

Practice Problems for Homework 1

1. (15 pts) Fill in the following table. For the last column, you are to assume that your goal is to pick the fastest algorithm (based on the worst-case running time) with A_1 having worst-case time complexity of $T_1(n)$ and A_2 having worst-case time complexity of $T_2(n)$. In addition to the table, please give us a very brief explanation for how you determined your answer.

	$T_1(n)$	$T_2(n)$	Is $T_1(n) = O(T_2(n))$?	Is $T_1(n) = \Omega(T_2(n))$?	Is $T_1(n) = \Theta(T_2(n))$?	Which is best?
a	$25n \ln n + 5n$	$\frac{1}{2}n \log_2 n$				
b	$\frac{1}{2}n^2 + n \log_2 n$	$5n \log_2 n$				
c	$\sqrt{n}(\log_2 n)$	n				
d	$2^{\log_2 n}$	$2n^2$				
e	$n\sqrt{n}$	$n^{1.4}$				

2. For each of the following program fragments compute the worst-case asymptotic time complexity (as a function of n). Whenever it says “loop body” you can assume that a constant number of lines of code are executed there. (*Hint: Write nested summations to express the number of times each of the loop bodies is executed.*)

(a) `for (i=0; i<=n-1; i++)
 for (j=1; j<=10 j++)
 loop body`

(b) `for (i=0; i<=n-1; i++)
 loop body
 for (i=0; i<=n-1; i++)
 for (j=i; j<= n-1; j++)
 loop body`

3. (12 pts) Suppose that you are to design an algorithm to solve a problem given to you. You have thought of 5 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: $T(n) = T\left(\frac{3n}{4}\right) + 3n^2 + n$

Algorithm B: $T(n) = 4T(n/4) + 3$

Algorithm C: $T(n) = 3T(n/2) + 3n \ln n$

Algorithm D: $T(n) = 4T(n/2) + \Theta(n^2 \log_2 n)$

Algorithm E: $T(n) = T(n - 1) + 2n$

Which is best? Suppose that the above recurrences describe the exact time complexities of different algorithms to solve the same problem. Which algorithm is the fastest?

4. In this problem you are to analyze the asymptotic time complexity $T(n)$ for the following two divide-and-conquer algorithm. You may assume that n is a power of 2. (NOTE: It doesn't matter what these do!)

```
(a) int foo(A){
    n = A.length;
    if (n==1){
        return A[0];
    }
    int half = (int) n/2;
    int[] A1 = new int[half];
    int[] A2 = new int[n-half];
    for (int i=0; i <= half-1; i++){
        for (int j=0; j <= 1; j++){
            if (j==0) A1[i] = A[2i]
            else A2[i] = A[2i+1];
        }
    }
    A2[n-half-1] = A[n-1];

    b1 = foo(A1);
    b2 = foo(A2);

    if (b1>b2) return b1;
    else return b2;
}

(b) float foo(A){
    n = A.length;
    if (n==1){
        return A[0];
    }
    let A1,A1,A3,A4 be arrays of size n/2
    for (i=0; i <= (n/2)-1; i++){
        for (j=0; j <= (n/2)-1; j++){
            A1[i] = A[i];
            A2[i] = A[i+j];
            A3[i] = A[n/2+j];
            A4[i] = A[j];
        }
    }

    b1 = foo(A1);
    b2 = foo(A2);
    b3 = foo(A3);
    b4 = foo(A4);

    return b1*b2*b3*b4;
}
```

5. Complete the following divide-and-conquer algorithm to determine if all integers in an array a are equal. The initial call would be `allEqual(a,0,a.length-1)`. (Yes, there is an easy iterative algorithm for this problem. The goal here is to provide practice with the design and analysis of divide-and-conquer algorithms.)

```
boolean allEqual(int a[],int p,int r){
    if (p == r)
        return true;
    if (A[p] != A[r])
        return false;

    //take it from here
}
```

Write a recurrence relation for your algorithm and then solve it to obtain the worst-case asymptotic time complexity for your algorithm.