CMPSCI 311: Introduction to Algorithms **Spring 2017**

Homework 4

Last Compiled: April 6, 2017

Your Name: _____

Collaborators and sources: _____

You make work in groups, but you must write solutions yourself. List collaborators on your submission.

If you are asked to design an algorithm, please provide: (a) the pseudocode or a precise English description of the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submission instructions. This assignment is due by 8:00pm on Friday, Feb 10 in Gradescope (as a pdf file). Please review the course policy about Gradescope submissions on the course website.

1. **(20 points) Master Theorem.** In class we solved recurrences of the form $T(n) \leq aT(n/2) + O(n)$ for different values of a, and observed different outcomes for the cases $a < 2$, $a = 2$, and $a > 2$. These cases corresponded to whether the total work at each level of the recursion tree was decreasing, staying constant, or increasing.

   We can state an even more general result known as the *Master Theorem*. Suppose $T(n) = aT(n/b) + O(n^d)$. Then:

   $$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

   The three cases again correspond to whether or not the total work is decreasing, staying constant, or increasing. You may use this theorem in this problem and in the remainder of the homework.

   (a) (5 points) Describe in words an algorithm with the stated recurrence. How many recursive calls? On what size problem? How much work outside of the recursion?

   (b) (5 points) Prove the second case of the Master Theorem, that $T(n) = O(n^d \log n)$ if $d = \log_b a$.

   (c) (10 points) Suppose you are choosing between the following three algorithms:

   - Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
   - Algorithm B solves problems of size n by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.
   - Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

   What are the running times of each of these algorithms (in big-O notation), and which would you choose?

2. **(20 points) Decimal to Binary.** Recall that in class we designed a divide-and-conquer algorithm to multiply two n-digit *decimal* numbers in time $O(n^{\log_2(3)})$. In this problem, we'll assume the existence of a subroutine `fastmultiply`(x,y) that takes two *n-bit* binary numbers x and y, and returns the binary representation of their product xy in time $O(n^{\log_2(3)})$. We'll then use this subroutine to convert numbers from decimal to binary.

   Decimal numbers will be represented as arrays of digits, so you can index them to access the ith digit, but are unable to directly multiply them without first converting to binary.

(a) We'll first design an algorithm `pwr2bin` to convert a decimal number that is a power of 10 to binary. Specifically, `pwr2bin`(n) computes the binary representation for 10^n. Assume that n is a power of 2.

def `pwr2bin`(n):
If $n = 1$: return 1010 (decimal 10 in binary)
Else:
z = /* FILL ME IN */
Return `fastmultiply`(z,z).

What is the appropriate value for z? What is the running time of the algorithm?

(b) Next, we will design an algorithm `dec2bin`(x), which accepts as input an n-digit decimal number x, and returns the binary representation of x. Assume that n is a power of 2.

def `dec2bin`(x):
If `length`(x) = 1: return `binary`(x)
Else:
Split x into x_L and x_R, where x_L contains the first $n/2$ digits, and x_R contains the last $n/2$ digits.
Return /* FILL ME IN */.

The subroutine `binary`(x) performs a lookup into a table containing the binary value of all decimal numbers $0, \ldots, 9$. What are we supposed to return? What is the running time of this algorithm?

3. **(20 points) Fast Matrix Multiplication.** Given two $n \times n$ matrices X, Y of integers, their product is another $n \times n$ matrix Z with,

$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

Naively, computing Z seems to take $O(n^3)$ time since there are n^2 entries, and computing the value for a single entry involves n addition operations. In this problem, we'll develop a faster matrix multiplication algorithm. For simplicity, assume that n is a power of 2.

First, define eight $n/2 \times n/2$ matrices A, B, C, D, E, F, G, H so that,

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

From basic linear algebra, we know that:

$$Z = XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

(a) The equation above suggests a divide and conquer algorithm for matrix multiplication. Describe the algorithm in words, write down the running time as a recurrence relation, and solve the recurrence.

(b) The above algorithm uses eight recursive calls, but a more clever decomposition due to Strassen can achieve the same result using only seven recursive calls of similar form. What is the recurrence for such an algorithm and what would the running time be?

4. **(20 points) K&T Chapter 5, Exercise 1**. You are working as a programmer for UMass administration, and they ask you to determine the median GPA for all students. However, student GPAs are stored in two different databases, one for in-state students and one for out-of-state students. Assume there are n students of each type, so there are $2n$ students total. You'd like to determine the median of this set of $2n$ values, which we will define here to be the nth smallest value.

However, security is very tight, so the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen

database will return the kth smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most $O(logn)$ queries.

5. **(15 points) K&T Chapter 5, Exercise 3**. Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we'll say that two bank cards are equivalent if they correspond to the same account.

   It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester that takes two bank cards and, after performing some computations, determines whether they are equivalent.

   Their question is the following: among the collection of n cards, is there a set of more than $n/2$ of them that are all equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester. Show how to decide the answer to their question with only $O(n \log n)$ invocations of the equivalence tester.

6. **(0 points).** How long did it take you to complete this assignment?