

General Instructions

1. All numbered problems are from the text *Introduction to Algorithms*, Third edition.
 2. Note the difference in the text between *exercises*, which appear at the end of each section, and *problems*, which appear at the end of each chapter.
 3. Please write your name on each page of your solution and staple the pages together. If you worked with or discussed with someone else, including the TAs or the professor, please remember to indicate on each problem with whom you worked.
 4. To indicate the problem number, you can either use the number in the list below, or the number from the text. So you could write "Problem 1" or "Exercise 3.1-1". Better yet, write both!
 5. For general advice on homework, homework lateness policy, and honor code, refer to the *Course Syllabus* (on the CS 25 web page), which we discussed at the beginning of the term.
-

1. (6 points) Exercise 3.1-1
2. (10 points) Let $f(n) > 2$ and $g(n) > 2$ be any two functions. Prove that if $\lg f(n) = o(\lg g(n))$, then $f(n) = o(g(n))$. Show that the converse does not always hold.
3. (20 points) For each pair of functions $(f(n), g(n))$ below, you must state and prove the strongest possible relationship between them, in terms of say whether $f(n) = o(g(n))$, or $f(n) = O(g(n))$, or $f(n) = \omega(g(n))$, or $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$, or none of these.

Note that we are looking for the *strongest* relationship; thus, for example, if $f(n)$ happens to be $o(g(n))$, no points will be awarded if you state and prove $f(n) = O(g(n))$.

You must supply a clear justification or proof for each pair.

- (a) 2^n and $n!$
 - (b) $6^{\lg n}$ and $5^{\lg n}$
 - (c) \sqrt{n} and $n^{\sin n}$
 - (d) $(n!)^n$ and 2^{2^n}
4. (6 points) Prove that if $f(n) = \Omega(n^{\lg_b a + \epsilon})$ for some constant $\epsilon > 0$ and $f(n) = n^k$ for some constant k , then the regularity condition of the Master method holds: i.e., there are positive constants n_0 and $c < 1$ such that $af(n/b) \leq cf(n)$ for all $n \geq n_0$.
 5. (42 points) Problem 4-1, all parts.

You may use Master method or the Substitution method, but not the Recursion tree method. (Recursion tree may be used only to guess the solution, but once you guess the solution, you are required to prove it using the Substitution method.)

To ensure that you get practice with all methods, I require you not to use either of Master method or the Substitution method on more than four parts.

When proving by substitution, please follow the format I used in the class.

6. (16 points) I want you to solve Problem 4-5 in the book. The book sets up the problem well, but states the questions in Parts (a), (b), and (c) a bit ambiguously. So I am rephrasing Parts (a), (b), and (c) below.

Let $\text{DETECT}(n, m)$ denote the problem of identifying any one correct chip from a pile of n chips of which at most m are faulty. (For example, if $m = 3$, the actual number of faulty chips can be any number between 0 and 3.)

- For all $n > 1$ and $m > n/2$, prove that there is no algorithm for the $\text{DETECT}(n, m)$ problem.
- Show that $n/2$ pairwise tests are sufficient to reduce the problem to one of nearly half the size, assuming that less than half the chips are faulty. More precisely, if $m < n/2$, prove that $\text{DETECT}(n, m)$ can be reduced to $\text{DETECT}(n/2, m/2)$ using only $n/2$ pairwise tests.
- Assuming $m < n/2$, describe an algorithm that solves the $\text{DETECT}(n, m)$ problem using $O(n)$ pairwise tests. Prove that the algorithm performs $O(n)$ pairwise tests.

Extra Credit Problem

Please read what I said about extra credit on the *Course Syllabus* page. Spend time on extra credit problems only if they interest you.

In class, we looked at a Divide-and-Conquer strategy for multiplying two $n \times n$ matrices A and B to obtain their product C . If

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

and

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

then Strassen made a very clever observation that their product

$$C = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

can be computed as follows. (In the above a, b, c etc. are $n/2 \times n/2$ matrices.)

$$\begin{aligned}
P_1 &= a(f - h) \\
P_2 &= (a + b)h \\
P_3 &= (c + d)e \\
P_4 &= d(g - e) \\
P_5 &= (a + d)(e + h) \\
P_6 &= (b - d)(g + h) \\
P_7 &= (a - c)(e + f) \\
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}$$

- a. Verify that Strassen's method is correct.
- b. Set up the recurrence for the time complexity of this method and then solve the recurrence. Do you see any improvement over the naive Divide-and-conquer method?