Banner number: Name:

Final Exam

CSCI 3110: Design and Analysis of Algorithms

Dec 15, 2015

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		
Question 1.3		Question 2.3		Question 3.3		
$\sum$		$\sum$		$\sum$		

Instructions:

- The questions are divided into three groups: Group 1 (36%), Group 2 (36%), and Group 3 (28%). You have to answer **all questions in Groups 1 and 2** and **exactly two questions in Group 3**. In the above table, put a check mark in the **small** box beside the question in Group 3 you want me to mark. If you select none or both questions, I will randomly choose which one to mark.
- 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. Keep your answers short and to the point.
- **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 question.
- **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 12 pages, including this title page. Notify me immediately if your copy has fewer than 12 pages.**

Question 1.1 (Asymptotic growth of functions) 10 marks

(a) *Formally* define the set $o(f(n))$.

(b) *Formally* define the set $\Omega(f(n))$.

Question 1.2 (Average-case analysis and randomization) 6 marks

We studied three variants of Quick Sort in class. They differ in how they choose the pivot around which they partition the input. Worst-Case Quick Sort uses the same strategy as worst-case linear-time selection to find an approximate median as pivot. Simple Quick Sort uses the last input element as pivot. Randomized Quick Sort randomly picks one of the input elements as pivot. Compare these three algorithms according to the following three properties. Write your answers into the table. The first column asks you to list the worst-case running times of the algorithms. The second column asks you to list their expected running times. The final column asks, for each algorithm, whether it has a worst-case input, that is, an input that forces it to achieve its worst-case running time.

Algorithm	**Worst-case running time**	**Expected running time**	**Worst-case input**
Worst-Case Quick Sort			
Simple Quick Sort			
Randomized Quick Sort			

Question 1.3 (Complexity classes) **9 marks**

(a) *Formally* define the complexity class P.

(b) *Formally* define the complexity class NP.

(c) *Formally* define what an NP-hard language is.

Question 2.1 (What does it do?) 8 marks

Consider the following simple algorithm:

```
MAGIC($x, y$)
1  if $y = 0$
2      then return 0
3  if $y$ is even
4      then return MAGIC($2 \cdot x, y$ div 2)
5      else  return MAGIC($2 \cdot x, y$ div 2) $+ x$
```

The input consists of two non-negative integers x and y and the return value is another integer z. **div** denotes integer division: x **div** $y = \lfloor x/y \rfloor$. State what the algorithm computes, that is, state the relationship between x, y, and z. Prove that this is indeed what the algorithm computes.

Question 2.2 (Algorithm analysis) **8 marks**

Express the running time of the algorithm from Question 2.1 as a function of y. Prove that this is indeed the algorithm's running time by giving a recurrence for the running time and solving this recurrence using the Master Theorem, substitution or a recursion tree. Whichever method you use, show the steps you take to solve the recurrence.

Question 2.3 (Polynomial-time reductions) 9 marks

Let $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$ be two formal languages. Assume L_1 is NP-hard and there exists a polynomial-time reduction R from L_1 to L_2. Prove that this implies that L_2 is also NP-hard.

Question 3.1 (Greedy algorithms) **10 marks**

Consider a set of n jobs you want to run on a single computer. Let d_i be the *duration* of the ith job, that is, the amount of time it takes to run this job to completion. If you choose some permutation π of the jobs and you start each job immediately after the previous job in the permutation finishes, then the finish time of job $\pi(i)$ is $t_\pi(\pi(i)) = \sum_{j=1}^{i} d_{\pi(i)}$. We call $t_\pi(i)$ the *completion time* of job i. Your goal is to find the permutation π that minimizes the average completion time $\bar{t}_\pi = \frac{1}{n}\sum_{i=1}^{n} t_\pi(i)$. Develop an algorithm that solves this problem in $O(n \lg n)$ time, argue briefly that this is indeed the running time of your algorithm, and prove that the permutation π it produces does indeed minimize $\bar{t}_\pi$.

Extra space for Question 3.1

Question 3.2 (Dynamic programming) 10 marks

Recall the Subset Sum problem from class: Given a set S of n positive numbers and a target number t, the problem asks us to decide whether there exists a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = t$. We also proved in class that this problem is NP-hard, but the proof required that the numbers were exponentially large (exponential numbers can be represented in a linear number of bits). This is not just a caveat of the proof, as you will show here: Develop an algorithm that solves the Subset Sum problem in $O(nt)$ time (which is polynomial in n if t is polynomial in n). Argue briefly that your algorithm is correct and that it achieves the desired running time.

Extra space for Question 3.2

Question 3.3 (Data structures) **10 marks**

Describe a data structure that stores a set S of n numbers and supports MinPair(S) queries. A MinPair(S) query reports a pair $(x, y) \in \binom{S}{2}$ such that $|x - y| = \min_{(u,v)\in\binom{S}{2}} |u - v|$, where $\binom{S}{2} = \{(x, y) \in S \times S \mid x \neq y\}$. In other words, a MinPair(S) query reports the two elements in S with the smallest difference between them. If $|S| < 2$, MinPair(S) reports *nil*. The cost of a MinPair query should be in $O(1)$. The data structure should also support insertions of new elements into S and deletions of elements from S. These update operations should take $O(\lg n)$ time. Argue briefly that your implementations of Insert, Delete, and MinPair queries are correct and take $O(\lg n)$, $O(\lg n)$, and $O(1)$ time, respectively.

Extra space for Question 3.3