Banner number: Name:

Midterm Exam

CSCI 3110: Design and Analysis of Algorithms

June 26, 2008

Group 1		Group 2		Group 3		
Question 1.1		Question 2.1		Question 3.1	☐	Σ
Question 1.2		Question 2.2		Question 3.2	☐	
▨	▨	Question 2.3		▨	▨	
Σ		Σ		Σ		

Instructions:

- The questions are divided into three groups. 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 9 pages, including this title page. Notify me immediately if your copy has fewer than 9 pages.**

Question 1.1 (Algorithm design paradigms) 10 marks

a. Explain what characterizes a greedy algorithm.

A greedy algorithm solves an optimization problem. In trying to find a globally optimal solution, it makes quite natural, local choices.

b. List the three main steps in a divide-and-conquer algorithm.

1. *Divide the input instance I into one or more smaller instances of the same problem.*
2. *Recursively solve these smaller instances.*
3. *Combine their solutions to obtain a solution to instance I.*

Question 1.2 (Analysis of algorithms) 5 marks

a. Give the formal definition of the condition functions f and g have to satisfy so that $f(n) = o(g(n))$.

For every constant $c > 0$, one can find a constant $n_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

b. Now assume that $f(n) = o(g(n))$, that you have an algorithm A with running time $f(n)$ and an algorithm B with running time $g(n)$, and that you run both algorithms on the same, sufficiently large, input. Assume further that algorithm A runs on a slower computer than algorithm B. Which of the two algorithms will finish first? Justify your answer.

Assume that the faster computer is c times faster than the slower computer. Then we know that there exists an input size n_0 such that, for all $n \geq n_0$, algorithm A performs $2c$ times less operations than algorithm B. Hence, starting at input size n_0, algorithm A on the slower computer takes at most half as long to finish as algorithm B on the faster computer.

Question 2.1 (Asymptotic growth) 5 marks

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

$$n^2 \quad \lg n \quad n \lg n \quad \sqrt{n} \quad 2^{\lg n}$$

$$\lg n \quad \sqrt{n} \quad 2^{\lg n}\ (= n) \quad n \lg n \quad n^2$$

b. Prove that you have arranged the last two 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 $f_4(n) = o(f_5(n))$.

The last two functions are $n \lg n$ and n^2. We claim that $n \lg n = o(n^2)$ and prove this by showing that $\lim_{n\to\infty}(n \lg n / n^2) = 0$:

$$\begin{aligned}
\lim_{n\to\infty} \frac{n \lg n}{n^2} &= \lim_{n\to\infty} \frac{\lg n}{n} \\
&= \lim_{n\to\infty} \frac{\lg e(1/n)}{1} \\
&= \lim_{n\to\infty} \frac{\lg e}{n} \\
&= 0.
\end{aligned}$$

Question 2.2 (Recurrence relations) 5 marks

Solve the following recurrences using the Master theorem. To justify your answer, state which case applies and show that $n^{\log_b a}$ and $f(n)$ satisfy the conditions that need to be satisfied for this case to apply.

1. $T(n) = 3T(n/2) + \Theta(n^2)$

$a = 3$, $b = 2$. Hence, $\log_b a = \log_2 3 < 2$, that is, $n^2 = \Omega(n^{\log_2 3 + \epsilon})$. Since $3(n/2)^2 = 3n/4$, Case 3 applies, and $T(n) = \Theta(n^2)$.

2. $T(n) = 4T(n/2) + \Theta(1)$

$a = 4$, $b = 2$. Hence, $\log_b a = \log_2 4 = 2$, that is, $1 = n^0 = O(n^{\log_2 4 - \epsilon})$. Thus, we have Case 1, and $T(n) = \Theta(n^2)$.

3. $T(n) = 3T(n/3) + \Theta(n)$

$a = 3$, $b = 3$. Hence, $\log_b a = \log_3 3 = 1$, that is, $n = \Theta(n^{\log_3 3})$. Thus, we have Case 2, and $T(n) = \Theta(n \lg n)$.

