CSIE 2136: Algorithm Design and Analysis

Midterm

Time: 10:20-13:20 (180 minutes) November 14, 2014

Instructions

- This is a 3-hour close-book exam.
- There are 7 questions worth of 100 points plus 30 bonus points. The maximum of your score is 100, so you can allocate your time and select problems based on certain optimal strategies of your own.
- For any of the questions or sub-questions except bonus ones, you get 1/5 of the credit when leaving it blank and get 0 when the answer is completely wrong. You get partial credit when the answer is partially correct.
- Please write clearly and concisely; avoid giving irrelevant information.
- You have access to the appendix on the last page.
- Please print **your name, student ID, and page number on every answer page**.

Problem 1: Short Answer Questions (30 points)

Answer the following questions and briefly justify your answer. For (a)-(d), decide whether you think the statement, particularly the underlined phrase, is true or false. Briefly explain your choice to receive full credit.

(a) (3 points) True or False: To prove the correctness of a greedy algorithm, we must prove that every optimal solution contains our greedy choice.

(b) (3 points) True or False: When items have non-integer weights, it may be extremely inefficient to solve the 0/1 Knapsack Problem using dynamic programming because of lack of overlapping subproblems.

(c) (3 points) True or False: $(m_1, w_3), (m_2, w_2), (m_3, w_1)$ is one stable matching in the following preference lists:

	1st	2nd	3rd
m_1	w_3	w_2	w_1
m_2	w_2	w_1	w_3
m_3	w_3	w_2	w_1

	1st	2nd	3rd
w_1	m_1	m_2	m_3
w_2	m_1	m_2	m_3
w_3	m_2	m_1	m_3

(d) (3 points) True or False: If $f(n)$ is $O(g(n))$, then $2^{f(n)}$ is $O(2^{g(n)})$.

(e) (3 points) Given the recurrence relation $A(i,j) = F(A(i,j-1), A(i-1,j-1), A(i-1,j+1))$, provide a valid traversal order to fill the DP table or justify why no valid traversal exists.

(f) (3 points) Given the recurrence relation $A(i,j) = F(A(i-1,j-1), A(i+1,j+1))$, provide a valid traversal order to fill the DP table or justify why no valid traversal exists.

(g) (3 points) Suppose you have designed a Divide and Conquer algorithm whose running time $T(n)$ can be expressed by $T(n) = aT(n/b) + f(n)$. Briefly explain what a, b, and $f(n)$ represent in your algorithm.

(h) (3 points) Solve the following recurrence by giving a tight Θ-notation bound:

$$T(n) = 2T(n/2) + n^2 \log n$$

(i) (6 points) Describe a real-world problem that you have solved or have thought about solving using technique learned in the class. Make an educated guess about how fast your algorithm might be, and how much time it would take to solve the same problem using a brute-force approach instead.
Answer. Anything that makes sense.

Problem 2: nth Smallest in Two Databases (15 points)

You're given two databases D_1 and D_2, each of which contains n numbers. These $2n$ numbers are all distinct. For each database, you can query for the kth smallest number for any $1 \leq k \leq n$. You want to find the nth smallest number among these $2n$ numbers.

(a) (10 points) Please design a Divide-and-Conquer algorithm to find the nth smallest number among these $2n$ numbers in $O(\log n)$ queries.

Hint: Let $m = \lfloor n/2 \rfloor$ and let $v_{1,m}$ and $v_{2,m}$ be the mth smallest number in database D_1 and D_2, respectively. Let x be the nth smallest number among these $2n$ numbers. Consider where x is in three cases: $v_{1,m} > v_{2,m}$, $v_{1,m} = v_{2,m}$, and $v_{1,m} < v_{2,m}$.

(b) (5 points) Briefly justify the running time of your algorithm is indeed $O(\log n)$ queries.

Problem 3: Longest Path (20 points)

(a) (3 points) Briefly explain what optimal substructure is.

(b) (5 points) Consider a graph $G = (V, E)$, a start node $s \in V$, an end node $t \in V$, and weight w_e of each edge $e \in E$. The Longest Path Problem is to find a longest simple path going from s to t on this graph. (The length of a path is the sum of weights of all edges on the path. A simple path is a path that visits a node at most once, that is, a path without cycles.) Please construct a counterexample to show that the Longest Path Problem has no optimal substructure.

(c) Your friend made an interesting observation that the Longest Path Problem on certain types of graphs does exhibit optimal substructure. To test your friend's conjecture, you decide to use a dynamic programming algorithm to find one longest path from top-left to bottom-right on a grid-like directed graph with edges going downward and rightward only. As an example, Figure 1 Shows a 4-by-4 grid in which each edge is labeled with a weight.

Answer the following questions:

1. **(3 points)** Find one longest path in the graph in Figure 1.
   Answer. D=Down, R=Right. One longest path is D, R, D, R, R, D.

2. **(6 points)** Consider a n-by-n grid. Suppose you want to find the length of a longest path using dynamic programming. Please define your subproblems and represent the value of the optimal solution using a recurrence.

3. **(3 points)** What's special about this type of graphs? Please explain why the Longest Path Problem has no optimal substructure in general but has optimal substructure in this special type of graphs. An informal observation would be sufficient.

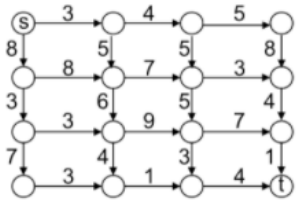

Figure 1: Find one longest path from s to t.

