CS251, Spring 2013
Homework 2

Due: Monday, Feb 4 at the beginning of class.
As always, write neatly and staple your work.

Part I: From Aho and Ullman:

3.8.2, 3.8.3

Part II:

(1) In elementary school you learned algorithms for addition and multiplication. You were the computer!

For both multiplication and addition, there are just a couple of primitive operations operating on digits which the entire procedure is built:

Given Digits (a, b, c), determine the two digit sum a+b+c:

pq (where p is the "carry-out" and q is the 1's place of the sum.)

Given Digits (a, b, c), determine the two digit value $ab + c$

Assume that these operations are constant time (essentially lookups). In big-Oh terms, how long does it take to:

(a) Add two n-digit integers
(b) Multiply two n-digit integers

Give tight bounds. We expect an argument of correctness (not just "it's O(XXX)").

(2) Give tight runtime bounds for each of the 4 code segments below. One is a little tricky!

```
/* A */
for(c1=0, i=1; i<=n; i++)
    for(j=1; j<= n; j++)
        c1++;


/* B */
for(c2=0, i=1; i<=n; i++)
    for(j=1; j<= i; j++)
        c2++;

/* C */
for(c3=0, i=1; i<=n; i = 2*i)
    for(j=1; j<= n; j++)
        c3++;

/* D */
for(c4=0, i=1; i<=n; i = 2*i)
    for(j=1; j<= i; j++)
        c1++;
```

(3) Give a tight runtime bound for the following code segment. Pay close attention to the conditions under which certain code segments are executed. and to the loop bounds.

```
x=0;
for(i=0; i < n*n; i++) {

    if(i % n == 0) {
        for(j=0; j<n; j++)
            x += (i+j);
    }
    else
            x--;
}
```

(4) Recursive max. You all know how to write a C function to find the maximum value in an array of integers. In this problem, you are going to write a recursive max-finding function which works a little differently:

> Frame the problem as: given a[], and i, j, determine the largest value in a[i..j].
>
> Then the max for the entire array is determined by using i=0 and j=n-1.
>
> If i==j, the max of the subarray must be a[i] (base case)
>
> Otherwise divide the subarray into two roughly equal parts and do the following:
>
> > recursively compute the max of the left subarray
> >
> > recursively compute the max of the right subarray.
> >
> > return the largest of these two values.

(4a) Write a C function implementing this algorithm. You can write it on pencil and paper for this assignment.

(4b) Can you come up with a tight runtime bound for the algorithm?

(4c) Can you imagine any scenario in which looking at the problem this way might be useful?