

Homework 1

February 3, 2004

Due Date: February 10

1. (20 points) For a given problem suppose you have two algorithms: A_1 and A_2 with worst-case time complexity of $T_1(n)$ and $T_2(n)$, respectively. For each part, 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 $T_1(n) = \Theta(T_2(n))$? If your goal is to pick the fastest algorithm for large n would you pick A_1 or A_2 ? You might find it helpful to first express both $T_1(n) = \Theta(f(n))$ and $T_2(n) = \Theta(g(n))$ for the simplest possible $f(n)$ and $g(n)$. For example for part (a), $T_1(n) = \Theta(\sqrt{n})$ and $T_2(n) = \Theta(\log n)$. Briefly justify your answers.

(a) $T_1(n) = 5\sqrt{n} + \log_2 n + 3$, and $T_2(n) = 100 \log_2 n + 25$

(b) $T_1(n) = 5\sqrt{n} + \ln n$, and $T_2(n) = \sqrt{n} \cdot \log_2 n$

(c) $T_1(n) = 5 \log_{10} n - 20$, and $T_2(n) = \frac{1}{2} \log_2 n$

(d) $T_1(n) = 10n^{3/4}$, and $T_2(n) = \sqrt{n} \cdot \log_2 n$

(e) $T_1(n) = 10^{(2 \log_{10} n)} - 2n$, and $T_2(n) = 5n^3 + 10n - 3$

2. (15 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). Show your work!

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

Algorithm B: $T(n) = 4T(n/2) + 2n^2 + 100 \log_{10} n - 5$

Algorithm C: $T(n) = 4T(n/8) + 5\sqrt{n} \cdot \ln n$

Algorithm D: $T(n) = 3T(n/2) + 5n + 10 \log_2 n$

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

3. (30 points) In this problem you will determine the asymptotic time complexity of new algorithms. For each part you should give a recurrence equation for the algorithm and then solve it (using the Master method) to get the asymptotic time complexity. Be sure to explain the derivation of each recurrence equation including the asymptotic time complexity for both the dividing and combining phases of the algorithm.

(a) Here we consider a variant of the the divide-and-conquer algorithm where instead of dividing the points into a left and right half, you divide the point into three thirds (left, middle, right) each with roughly 1/3 of the points. You can recursively solve the 3 subproblems and then you can combine in the same basic way except you'll need to create two "Ystrips" and apply the combining algorithm over both.

(b) Here we consider a new divide-and-conquer algorithm for finding the closest pair of points that will create 4 subproblems, the left half, the right half, the top half, and the bottom half. The recursive procedure `NewClosestPair` finds the closest pair among the points with x-coordinate between `ptsByX[x1].x` and `ptsByX[x2].x`, and with y-coordinate between `ptsByY[y1].y` and `ptsByY[y2].y`.

The initial call is `NewClosestPair(ptsByX,0,n-1,ptsByY,0,n-1)` where n is the number of points (i.e. `ptsByX.length`).

```
NewClosestPair(ptsByX,x1,x2,ptsByY,y1,y2){
// As in standard divide-and-conquer algorithm terminate
// if there are 1 or 2 points returning the right answer
    xmid = (int) (x1 + x2)/2
    ymid = (int) (y1 + y2)/2
    pLeft = NewClosestPair(ptsByX,x1,xmid,ptsBy,y1,y2);
    pRight = NewClosestPair(ptsByX,xmid+1,x2,ptsBy,y1,y2);
    pBottom = NewClosestPair(ptsByX,x1,x2,ptsBy,y1,ymid);
    pTop = NewClosestPair(ptsByX,x1,x2,ptsBy,ymid+1,y2);
// Let P be the closest pair of points among those returned
// in the four recursive calls above
// Let d be the distance between the pair in P
// Create and fill an array A of points within distance d of
// (ptsByX[xmid].x,ptsByY[ymid].y)
// Find the closest pair P' among the points in A
// Return the closest pair between P' and P
}
```

- (c) Here is a new sorting algorithm (and it does really work, convince yourself why). The original call made will be `NewSort(A,0,n-1)` where A is an array of integers.

```
void NewSort(int A[],int i,int j){  \\ sorts the subarray A[i..j]
    if (j == i+1)                  \\when there are only 2 elements
        if (A[i] > A[j]) swap(A,i,j)  \\swaps A[i] and A[j]
    else {
        int k = (j-i+1)/3;
        NewSort(A,i,j-k);           \\ sort first two thirds
        NewSort(A,i+k,j);           \\ sort second two thirds
        NewSort(A,i,j-k);           \\ sort first two thirds again
    }
}
```

4. (35 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). 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.

Useful Observations: (1) Multiplying a number by a power of 10 can be implemented as a shift operation, (2) The addition or subtraction of two $\Theta(n)$ digit numbers can be a in $\Theta(n)$ time (by breaking it down into $\Theta(n)$ 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 $\lfloor n/2 \rfloor$ 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 .
Using the fact that $x = x_\ell \cdot 10^{\lfloor n/2 \rfloor} + x_r$ and $y = y_\ell \cdot 10^{\lfloor n/2 \rfloor} + y_r$, 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 and clearly describe what recursive calls are made, and what you do with the results of the recursive calls to obtain the final solution.
After clearly describing your algorithm you should give and solve a recurrence equation for it to determine its asymptotic time complexity.
Hints: 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 equation to find the asymptotic time complexity.
- (d) Of the three algorithms you have developed which is the fastest?

Extra Credit Problems:

Only work on these after you have completed the required problems.

- 4* (3 points) Give a tight asymptotic for the following recurrence when n is any power of 2. $T(1) = 1$, and for $n > 1$, $T(n) = 2T(n/2) + n/(\log_2 n)$.

Hint: Use a recurrence tree. As shown in the text, using integrals to bound the harmonic series $H_m = \sum_{i=1}^m \frac{1}{i}$ by $\ln(m+1) \leq H_m \leq (\ln m) + 1$.

- 5* (2 points) Which divide-and-conquer closest pair algorithm would you recommend between that from 3a and the one we described in class. If they have the same asymptotic time complexity you will need to think about the constants hidden within the asymptotic notation to see if you can figure out which will be faster. You may need to make an assumption about the distribution of points. That is fine, but you should clearly state whatever assumption(s) you make.