

Homework 1

January 28, 2003

Due Date: February 4

For problems 3 and 4, describe your divide-and-conquer algorithms in enough detail that each can be clearly understood and that you can explain and justify the recurrence relation you give. Use the pseudo-code given in class for the divide-and-conquer closest pair algorithm as a model.

- (20 points) In each part below you should assume that A_1 has worst-case time complexity of $T_1(n)$ and A_2 has worst-case time complexity of $T_2(n)$. For each the 5 pairs of functions you are to answer the following four questions: Is $T_1(n) = O(T_2(n))$? Is $T_1(n) = \Omega(T_2(n))$? Is $T_1(n) = \Theta(T_2(n))$? And finally, assuming that your goal is to pick the fastest algorithm for large n would you pick A_1 or A_2 ? Briefly justify your answers.

(a) $T_1(n) = n \log_2 n + 10n + 5$, and $T_2(n) = n\sqrt{n}$

(b) $T_1(n) = n^{\log_2 n}$, and $T_2(n) = \log_{10} n \cdot n^{\log_{10} n}$

(c) $T_1(n) = n\sqrt{n} + 20n \log n$, and $T_2(n) = n \ln n$

(d) $T_1(n) = 2^{\log_2 n} + 10 \log_2 n$, and $T_2(n) = n$

(e) $T_1(n) = 100 \cdot n\sqrt{n} \cdot \log_2 n$, and $T_2(n) = n^{1.6}$

- (20 points) Suppose that you are to design an algorithm to solve a problem given to you. You have thought of five divide-and-conquer algorithms that have time complexities as given by the following recurrences (for each, $T(1) = \Theta(1)$). You are to give tight asymptotic bounds for $T(n)$ in each of the following recurrences (i.e. it's sufficient to use Θ notation so use the master method whenever you can). Then answer the final question. Show your work!

Algorithm A: $T(n) = 2T(2n/3) + 5n - 4$

Algorithm B: $T(n) = 8T(n/2) + 2n + 100 \log_{10} n - 5$

Algorithm C: $T(n) = 2T(n/4) + \sqrt{n} + 10 \log_2 n$

Algorithm D: $T(n) = 3T(n/2) + \Theta(n^2)$

Algorithm E: $T(n) = T(n - 1) + n$

Final Question: Which algorithm is fastest?

- (30 points) Here we explore the task of designing a fast algorithm to multiply two n -digit numbers x and y where n is large and thus must be represented using a data structure such as a list or array where each element holds a single digit. In one instruction you can only multiply, add or subtract two 1-digit numbers. So you must design an algorithm to multiply two n digit numbers using only basic arithmetic operations on 1 digit numbers (which can be performed as a table look-up).

Useful Observations: (1) Multiplying a number by a power of 10 can be implemented as a single shift operation, (2) Two n digit numbers can be added in $O(n)$ time (by breaking it down into $n + 1$ additions of two one digit numbers).

(a) Briefly describe the algorithm you would use to multiply two n -digit numbers by hand and then argue that this method has time complexity $\Omega(n^2)$? (That is you are showing a lower bound on the time complexity of this naive algorithm. If you don't believe your algorithm has time complexity $\Omega(n^2)$ let me know. But so far nobody has shown me an algorithm that can be done easily by hand that does not have time complexity $\Omega(n^2)$.) *Hint: How many single digit multiplies are made?*

(b) Let's now design a divide-and-conquer algorithm for this problem. I'll get you started. Divide x into x_ℓ and x_r where x_ℓ is the most significant $\lfloor n/2 \rfloor$ digits of x and x_r is the least significant $\lceil n/2 \rceil$ digits of x . So for example if $x = 783621$ then $x_\ell = 783$ and $x_r = 621$. In the same manner split y into y_ℓ and y_r .

Complete the design of this algorithm, clearly describing it. You need not go into details about how to create x_ℓ , x_r , y_ℓ , and y_r but should discuss the worst-case time to do this splitting. Give and solve the recurrence relation describing the time complexity. *Hints: Note that $x = x_\ell \cdot 10^{\lfloor n/2 \rfloor} + x_r$ and $y = y_\ell \cdot 10^{\lfloor n/2 \rfloor} + y_r$. How could you use this fact if you recursively use your algorithm to multiply two $\lfloor n/2 \rfloor$ digit numbers?*

(c) Try to reduce the time complexity of your algorithm in part (b) by taking advantage of the fact that $x_\ell y_r + x_r y_\ell = (x_\ell - x_r)(y_r - y_\ell) + x_\ell y_\ell + x_r y_r$.

