CSIE 2136: Algorithm Design and Analysis (Fall 2015)

Midterm

Time: 14:20-17:20 (180 minutes) November 10, 2015

Instructions

- This is a 3-hour close-book exam.
- There are 7 questions worth of 130 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.
- Please print **your name, student ID, and page number on every answer page**.

Problem 1: Short Answer Questions (25 points)

Answer the following questions and briefly justify your answer.

(a) (3 points) True or False: The 0/1 knapsack problem can be solved in polynomial time with respect to the size of the input, when the knapsack as well as every object has an integer weight.

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

(c) (3 points) True or False: When proving correctness using an exchange argument, we want to transform an optimal solution into the greedy solution without hurting its quality.

(d) (3 points) True or False: Any dynamic programming algorithm that solves n subproblems must run in $\Omega(n)$ time.

(e) (3 points) Given the recurrence relation $A(i,j) = F(A(i-2,j+1), A(i+1,j-2))$, provide a valid traversal order to fill the DP table or justify why no valid traversal exists.

(f) (3 points) Your friend implemented Merge Sort based on the following pseudocode, in which A is an array and $MergeSort(A,p,r)$ should sort the elements in A between indices p and r. However, the running time of $MergeSort(A,1,n)$ seems to be $\Theta(n^2)$ in worst case, higher than $O(n \log n)$. Please help your friend debug the code by explaining which one of these four steps should be modified.

```
MergeSort(A, p, r)
        if p = r //step 1
                return
        randomly select q between p and r //step 2
        MergeSort(A, p, q) //step 3
        MergeSort(A, q + 1, r) //step 4
        Combine the two sorted subarrays in linear time
```

(g) (7 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.

Problem 2: Asymptotic Notions (15 points)

(a) (7 points) Rank the following functions by an increasing order of the growth rate: $n^{3/4}$, n^n, $\log 2n$, $n \log n$, $e^{\ln n}$, 3^n, $2^{\sqrt{\log n}}$.

(b) (8 points) Solve the following recurrence by giving a tight Θ-notation bound:

$$T(n) = 2T(\sqrt{n}) + (\log n)^2 \log \log n$$

Problem 3: Maximum Subarray of a Circular Infinite Sequence (30 points)

Recall that a maximum subarray of A is a contiguous subarray $a_s, \cdots, a_t$ of A such that $\sum_{s \le i \le t} a_i$ is maximized over all s and t, $0 \le s \le t$.

Given a *circular infinite sequence* $A = \langle a_0, a_1, a_2, \cdots \rangle$ in which $a_i = a_j$ if $i = j \mod n$, please answer the following questions.

(a) (5 points) Suppose $\sum_{0 \le i < n} a_i > 0$. What is the length of the maximum subarray of A? Briefly explain your answer.

(b) (5 points) Suppose $\sum_{0 \le i < n} a_i < 0$. Please briefly explain why the length of any maximum subarray is at most n.

(c) (10 points) Please design an algorithm to find a maximum subarray of the circular infinite sequence A in $O(n \log n)$ time. Please justify the correctness and running time of your algorithm.

Hint: we learned how to find a maximum subarray of a finite sequence in $O(n \log n)$ using divide and conquer. Can you modify it to solve this similar problem?

(d) (10 points) Can you reduce the running time of your algorithm to $O(n)$? If you have designed a $O(n)$ algorithm in (c), then you will get full credits for (d) automatically.

Hint: use dynamic programming. How to represent a maximum subarray ending at a certain position?

Problem 4: SimCity (20 points)

You are a mayor of a virtual city. The city has n districts located along a line $[0, n-1]$, and the ith district is located at i with a population $p_i > 0$. As a mayor, you need to decide where to build power plants such that every citizen can enjoy electricity while minimizing their complaints about pollution.

Each power plant is built at an integer location and can provide electricity to citizens living within an integer range R. For example, if $R = 2$, a power plant in District 3 can provide electricity to citizens living in Districts 1 to 5.

The compliant level is quantified by the total population living in the districts with power plants. For example, if $R = 2, n = 6$, then we can build power plants in Districts 2 and 5 for a full coverage, and the cost of pollution is $p_2 + p_5$.

(a) (10 points) Given R, n and p_i for all $0 \leq i < n$, please design a $O(nR)$ dynamic-programming algorithm to find an optimal allocation of power plants such that every citizen can enjoy electricity while minimizing their compliant level. Please prove that the problem has optimal substructure and justify the running time of your algorithm.

(b) (10 points) Like in every city, your citizens love to live nearby schools, subway stations, etc., which result in an uneven geographical distribution of population. Particularly, in your city, $p_i \geq p_{i+1}$ for all $0 \leq i \leq n-2$. Knowing this skewed distribution, please design a greedy algorithm to find an optimal allocation of power plants. Your greedy algorithm should be asymptotically faster than the one in (a). Please explain the running time of your algorithm and prove the greedy-choice property.

Problem 5: Pirates of the Caribbean (20 points)

You are a happy pirate sailing on the Caribbean Sea. Your captain, Jack, recently obtains a new ship and asks you to design an inter-ship communication system that can encode and broadcast his commands. Luckily you have taken an algorithm class!

(a) (5 points) You have been on the ship long enough to know that Jack uses the following seven commands most frequently:

Command	accelerate	stop	left	right	ack	attention	fire
Occurrences per day	10	8	6	6	4	2	1

Please create an optimal code using Huffman coding so as to minimize the expected length of binary codewords per command. Please also calculate the expected length of codewords.

(b) (10 points) The initial testing was successful, so now the captain asks you to extend your system to cover n commands he uses daily. Since there is no computer on the ship, computing an optimal code by hand can be pretty tiring for a large n. Another pirate suggests the following divide-and-conquer approach:
Given n commands and their occurrences f_i $(0 \leq i < n)$,

- *Divide* Sort the n commands by their occurrences from left to right, and then divide them into two groups, G_l and G_r, such that the sum of occurrences in the left group is as close to the sum in the right as possible (ties may be broken arbitrarily). For example, if the sorted frequencies are $30, 25, 20, 20, 5$, then $G_l = \{30, 25\}$ and $G_r = \{20, 20, 5\}$.
- *Conquer* Assign G_l and G_r to two different pirates and ask them to compute an optimal code for their respective group. If one thinks the group is still too large to solve, he or she can further divide the work by repeating the first step.
- *Combine* To combine, prepend 0 to the codewords in G_l, and prepend 1 to the codewords in G_r.

Do you agree that this divide-and-conquer approach can also produce an optimal code? If yes, please justify the correctness of this algorithm. If not, please provide a counterexample with its codewords.

Problem 6: HH-code Again (10 points)

HH-code is a kind of string with `0` and `1`. For a given HH-code H, any subsequence of H is called a hidden HH-code of H. For example, if there is an HH-code $H =$ `1011`, then `1, 0, 10, 01, 11, 101, 111, 011, 1011` are all the *different* hidden HH-codes of H.

When analysing time complexity, assume every basic arithmetic operation (i.e. addition, subtraction, multiplication, and division) takes $O(1)$ time.

If the length of HH-code is N, and we want to know the number of different hidden HH-code with a specific length K. Please design an algorithm to calculate the number in $O(KN)$ time.

Problem 7: Counting Significant Inversions (10 points)

We have learned how to count inversions in class. Now let's consider a slightly different problem.

Given a sequence of unique numbers $B = b_1, b_2, \ldots, b_n$, 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 a sequence B with n unique numbers, please design an algorithm to calculate the number of significant inversions $|SI(B)|$ in $O(n \log n)$ time. Please briefly explain why your algorithm indeed runs in $O(n \log n)$.