CSIE 2136: Algorithm Design and Analysis (Fall 2016)

Midterm

Time: 14:20-17:20 (180 minutes) November 10, 2016

Instructions

- This is a 3-hour close-book exam.
- There are 7 questions worth of 120 points in total. The maximum of your score is 100, so you can allocate your time and select problems based on certain optimal strategies of your own.
- 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 (20 points)

Answer the following questions and briefly justify your answer.

(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: Any dynamic programming algorithm that solves n subproblems must run in $\Omega(n)$ time.

(c) (4 points) Please explain why the 0/1 knapsack problem may not be solved in $O(nW)$ time (where n is the number of items and W is an integer representing the knapsack's capacity) when items have non-integer weights.

(d) (5 points) Recall that in the interval scheduling problem, we want to find the maximum number of compatible jobs given n job requests and their start times $s[i]$ and finish times $f[i]$ for all $1 \leq i \leq n$. Please provide a counterexample showing that a shortest-interval-first greedy strategy does not always lead to an optimal solution.

(e) (5 points) You're working on a dynamic programming problem that has a recurrence relation $A(i,j) = F(A(\lfloor i/2 \rfloor, j), A(i, \lfloor j/2 \rfloor))$, where F is a known function that can be evaluated in constant time, and $A(i,j) = 0$ when $i = 0$ or $j = 0$. To compute $A(m,n)$ for some m and n, you can use either a top-down or a bottom-up method. Which one is more efficient in solving this problem?

Problem 2: Asymptotic Notions (10 points)

(a) (5 points) Three divide-and-conquer algorithms are proposed to solve the same problem. Suppose they are all correct, what are their running time (in big-O notation) and which one is more efficient? Please justify your answer.

- Algorithm A divides the problem of size n into three subproblems of size $n/2$ and combines the solutions in linear time.
- Algorithm B divides the problem of size n into two subproblems of size $n-1$ and combines the solutions in constant time.
- Algorithm C divides the problem of size n into nine subproblems of size $n/3$ and combines the solutions in $O(n^2)$ time.

(b) (5 points) Does $f(n) = O(g(n))$ imply $2^{f(n)} = O(2^{g(n)})$? Please justify your answer.

Problem 3: Integer Multiplication (10 points)

Given two n-bit integers x and y, please write pseudocode to solve integer multiplication by dividing the original n-bit problem (xy) into three $\frac{n}{2}$-bit subproblems, recursively. Your algorithm should run in $O(n^{\lg 3})$ time.

Problem 4: Christmas Again (20 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.

(a) (10 points) 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.

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

(b) (5 points) Please write down the recurrence relation you derived in (a), and use it to complete a DP table for solving the following problem instance.

Gift i	1	2	3	4
Weight w_i	2	2	4	3

(c) (5 points) You are now asked to divide n gifts into 2^k groups such that the sum of the pairwise weight differences (i.e., $\sum_{\forall i>j} |s_i - s_j|$, where s_i is the weight of group i) is minimized. k is a positive integer. You come up with a divide-and-conquer algorithm that recursively divides gifts into two groups while minimizing the weight difference of these two groups. The algorithm stops when the recursion depth reaches k.

Please provide a counterexample to show that this approach is not always optimal.

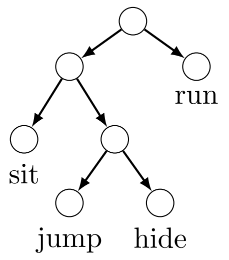
Problem 5: Zoologist (20 points)

As a zoologist, you're interested in studying how to communicate with Pokemons. Since most Pokemons can make sound, the length of sound can be used to encode 1-bit of information. We will use a long sound to represent 1 and a short sound to represent 0.

(a) (5 points) You want to teach Pokemons seven commands that are frequently used during training. Please draw an optimal prefix tree for encoding these commands:

Word	eat	sleep	jump	drink	run	sit	hide
Frequency	0.25	0.15	0.12	0.19	0.07	0.14	0.08

(b) (9 points) You found a secret training document written by another trainer, which shows that the trainer encodes four commands using a prefix tree as follows:



Suppose the prefix tree is constructed using Huffman encoding. For each of the following statements, please explain whether it is true or false:

- The command *run* must have a frequency higher than 1/3.
- The command *jump* must have a frequency less than 1/8.
- The frequency of command *run* is no less than it of command *sit*.

(c) (6 points) You observed that making a long sound consumes r times more energy than making a short one. Please revise the prefix code you came up with in (a) to minimize the expected energy consumption per word for the 7-word example when $r = 100$. Please briefly justify your answer.

Problem 6: Zombie Apocalypse (20 points)

Due to a zombie virus outbreak, some cities have been occupied by zombies and are no longer safe. You and your survivor team need to travel through several cities to get to a far away shelter.

There are n cities forming a line topology. You are at city 1 now and the shelter is at city n. The location of city i is $L[i]$, and $L[i] < L[j]$ $\forall 1 \leq i < j \leq n$. $z[i] = 1$ indicates city i has been occupied by zombies; otherwise, $z[i] = 0$ indicates the city is still safe to stop at night.

(a) (10 points) If you plan to move at most 100km a day, and you need to rest at a safe city at night, please design a greedy algorithm to pick the cities for resting at night so that you can arrive at the shelter as soon as possible. Your algorithm should runs in $O(n)$ time. Please show that your algorithm has the greedy choice property.

(b) (10 points) In a zombie apocalypse, everyone is stressed. You observed that when you travel for x km in a day, the level of stress within your team is increased by $(100 - x)^2$. Please design an algorithm to pick the cities for resting at night so that you can arrive at the shelter while minimizing the total stress. Your algorithm should runs in $O(n^2)$ time. Please justify the correctness of your answer.

Problem 7: Majority Elements (20 points)

Given an array A of size n, we call an element a *majority element* of A if the element appears more than $\lceil n/2 \rceil$ times in A. Please design an algorithm to find the *majority element* if it exists; your algorithm should also correctly return `NONE` if the majority element does not exist. Note that the elements of the array are not necessarily from some ordered domain like the integers, so you can only check whether two elements are the same or not. In other words, you cannot use `sort` to solve this problem.

(a) (10 points) Design an algorithm that finds the majority element in $O(n \log n)$ time.
Hint: Split the array A into two arrays A_1 and A_2 of half size. Can you find the majority element of A from that of A_1 and A_2?

(b) (10 points) Design an algorithm that finds the majority element in $O(n)$ time.
Hint: Pair up the elements in A to get $n/2$ pairs.

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 $T(n) = \Theta(f(n))$.