In particular, how many subproblems of size roughly $n/2$ do you now need to solve in order to then combine to solve the original problem?

Describe the algorithm you obtain by applying this fact. Then give and solve the recurrence relation describing the time complexity.

(d) Of the three algorithms you have developed which is the fastest?

4. (30 points) Here we consider the *Maximum Contiguous Subsequence Sum Problem*: Given an array A of n (possibly negative) integers, find the maximum value of $\sum_{k=i}^j A[k]$. If all the integers are negative, then the maximum subsequence is the empty sequence, so the subsequence sum is zero. Here are some examples to be sure the problem is clear. The subsequence giving the optimal solution is underlined, but your algorithm should just return the answer itself.

- $A = \langle \underline{1, 11} - 8, 7, -4, 3 \rangle$ has an answer of 12.
- $A = \langle 2, -1, 1, 3, -4, -6, \underline{7, -2, 3}, -5 \rangle$ has an answer of 8.
- $A = \langle -1, \underline{4, -3, 5, -2}, \underline{-1, 2, 6}, -2, 1 \rangle$ has an answer of 11.

- (a) What is the worst-case asymptotic time complexity (using big-Theta notation) for the following algorithm to solve the Maximum Contiguous Subsequence Sum Problem? Be sure to show your work!

```
int maxSubsequenceSum(int[] A){
    n = A.length;
    int maxSum = 0;
    for (int i=0; i < n; i++){
        int thisSum = 0;
        for (int j=i; j < n; j++){
            thisSum = thisSum + A[j];
            if (thisSum > maxSum)
                maxSum = thisSum;
        }
    }
    return maxSum;
}
```

- (b) Now you will design and analyze a divide-and-conquer procedure `MaxSubSum` for this problem. To avoid copying the array to split it into two halves, in each recursive call you will find the maximum subsequence sum for the subarray $A[\textit{left}], \dots, A[\textit{right}]$. The initial call would be `MaxSubSum(A, 0, A.length-1)` and the signature is:

```
int MaxSubSum(int[] A, int left, int right){
```

Let $\textit{mid} = (\textit{int}) (\textit{left} + \textit{right}) / 2$. Suppose that you recursively found the solution for the subarray $A[\textit{left}], \dots, A[\textit{mid}]$ and recursively found the solution for the subarray $A[\textit{mid}+1], \dots, A[\textit{right}]$. Here's a useful hint for helping with **combining**. For the left half you could compute a running sum from $A[\textit{mid}]$ back to $A[\textit{left}]$. And for the right half you could compute a running sum from $A[\textit{mid}+1]$ up to $A[\textit{right}]$. Here is an illustration (using the third example above) of the proposal where $\textit{left} = 0$ and $\textit{right} = 9$.

	index		0	1	2	3	4		5	6	7	8	9
	A		-1	4	-3	5	-2		-1	2	6	-2	1
	sum from middle		3	4	0	3	-2		-1	1	7	5	6

For part (b) of this problem you are to submit the following:

- i. Pseudo-code for your divide-and-conquer algorithm.
- ii. An argument as to why your algorithm is correct.
- iii. An analysis of the asymptotic time complexity of your algorithm.
- iv. A brief discussion of how this algorithm compares to the one presented in part (a).

Challenge Problems:

Only work on these after you have completed the required problems. Note that you can obtain 100% without doing either of these.

1. (2 points) Give a tight asymptotic for the following recurrence. $T(n) = 2T(n/2) + n/(\log_2 n)$ where $T(1) = 1$ for $n \geq 1$ any power of 2.

Hint: Use a recurrence tree. As shown in the text, using integrals to bound the harmonic series $H_m = \sum_{i=1}^m \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{m}$ yields:

$$\ln(m+1) \leq H_m \leq (\ln m) + 1$$

2. (3 points) Design an $O(n)$ algorithm for the Maximum Contiguous Subsequence Sum Problem. Be sure to prove your algorithm is correct and analyze the asymptotic time complexity. *Don't try to design a divide-and-conquer algorithm.*