
Homework 3 – Due Friday, February 9, 2007

Reminders Collaboration is permitted, but you must write the solutions by yourself without assistance, and be ready to explain them orally to a member of the course staff if asked. You must also identify your collaborators. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

When you describe an algorithm, remember that clarity is the main goal. Pseudocode is preferred, but an English-language description is also acceptable if it is unambiguous.

Exercises These should not be handed in, but the material they cover may appear on exams.

- Recurrences: CLRS Ex. 4.3-1, 4.3-2, 4.3-3, 4.3-4, Problem 4-4
- Quicksort: CLRS Ex. 7.1-1, 7.1-2, 7.1-3, 7.1-4, 7.2-1, 7.2-2, 7.2-3, 7.2-4
- Divide and Conquer: CLRS Problem 2-4.

Problems to be handed in

1. (**Recurrences**) Give asymptotic upper and lower bounds for $T(n)$ for each of the following recurrences. Assume $T(n)$ is constant for $n \leq 10$. Make your bounds as tight as possible. If you are applying the Master Theorem, state which of the three cases you are using. If the Master Theorem is not applicable: (i) Draw a recursion tree. Indicate the height of the tree as well as the approximate sum of the nodes at each level. (ii) Use the substitution method to prove asymptotic upper and lower bounds for $T(n)$.

(a) $T(n) = 3T(n/3) + n^3$

(b) $T(n) = T(6n/7) + n$

(c) $T(n) = 27T(n/3) + n^3$

(d) $T(n) = 7T(n/3) + n^2$

(e) $T(n) = 7T(n/2) + n^2$

(f) $T(n) = 3T(n/9) + \sqrt{n}$

(g) $T(n) = 3T(n/4) + n \log n$

(h) $T(n) = 3T(n/3) + n/\log n$

(i) $T(n) = 2T(n/5) + \log^2 n$

(j) $T(n) = 64T(n/2) + 2^n$

(k) $T(n) = T(n/4) + T(3n/4) + 2n$

(l) (For fun, do not hand in.) $T(n) = T(\sqrt{n}) + \log \log n$. (Hint: substitute $m = \log n$.)

2. You are given an array of n integers ranging from 1 to 999. Design an $O(n)$ algorithm to rearrange elements of the array *in place* so that all 1-digit numbers precede all 2-digit numbers, and all 2-digit numbers precede all 3-digit numbers. Analyze the worst-case running time of your algorithm.
3. (**Divide-and-conquer**) You are consulting for a small investment company. They give you a price of Google's shares for the last n days. Let $p(i)$ represent the price for day i . During this time period, the company wanted to buy 1,000 shares on some day and sell all these shares on some later day. The company wants to know when they should have bought and when they should have sold the shares in order to maximize the profit. If there was no way to make money during the n days, you should report this instead.

For example, suppose $n = 3, p(1) = 9, p(2) = 1, p(3) = 5$. Then you should return "buy on day 2, sell on day 3".

Your goal is to design a divide-and-conquer algorithm for this problem that runs in time $O(n)$.

- (a) In one sentence, describe a simple algorithm for the problem that takes time $O(n^2)$.
 - (b) Use the same "divide" strategy as in Merge Sort. Consider two cases: (i) there is an optimal solution in which the investors are holding the stock at the end of day $n/2$; (ii) there no such optimal solution. For each case, express the optimal solution in terms of something you can compute on the two sublists.
 - (c) State your algorithm and analyze its running time.
4. (**Groundhog Day Problem**) You are given n groundhogs of different sizes and n corresponding holes (burrows). You are allowed to try to stick a groundhog into a hole, from which you can determine whether the groundhog is larger than the hole, smaller than the hole, or matches the hole exactly. However, there is no way to compare two groundhogs together (they squirm too much) or two holes together (you don't have a ruler with you). The problem is to match each groundhog to its hole. Design a randomized divide-and-conquer algorithm for this problem with expected efficiency $\Theta(n \log n)$.
 - (a) State the algorithm.
 - (b) State the recurrence for the running time and solve it.

- 5*. (**Lightest subarray**)(**Extra credit: Please hand in on a separate sheet of paper**) Given an array of real numbers $A[1..n]$, we define the **weight** of a subarray $A[i..j]$ for $1 \leq i \leq j \leq n$ to be the sum of the elements contained in this subarray, $weight(i, j) = \sum_{k=i}^j A[k]$. The **absolute weight** is the absolute value of the weight, $|weight(i, j)|$. Notice that the numbers in the array are allowed to be negative, but the absolute weight is always non-negative. Your task is to design (and analyze) a divide-and-conquer algorithm that finds a subarray with the minimum absolute weight. Please submit your solution only if your algorithm runs in time $o(n^2)$.