight 1 to all blue edges and weight 0 to all red edges and ask whether the MST with respect to these edge weights has weight at most $n-k-1$.

Extra space for Question 3.1

Question 3.2 (Greedy algorithms) 12 marks

In class, we discussed the interval scheduling problem where we tried to maximize the number of non-overlapping intervals we can choose from a given set of intervals. One possible application I mentioned was that the intervals could represent the times when certain classes need to be held and we want to choose the largest subset of classes that can be held in the same classroom without holding two classes in the same classroom simultaneously. Now, simply cancelling the classes we cannot schedule in this fashion is not an option in practice. This leads us to the following problem: You are given a set of classes to be scheduled, each with a start time and an ending time. Your task is to find the minimum number, k, of classrooms such that all classes can be scheduled while ensuring that the time intervals of any two classes scheduled in the same classroom are disjoint. Your algorithm should compute a schedule of the given set of classes in k rooms and should run in $O(n \lg n + kn)$ time, where n is the number of classes provided as input and k is the number of classrooms you determine are needed. Prove that your algorithm does indeed find the minimum k such that it is possible to schedule all classes in these k classrooms.

Hint: Observe that you need at least k classrooms if there are k classes that need to be held simultaneously, that is, if there is a time t that is contained in the time intervals of k classes. Thus, your algorithm should have the property that, if it uses k classrooms to schedule all n classes, then there exists such a set of k simultaneously held classes.

Extra space for Question 3.2