Fall 16: CSci 5421—Advanced Algorithms and Data Structures

Out 9/7 **Homework 1** Due 9/21

INSTRUCTIONS:

- Please be sure to read and understand the sections on assignment submission and academic integrity in the syllabus (under "Class Policies"). They apply to this and all future coursework.
- Please do all problems. *However, due to TA resource limitations, we will grade only a subset of the assigned problems (same subset for everyone).*
- Any exercise/problem number specified refers to the *3rd edition* of the text.
- Remember that we will be looking at a very large number of assignments, so it is important that you do your best to facilitate the grading process (and keep us in good humor :-)). Towards this end, please keep the following in mind:

  Write *legibly* or, better still, *type* your answers (using LaTeX if possible). *Staple* your answer sheets and include your *name* and *student ID#*.

  Communicate your ideas clearly and precisely, especially where proofs and/or algorithms are involved. Don't leave it to us to guess what you mean; we can give points only for what we see on the answer sheet, not for what you may have *intended* for us to see. At the same time, don't be verbose; keep your answers concise and to-the-point. The points you earn will be determined by the content and clarity of your answers, not their length (see below).
- As a guideline, answers will be graded according to the following *general* criteria:
  - Answer demonstrates complete understanding of the problem. It has no errors and is crystal clear. It belongs in the textbook. Answers in this category will receive 100%.
  - Answer demonstrates substantial understanding of the problem. It has only one or two minor errors and is explained clearly. Answers in this category will receive between 66% and 100%.
  - Answer demonstrates only partial understanding of the problem. It has several minor errors and/or a major error. Answers in this category will receive between 33% and 66%.
  - Answer demonstrates little or no understanding of the problem. It has several major errors. Answers in this category will receive between 0% and 33%, depending on the severity of the errors.

  Where algorithms are requested, you must generally give three things: (a) a brief description of the main ideas behind your algorithms, including data structures used, from which the correctness of your approach should be evident; (b) pseudocode (along the lines of the text); and (c) an analysis of the running time. (Bear in mind that these are general guidelines—some problems may have more specific requirements.)

  To give you a better idea for what is meant, the class web page has a sample solution that you may use as a model.
- Above all, have fun! Algorithm design is similar to solving a recreational puzzle—and should be just as enjoyable.

Over $\Longrightarrow$

1. (12 points) In each case below, use the Master Theorem (MT) to provide tight asypmtotic bounds for the indicated recurrence. *Show your work.* (For each case, assume that $T(1) = \Theta(1)$.)

(a) $T(n) = 2T(n/4) + n \log n$, if $n > 1$.
(b) $T(n) = 2T(n^{1/4}) + 1$, if $n > 1$. (Consider doing a change of variables, as in Sec. 4.3.)
(c) $T(n) = aT(n/b) + dn$, if $n > 1$, where $a \geq 1$ and $b > 1$ are integer constants and $d > 0$ is a real constant. Recall that this is the recurrence solved by the Little Master Theorem seen in class. There we derived the solution by iterating the recurrence from first principles; here you should derive it directly from the MT. Specifically, show that

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b \\ \Theta(n \log_2 n) & \text{if } a = b \\ \Theta(n) & \text{if } a < b \end{cases}$$

2. (9 points) Let y and z be n-bit integers, where n is a power of 3. Consider the following divide-and-conquer algorithm to compute the product yz.

Break y into three $\frac{n}{3}$-bit pieces, a, b, and c; thus $y = a2^{2n/3} + b2^{n/3} + c$, where the powers of 2 denote appropriate bit-shifting. Similarly, break z into pieces d, e, and f. Now compute yz recursively as:

$$yz = ad2^{4n/3} + (ae + bd)2^n + (af + be + cd)2^{2n/3} + (bf + ce)2^{n/3} + cf.$$

You may ignore the issue of "carries" throughout.

(a) What is the running time of this algorithm as a function of n? Justify your answer by writing down and analyzing the recurrence.