Question 2.3 (Correctness proofs) 10 marks

Consider the algorithm for merging two sorted sequences L and R to produce a new sequence S containing all their elements in sorted order:

```
MERGE($L$,$R$)
 1  $S \leftarrow \emptyset$
 2  while $L \neq \emptyset$ and $R \neq \emptyset$
 3        do $x \leftarrow$ first element of $L$
 4           $y \leftarrow$ first element of $R$
 5           if $x < y$
 6              then remove $x$ from $L$
 7                   append $x$ to $S$
 8              else remove $y$ from $R$
 9                   append $y$ to $S$
10  if $L \neq \emptyset$
11     then append $L$ to $S$
12     else append $R$ to $S$
13  return $S$
```

Your task is to prove that this algorithm is correct, that is, that the returned sequence S contains the elements of $L \cup R$ in sorted order. While you are free to do this whichever way you want, one way to structure the proof is to prove the following claims by induction on the number of iterations of the while-loop executed so far and argue that they imply the correctness of the algorithm:

(i) Let L_0 and R_0 denote the contents of L and R at the beginning of the algorithm. Then at any point in time $S \cup L \cup R = L_0 \cup R_0$.

(ii) At any point in time, $x \leq y$ holds for all pairs (x, y) such that $x \in S$ and $y \in L \cup R$.

(iii) At any point in time, S is sorted.

Base case: *(before the first iteration)*

(i) We have $L = L_0$, $R = R_0$, and $S = \emptyset$. Hence, $S \cup L \cup R = L_0 \cup R_0$.

(ii) This trivially holds because S is empty.

(iii) This trivially holds because S is empty.

Inductive step: *Denote the first element in L by x an the first element in R by y, and assume wlog. that $x < y$. (The other case can be handled analogously.) In this case, we remove x from L and add it to S. Let L_b, R_b, and S_b denote the contents of L, R, and S before the current iteration, and let L_a, R_a, and S_a denote the contents of L, R, and S after the current iteration.*

Extra space for Question 2.3

(i) *We have $S_a = S_b \cup \{x\}$, $L_a = L_b \setminus \{x\}$, and $R_a = R_b$. Hence, $S_a \cup L_a \cup R_a = (S_b \cup \{x\}) \cup (L_b \setminus \{x\}) \cup R_b = S_b \cup L_b \cup R_b = L_0 \cup R_0$, where the last equality follows by the induction hypothesis.*

(ii) *Consider an element $u \in S_a$ and an element $v \in L_a \cup R_a$. If $u \neq x$, then $u \leq v$ follows by the induction hypothesis because every element in S_a except x belongs to S_b and $L_a \cup R_a \subset L_b \cup R_b$. If $u = x$ and $v \in L_a$, then $u \leq v$ because x preceded v in L_b and L_b is sorted. If $u = x$ and $v \in R_a$, then $u \leq v$ because $x < y$, y precedes v in $R_a = R_b$, and R_b is sorted.*

(iii) *By the induction hypothesis, S_b is sorted. S_a is the same as S_b, except that x has been appended to the end. Thus, it suffices to prove that $z \leq x$ for every $z \in S_b$. This, however, follows immediately from the inductive hypothesis (part (ii)) because $z \in S_b$ and $x \in L_b \cup R_b$.*

Once the while-loop terminates, we have $L = \emptyset$ or $R = \emptyset$. Wlog., assume that $R = \emptyset$. Then $S \cup L = L_0 \cup R_0$. Hence, by appending L to S, we obtain a sequence that contains the same elements as L_0 and R_0. Moreover, L is sorted as a suffix of the sorted input sequence L_0, and S is sorted by (iii). Thus, the concatenation of S and L is sorted if we can prove that every element in S is no greater than any element in L. This, however, follows immediately from (ii). Thus, the algorithm produces an output sequence that contains the elements in $L_0 \cup R_0$ in sorted order.