Problem 4: The Rise of the Planet of the Apes (15 points)

Being a brilliant ape, Caesar is learning a sign language to communicate with his trainer, Will. In this sign language, each word is represented by a sequence of gestures. This sign

language should be *prefix-free*, meaning that no word is a prefix of another word, since Caesar is smart enough to combine words into phrases. Will wants to teach Caesar the following seven words that are frequently used in their daily activities:

Word	food	sleep	play	drink	open	who	come
Frequency	0.25	0.15	0.12	0.19	0.07	0.14	0.08

(a) (5 points) Will knows that Caesar can do two gestures—pointing his index finger upward (UP) pointing his index finger downward (DOWN)—without difficulty. Please help Will to create a *prefix-free binary sign language* so as to minimize the average number of gestures per word. This sign language should contain the above seven words and each word is represented by a sequence of UP and DOWN gestures.

Answer. There are sign language satisfies this goal. For example,

Word	food	sleep	play	drink	open	who	come
	00	010	100	11	0110	101	0111

0 and 1 each represent a different gesture.

(b) (5 points) Caesar now can do three gestures—UP, DOWN, and WAVE. Please create a *prefix-free ternary sign language* so as to minimize the average number of gestures per word. The sign language should contain the above seven words and each word is represented by a sequence of UP, DOWN, and WAVE gestures.

(c) (5 points) Following (a), Will observes that the UP gesture consumes three times of energy than the DOWN gesture. Will comes up with a greedy heuristic in the hope to minimize the average energy consumption per word. His greedy heuristic is as follows. Given n words and their frequencies, Will first draws a binary prefix tree that minimizes the average number of gestures per word in its corresponding binary sign language. He then re-labels each edge such that for every node v and its left node v_ℓ and right node v_r, edge (v, v_ℓ) is re-labeled with UP and edge (v, v_r) is re-labeled with DOWN if the sum of the frequencies of the leaf nodes in v's left subtree is lower than it of its right subtree. The labels are switched otherwise. Please show that this greedy algorithm is incorrect. Your don't need to use the same input (i.e., the 7 words and their frequency distributions) in your counterexample.

Problem 5: Counting Inversions (10 points + 10 bonus points)

(a) Counting inversions revisited (10 points) Given a sequence of unique numbers $B = b_1, b_2, ..., b_n$, an inversion is a pair of numbers b_i and b_j in this sequence such that $i < j$ and $b_i > b_j$. Let $I(B)$ be the set of inversions in B. For example, if $B = \langle 1, 3, 5, 2\rangle$, $I(B) = \{(3, 2), (5, 2)\}$, and the number of inversions $|I(B)|$ is 2.

Given an unsorted sequence B with n numbers, please design an efficient algorithm to calculate the number of inversions $|I(B)|$ in $O(n \log n)$ time. Please briefly explain why your algorithm indeed runs in $O(n \log n)$.

(b) Bonus: Counting significant inversions (10 points) Given a sequence of unique numbers $B = b_1, b_2, ..., b_n$, a *significant inversion* is a pair of numbers b_i and b_j in this sequence such that $i < j$ and $b_i > 2b_j$. Let $SI(B)$ be the set of significant inversions in B. For example, if $B = \langle 1, 3, 5, 2 \rangle$, $SI(B) = \{(5, 2)\}$, and the number of significant inversions $|SI(B)|$ is 1.

Given an unsorted sequence B with n numbers, please describe how to modify your algorithm in part (a) for calculating the number of significant inversions $|SI(B)|$ in $O(n \log n)$ time. Please briefly explain why your algorithm indeed runs in $O(n \log n)$.

Problem 6: Finding closest pair of points (10 points + 10 bonus points)

(a) (10 points) There are n points in a two-dimensional space, and point i's x-y coordinate is (x_i, y_i). However, in this bizarre world, the distance between points i and j is defined as $d(i, j) = \min\{|x_i - x_j|, |y_i - y_j|\}$, which is not Euclidean distance.

Suppose you are given arrays X_n and Y_n containing these n points sorted by their x-coordinates and y-coordinates, respectively. Please design a greedy algorithm to find a closest pair of points in $O(n)$ time, and explain why your algorithm is correct. You can prove it by showing your algorithm has the greedy choice property and optimal substructure. Alternatively, you can also use other proof techniques as long as you can show that your solution is indeed optimal.

Hint: first consider a 1D case where the distance is defined as $d(i, j) = |x_i - x_j|$ and then extend your solution to 2D.

(b) Bonus (10 points) There are n points in a two-dimensional space, and point i's x-y coordinate is (x_i, y_i). However, in this bizarre world, the distance between points i and j is defined as $d(i, j) = \max\{|x_i - x_j|, |y_i - y_j|\}$, which is not Euclidean distance.

Please design an algorithm to find a closest pair of points in $O(n \log n)$ time, and explain why your algorithm is correct.

Problem 7: Fair Division of Christmas Gifts (10 bonus points)

Christmas is approaching. You're helping Santa Clauses to distribute gifts to children.

For ease of delivery, you are asked to divide n gifts into two groups such that the weight difference of these two groups is minimized. The weight of each gift is a positive integer. Please design an algorithm to find an optimal division minimizing the value difference. The algorithm should find the minimal weight difference as well as the groupings in $O(nS)$ time, where S is the total weight of these n gifts. Briefly justify the correctness of your algorithm.

Hint: This problem can be converted into making one set as close to $S/2$ as possible.

Appendix

Master Theorem. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.