With a view towards improving the running time in part (a), consider the following approach, where we first compute certain intermediate products $(r_1, \ldots, r_6)$ and use these along with additions and bit-shifts to compute yz.

$$r_1 = ?,\ r_2 = (a + b)(d + e),\ r_3 = be,\ r_4 = ?,\ r_5 = cf,\ r_6 = ?,\ \text{ and } yz = ?.$$

(b) Fill in the missing information above for r_1, r_4, and r_6 and show how to compute yz.

(c) What is the running time of this algorithm and how does it compare with the one we designed in class? Justify your answer by writing down and analyzing the recurrence.

3. (9 points) Prof. M.A. Tricks claims to have discovered a new algorithm for multiplying two $n \times n$ matrices. His algorithm is similar to Strassen's, except that it partitions each matrix into submatrices of size $n/8 \times n/8$ and computes the desired product using k recursive matrix multiplications and a constant number of matrix additions. (Assume, for simplicity, that n is a power of 8.) What is the *largest* value of k for which the professor's algorithm beats the running time of Strassen's algorithm? Justify your answer carefully using the Master Theorem.

Over $\Longrightarrow$

4. (9 points) In class we discussed an $O(n \log n)$-time divide-and-conquer algorithm to find the closest pair among n points in the plane, under the Euclidean distance metric. Suppose that we wish to now solve the problem under the Manhattan (or city-block) distance metric, defined as follows: For points $p = (x_p, y_p)$ and $q = (x_q, y_q)$, the *Manhattan distance* $d(p, q) = |x_p - x_q| + |y_p - y_q|$. Discuss how to modify the above algorithm to solve this problem in $O(n \log n)$ time.

It is sufficient to *carefully* describe and justify the changes needed in words; pseudocode is not required. Pay particular attention to justifying how many points need to be checked during each step of the "conquer" phase. Also analyze the running time briefly.

5. (12 points) Recall the (worst-case) linear-time divide-and-conquer algorithm for finding the kth smallest of n reals, where we used groups of size 5. Suppose that we use the same algorithm but with groups of size g for some positive integer constant g. *Derive* the recurrence relation for this algorithm as a function of n and g. (Consider separately the case where g is even and where it is odd.)

Based on your recurrences, determine the smallest integer g for which the algorithm runs in linear time and justify your answer.

You may ignore floors and ceilings in your derivation and you do not have to write the algorithm itself.

6. (15 points) Let S be a set containing all but one of the integers in the range $[0, n]$, where $n + 1$ is a power of 2. Assume that S is implemented as a linked list. (The list is not necessarily in sorted order.) One way to find the missing integer efficiently, in $\Theta(n)$ time, is to insert the elements of S into an array $I[0 : n]$ and then scan I. However, this assumes that one can access any integer, i, in S in constant time with a single operation. Suppose that one adopts a different (more restrictive) computational model, where we are able to access only the jth bit in the binary representation of integer i in constant time. Give a divide-and-conquer algorithm to compute the missing integer in $\Theta(n)$ time even under this model.

Your answer should include (a) a brief description of the main ideas from which the correctness of your algorithm should be evident, (b) pseudocode, and (c) an analysis of the running time.

7. (15 points) Let S be a set of $n \geq 2$ distinct real numbers. Let $\max(S)$ and $\min(S)$ be the largest and smallest numbers in S, respectively. Define $avg\_gap(S)$ to be $\frac{1}{n-1}(\max(S) - \min(S))$; this is the average distance on the real line between consecutive elements of S when S is in sorted order. Give a divide-and-conquer algorithm to compute distinct elements x and y in S such that $|x - y| \leq avg\_gap(S)$. The running time should be $O(n)$. (Note, that the given set S is *not* necessarily in sorted order. You cannot afford to sort it since this would take $O(n \log n)$ time.)

Your answer should include (a) a brief description of the main ideas from which the correctness of your algorithm should be evident, (b) pseudocode, and (c) an analysis of the running time.