Question 3.1 (Greedy algorithms) **15 marks**

You have just opened a new restaurant/café in Halifax. It's called "The Greedy Place". Your restaurant has a sidewalk patio. The patio is quite long, but the sidewalk is narrow; so the tables on the patio are essentially arranged in a straight line. Since the restaurant is in Halifax, you absolutely need to cover the tables with umbrellas to allow your customers to sit outside in spite of the 24/7 rain. Your goal is to cover all tables with as few umbrellas as possible. Here are the constraints and a few simplifying assumptions:

- Every umbrella has a diameter of $8'$.
- You may assume that the tables are points; that is, a table is covered when the point where it is placed is contained in the interval covered by an umbrella.
- Assume that the table positions $p_1, p_2, \dots, p_n$ are given by increasing distance from one end of the patio. Let us call this the left-to-right order.

Develop a greedy algorithm that determines a covering of the tables with a minimum number of umbrellas. Your algorithm should take linear time. Prove that the algorithm finds a covering with the minimum number of umbrellas.

In the algorithm, we define the position of an umbrella to be its center.

```
PLACE-UMBRELLAS(p)
1  m ← −∞
2  for i ← 1 to n
3       do if p[i] > m
4             then Place an umbrella at position p[i] + 4′
5                  m ← p[i] + 8′
```

To prove that the algorithm produces a minimal set of umbrellas that covers all the tables, we have to prove that the set of umbrellas does indeed cover all the tables and that there is no smaller set with this property.

The first claim is easily proved by induction. In particular, we claim that all umbrellas placed by the end of the i-th iteration cover all tables at positions $p_1, p_2, \dots, p_i$ and the last umbrella covers all positions between p_i and m. This is most certainly true before the very first iteration. If $i > 0$, the induction hypothesis implies that, before the current iteration, tables $p_1, p_2, \dots, p_{i-1}$ are covered, and the last umbrella covers all positions between p_{i-1} and m. If the current iteration does not place any new umbrella, we have $p_{i-1} < p_i \le m$. Hence, by the induction hypothesis, table p_i is covered, and the last umbrella covers all positions between p_i and m. If the current interation place a new umbrella, then the new umbrella obviously covers p_i (because it is just at the left edge of the umbrella), and the last umbrella covers all positions between p_i and m (because m is updated to refer to the right edge of the umbrella). Since tables $p_1, p_2, \dots, p_{i-1}$ were already covered by the previously placed umbrellas, they are still covered. Hence, the current set of umbrellas covers all tables $p_1, p_2, \dots, p_i$. After the last iteration, we therefore have all positions covered.

Extra space for Question 3.1

Now let $u_1, u_2, \ldots, u_k$ be the set of positions produced by our algorithm, and let $o_1, o_2, \ldots, o_\ell$ be the positions of a minimal set of umbrellas covering all the tables, both sorted left to right. We know that $k \geq \ell$. We prove by induction on i that, for $0 \leq i \leq \ell$, $u_i \geq o_i$ and, thus, umbrellas $u_1, u_2, \ldots, u_i$ cover at least as many tables as umbrellas $o_1, o_2, \ldots, o_i$. Hence, since $o_1, o_2, \ldots, o_\ell$ cover all tables, so do umbrellas $u_1, u_2, \ldots, u_\ell$, that is, we have $k = \ell$.

For $i = 0$, the base case, neither subsequence contains an umbrella and, hence, both sequences cover no tables. Thus, our claim is true in this case.

For $i > 0$, assume that the claim is true for umbrellas $u_1, u_2, \ldots, u_{i-1}$ and $o_1, o_2, \ldots, o_{i-1}$. Let p_j be the leftmost table not covered by $u_1, u_2, \ldots, u_{i-1}$, and let p_h be the leftmost table not covered by $o_1, o_2, \ldots, o_{i-1}$. Since, by the induction hypothesis, $u_1, u_2, \ldots, u_{i-1}$ cover at least as many tables as $o_1, o_2, \ldots, o_{i-1}$, we have $h \leq j$, that is, $p_h \leq p_j$. Since umbrella o_i must cover table p_h and umbrella u_i is placed as far right as possible while still covering table p_j, we have $u_i \geq o_i$. This completes the inductive step.

