Maggie Johnson

Handout #15
CS109

due: 2/11, 108 points

Problem Set #4

This problem set will give you practice with big-oh analyses from a mathematical and an algorithmic point of view. At times, we will ask for exact instruction counts and other times, just a big-oh approximation. Both are important skills for computer scientists. You will also work on some recurrence relations, and the analysis of recursive algorithms. It is likely you will find the recurrence relations and recursion problems the most challenging, so be sure to allow enough time.

On this problem set, if we ask for a proof, be formal and precise (i.e., use a statement and reason chart format, or a detailed paragraph if you are doing a counterexample). If we ask for an inductive proof, include the six steps. On all questions, always show your work so partial credit can be awarded.

Big-Oh

Rosen, Section 1.8

#8	(5 points)	
#32	(5 points)	try and make this a "tight" upper bound, i.e., just about everything is $O(n^n)$ but that is not a tight bound.
#38	(8 points)	

Algorithm Analysis

1) You are given an increasing sequence of numbers u1, u2, ... um, and a decreasing series of numbers d1, d2, ...dn. You are given one more number C and asked to determine if C can be written as the sum of one ui and one dj. There is an obvious brute force method, just comparing all the sums to C, but there is a much clever-er solution. Define an algorithm that works in linear time. Do a line-by-line instruction count (worst case) of your algorithm to show it is linear. (15 points)

2) Give a formula representing the exact number of instructions executed in the worst case for the following function, and then express this in big-oh notation. (10 points)

```
void matrixmult(A, B, C, n);
int A[][n], B[][n], C[][n], n;
{
        int i, j, k;
        for (i = 1; i <= n; i++)
                for (j = 1; j <= n; j++)
                        for (k = 1; k <= n; k++)
                                A[i][j] := A[i][j] + (B[i][k] * C[k][j]);
}
```

3) Suppose that line (1) in the function foo of the C program in *Analysis of Algorithms* handout (Analyzing Programs with Non-Recursive Subprogram Calls) were the following:

```
for (i = 1; i < bar(n,n); i++)
```

What would the running time of the main program be? We are only asking for a big-oh estimate here rather than exact number of instructions, but give details of your analysis. (10 points)

Recurrence Relations

4) A sequence is defined recursively as follows:

$s_0 = 1$; $s_1 = 2$
$s_k = 2 * s_{k-2}$ for all integers k >= 2

a) Find a closed form formula for this sequence by enumerating some elements and guessing. (5 points)
b) Use *strong* induction to prove your formula is correct. (10 points)

5) Discover a closed form formula for:

$$\sum_{k=1}^{n} k^2$$

Prove your formula is correct using induction. (15 points)

6) A sequence is defined recursively as follows:

$u_0 = 1$
$u_k = 1 + 3\,u_{k-1}$ for all integers k >= 1

Find a closed form formula for this sequence by repeated substitution. (10 points)

7) The greatest common divisor (GCD) of two integers i and j is the largest integer that divides both i and j evenly. For example GCD(30, 24) = 6 and GCD(35, 24) = 1.

a) Write a recursive function that takes as input two integers i and j, with i > j, and returns GCD(i, j). (5 points)
b) What is the running time of this function? Set up a recurrence relation and solve it. (10 points)