

## Homework 2, CS420, Fall 2011

Due 9/6 at recitation

Modified 8/29 7:00

1. Get a better bound than the book does for the algorithm of Section 1 by having the method return four values instead of one, so that `Find-Max-Crossing-Subarray` can be carried out in  $O(1)$  time instead of  $\Theta(n)$ . This can be accomplished in a language like Java by having the method return a list of length 4.

The two extra values are the maximum sum  $r$  that ends at the righthand side of the call's interval and the maximum sum  $l$  that begins at the lefthand side of this interval, the sum  $s$  of all elements in the interval, and the best sum  $b$  achievable in the interval without these constraints, which is the answer to the problem.

This is an example of a paradoxical phenomenon that comes up often when you're designing a recursive algorithm: it is often easier to solve a more general problem. You would think that solving a more general problem can't be any easier, since it solves the given problem as a special case. However, there is a type of "good karma" you get from doing this in a recursive algorithm: your recursive calls will, in turn, solve the more general problem for you. The extra power of your recursive calls sometimes makes it easier for you to solve the more general problem in the main call.

- Tell how to find  $l$ ,  $r$ ,  $s$ , and  $b$  from the values  $l_1$ ,  $r_1$ ,  $s_1$  and  $b_1$  returned by the first recursive call and  $l_2$ ,  $r_2$ ,  $s_2$  and  $b_2$  returned by the second, in  $O(1)$  time. Your expressions should involve sums and the `max` operator.

**Example:**  $r = \max\{r_2, r_1 + s_2\}$ .

- Give a new recurrence, and get a big-O bound for it in closed form.

2. 4.2-1, page 82.
3. 4.2-5, page 82.
4. 4.2-3, page 82. Don't modify Strassen's. Instead, *reduce* the general problem to the one where  $n$  is a power of 2.

A *reduction* from problem A to problem B essentially means providing a way to use a program for problem B to find solutions to instances of problem A.

A reduction can save you from writing a lot of new code. Reductions are important for other reasons, also. For example, they are the way that a problem is shown to be *NP-hard*, which means that there is little hope of ever finding an algorithm that isn't hopelessly inefficient in the worst case. We will talk about this later in the course.

- (a) Briefly describe an algorithm for turning your given square matrices into ones where the dimensions are a power of two, and an algorithm for extracting the answer to your problem from what Strassen's returns. *Hint: consider padding your given matrices with 0's.*
- (b) Let  $M$  and  $M'$  be the given  $n \times n$  matrices to be multiplied. Let  $n'$  be the power of two that you have chosen. Show that your reduction is correct by showing

why the entry in position  $(i, j)$  of your final answer is the dot product of row  $M_{i,*}$  of your first given matrix and column  $M_{*,j}$  of your second given matrix. Do this by expressing the dot product of row  $M_{i,*}$  and column  $M'_{*,j}$  as a sum of dot products involving similar expressions for rows and columns of the submatrices  $A, B, C, D, E, F, G, H$  of size  $n'/2$ . You will have four expressions. Which one applies depends on the relationships of  $i$  and  $j$  to  $n'/2$ . This will make the result obvious from the associative property of addition, and the fact that the particular way you have changed  $M$  and  $M'$  to boost their dimensions to a power of two can't change the values of these dot products.

- (c) Strassen will take  $\Theta(n'^{\log_2 7})$  time. Explain why your conversion algorithms don't take longer than this.
- (d) Show that  $\Theta(n'^{\log_2 7})$  is  $\Theta(n^{\log_2 7})$ .

**5. The multiplication algorithm you learned in grade school is not optimal. (Postponed from Homework 1)**

The number 3276 is just the value of the polynomial  $f(n) = 3x^3 + 2x^2 + 7x + 6$ , evaluated at 10. If we want to find  $3276 * 2421$ , we can multiply the polynomials  $f(x) = 3x^3 + 2x^2 + 7x + 6$  and  $g(x) = 2x^3 + 4x^2 + 2x + 1$  to get  $h(x) = 6x^6 + 10x^5 + 28x^4 + 47x^3 + 40x^2 + 19x + 6$ , and evaluate  $h(10)$  to get our answer.

Given two integers with a total of  $n$  digits that we want to multiply, we can pad the two integers with leading zeros so that our polynomials  $f(x)$  and  $g(x)$  have a number  $n'$  of terms that is the smallest power of two that is as big as the number of digits in the larger integer. We can then find  $h()$ , then evaluate it at 10.

- Convince yourself that the grade-school method is  $\Theta(n^2)$ .
- Convince yourself that  $n'$  is less than  $4n$ .
- We can then multiply the two polynomials to get  $h()$ . Convince yourself that the number of terms in  $h()$  is  $O(n')$ ?
- Convince yourself that it takes  $O(n')$  time, hence  $O(n)$  time, to evaluate  $h(10)$ .

Work parts *a* and *b* of 30-1 on page 920. I've shown part *c*, above, assuming you can show *a* and *b*.

For part *b*, rather than explaining the algorithm, show me that you know how it works by illustrating it on the example of  $f(n) = 3x^3 + 2x^2 + 7x + 6$  and  $g(n) = 2x^3 + 4x^2 + 2x + 1$ . They are implying that you should divide  $f(n)$  into  $a(n) = 3x + 2$  and  $b(n) = 7x + 6$ , and  $g(n)$  into  $c(n) = 2x + 4$  and  $d(n) = 2x + 1$ , and apply your strategy from part *a*, where *a*, *b*, *c*, and *d* are themselves these smaller polynomials. To illustrate it, do only the following:

- (a) Tell what polynomials result from your three recursive calls.
- (b) Show how to add, subtract, and shift these to get the coefficients of  $h(n)$ .

Then give a recurrence for the general algorithm and use the master theorem to get a better bound than  $\Theta(n^2)$ .

6. Problem 33.4-6, page 1044. Pass in indices  $i, j$  and array  $X$  of points. The preconditions are that  $i$  and  $j$  are valid indices into  $X$  and  $i \leq j$ . The postconditions are that the returned object records a closest pair of points, and that  $X[i..j]$  has been sorted by  $y$  coordinate.

Your procedure will be a modification of `mergesort`.

For your answer, list what operations are performed outside of the two recursive calls, in the order in which they occur. One of them is `merge`. You explain why the other operations take  $O(n)$  time. This will ensure that the recurrence is still the one given in the section,  $T(n) = 2T(n/2) + O(n)$ .