Question 3.2 (Divide and conquer) **15 marks**

You are on the team of a TV station covering the Tour de France. You would like to provide your audience with different kinds of statistics about each stage of the tour. This includes detailed information about the profile of the stage. One of the pieces of information about the profile you want to provide is the highest elevations the cyclists have to master in different parts of each stage. So you are given the profile of the stage, consisting of n heights, $h_1, h_2, \ldots, h_n$, sampled at a distance of 1km along the route; and you are given m parts $P_1, P_2, \ldots, P_m$ of today's stage. Each part P_j is a pair (f_j, t_j) indicating that this part starts at kilometer f_j and ends at kilometer t_j. For each part P_j, you want to report the highest elevation between kilometer f_j and kilometer t_j. Develop a divide-and-conquer algorithm that does this in $O((n+m)\lg n)$ time. Argue that the running time of your algorithm is what you claim.

We start by observing that this problem has the following abstract formulation: Given an array $h = (h_1, h_2, \ldots, h_n)$ and a set $P = \{P_1, P_2, \ldots, P_m\}$ of query ranges of the type (f_j, t_j), report, for every query P_j, the maximal element in $h[f_j\,..\,t_j]$. This problem is known as the range-maxima problem.

```
RANGE-MAXIMA($h, P, p, r$)
     ▷ The first call is with $p = 1$ and $r = n$.
 1   if $p = r$
 2      then Report $h[p]$ as the answer to every query in $P$
 3      else $q \leftarrow \lfloor (p+2)/2 \rfloor$
 4           Partition $P$ into three sets $P_l, P_r, P^*$ such that
               • $P_l = \{P_j \in P \mid t_j \le q\}$,
               • $P_r = \{P_j \in P \mid f_j > q\}$, and
               • $P^* = \{P_j \in P \mid t_j \le q < f_j\}$.
 5           RANGE-MAXIMA($h, P_l, p, q$)
 6           RANGE-MAXIMA($h, P_r, q+1, r$)
 7           $m[q] \leftarrow h[q]$
 8           $m[q+1] \leftarrow h[q+1]$
 9           for $i \leftarrow q-1$ downto $p$
10               do $m[i] \leftarrow \max(h[i], m[i+1])$
11           for $i \leftarrow q+2$ to $r$
12               do $m[i] \leftarrow \max(h[i], m[i-1])$
13           for every query $P_j \in P^*$
14               do Report $\max(m[f_j], m[t_j])$ as the answer to query $P_j$
```

To prove that the running time of the algorithm is $O((n+m)\lg n)$, we make the following observations: For $n = 1$, that is, $p = r$, the running time of the algorithm is $O(m)$ because it iterates over all queries in P and spends constant time per query. For $n > 1$, the running time is given by the recurrence

$$T(n,m) = O(n+m) + T(n/2, m_l) + T(n/2, m_r),$$

where $m_l = |P_l|$ and $m_r = |P_r|$. Indeed, finding the middle index q takes constant time. Partitioning P into the three sets P_l, P_r, P^* takes $O(m)$ time. The recursive calls are on the two halves of h and on P_l and P_r, respectively. Computing array m takes $O(n)$ time. Finally, we spend $O(m^*) = O(m)$ time to answer all queries in P^*, where $m^* = |P^*|$.

Extra space for Question 3.2

We claim that the solution to this recurrence is $O((n+m)\lg n)$. The base case ($n \in \{2,3\}$) is trivial because we make at most five recursive calls, each of which costs at most $O(1+m) = O((n+m)\lg n)$ time.

For the inductive step ($n \geq 4$), we prove our claim using substitution; that is, we prove that $T(n,m) \leq c(n+m)\lg n$, for some $c > 0$:

$$\begin{aligned}
T(n,m) &\leq a(n+m) + T\left(\frac{n}{2}, m_l\right) + T\left(\frac{n}{2}, m_r\right) \\
&\leq a(n+m) + c\left(\frac{n}{2} + m_l\right)\lg\frac{n}{2} + c\left(\frac{n}{2} + m_r\right)\lg\frac{n}{2} \\
&\leq a(n+m) + c(n+m)\lg\frac{n}{2}, \quad \textit{because } m_l + m_r \leq m \\
&= a(n+m) + c(n+m)(\lg n - 1) \\
&\leq c(n+m)\lg n, \quad \textit{for all } c \geq a.
\end{aligned}$$

This completes the inductive step and, thus, proves our claim.