

## Homework 1

January 29, 2008

Due Date: February 5

Practice problems with solutions for similar kinds of problems can be found on the course web page under “Homeworks”.

1. (15 points) For each problems suppose you have algorithms  $A_1$  and  $A_2$  with worst-case time complexity of  $T_1(n)$  and  $T_2(n)$ , respectively. For each part, answer the 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  $A_2$ ? Briefly justify your answers. **Please submit this on the template on the back of the cover sheet.**

- (a)  $T_1(n) = 5\sqrt{n} + \log_2 n + 3$ , and  $T_2(n) = 100n / \log_2 n + 25$   
 (b)  $T_1(n) = 2 \log_{16} n$ , and  $T_2(n) = \log_2 n + 10$   
 (c)  $T_1(n) = n^2 + \log_{10} n - 20$ , and  $T_2(n) = (\log n)^2 + 20n\sqrt{n}$   
 (d)  $T_1(n) = 5\sqrt{n} + (\ln n)^2$ , and  $T_2(n) = \sqrt{n}$   
 (e)  $T_1(n) = 2^{(\log_4 n)} + \sqrt{n}$ , and  $T_2(n) = n^{3/4} \cdot \log_2 n$

2. (15 points) For each of the following recurrences,  $T(0) = T(1) = \Theta(1)$ . You are to give tight asymptotic bounds for  $T(n)$  (i.e. it's sufficient to use  $\Theta$  notation so use the master method whenever you can). Show your work!

- (a) **Algorithm A:**  $T(n) = T(3n/4) + 2n - 4$   
 (b) **Algorithm B:**  $T(n) = 4T(n/2) + n^2 \cdot \log_2 n + 100 \log_{10} n$   
 (c) **Algorithm C:**  $T(n) = 2T(n/4) + \sqrt{n} + 10 \log_2 n$   
 (d) **Algorithm D:**  $T(n) = 3T(n/2) + 2n \ln n + 10n$   
 (e) **Algorithm E:**  $T(n) = T(n - 2) + n$   
 (f) Assuming that all of these algorithms solve the same problem, which one would you recommend using. Briefly explain.

3. (30 points) For each part, first give a recurrence relation for the stated 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's a sorting algorithm (and it does work). The initial call is `magicSort(A, 0, A.length-1)` where `A` is an array of integers. After doing what is requested above, let us know if you'd recommend using `magicSort`.

```
void magicSort(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;
    magicSort(A,i,j-k);              \ \ sort first two thirds
    magicSort(A,i+k,j);              \ \ sort second two thirds
    magicSort(A,i,j-k);              \ \ sort first two thirds again
  }
}
```

- (b) Consider the following divide-and-conquer algorithm (shown using informal pseudo-code) 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 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){
    If there are 1 or 2 points, return the correct 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 Square of points within distance d of
        (ptsByX[xmid].x,ptsByY[ymid].y)
    Find the closest pair P' among the points in Square
    Return the closest pair between P' and P
}
```

Observe that the array, `Square`, can only hold a constant number of points. You can use this fact without proving it.

- (c) Next consider a variant of the the divide-and-conquer algorithm you implemented in Lab 1 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 step over both yStrips.

4. (40 points) In this problem you will develop algorithms to multiply two  $n$ -digit numbers  $x$  and  $y$  where  $n$  is very large and thus must be represented using a data structure such as a list or array where each element holds a single digit. Suppose that in one instruction you can only multiply, add or subtract two 1-digit numbers. Your task is to 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 performed in  $\Theta(n)$  time (by breaking it down into  $\Theta(n)$  additions of two one digit numbers).*

- (a) (5 pts) Consider 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) (12 pts) Let's now design a divide-and-conquer algorithm for this problem. I'll get you started. Divide  $x$  into  $x_\ell$  and  $x_r$  where  $x_\ell$  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  $y_\ell$  and  $y_r$ .

Using the fact that  $x = x_\ell \cdot 10^{\lceil n/2 \rceil} + x_r$  and  $y = y_\ell \cdot 10^{\lceil n/2 \rceil} + y_r$ , complete the design of this algorithm, clearly describing it. You need not go into details about how to create  $x_\ell$ ,  $x_r$ ,  $y_\ell$ , and  $y_r$  but should discuss the worst-case time to do this splitting. You are expected to clearly state the recursive calls made (including the arguments), 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: Multiply out the formulas given above for  $x$  and  $y$ . Could you make use of the product of any  $\lceil n/2 \rceil$  digit numbers?*

- (c) (20 pts) 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) (3 pts) 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.

- 5\* (3 points) Give a tight asymptotic bound for the following recurrence when  $n$  is any power of 3.  $T(1) = 1$ , and for  $n > 1$ ,  $T(n) = 3T(n/3) + 2n/(\log_3 n)$ .

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

- 6\* (2 points) Which divide-and-conquer closest pair algorithm would you recommend among 3b, 3c, and the one we described in class. If any have the same asymptotic time complexity you will need to think about the constants hidden within the asymptotic notation. 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.