CSE 326 DATA STRUCTURES
HOMEWORK 1

Due: **Friday, October 5, 2007** at the beginning of class.

Problem 1. Induction gone wrong

Induction is a key tool in the analysis of the running times of algorithms, especially for recursive algorithms. This problem is designed to test out your inductive proving skills by spotting the error in an inductive proof. (Original idea due to Polya and Kaser.) The "theorem" that follows is not true. Thus, we hope that any proof of it must be invalid. Find the (biggest) logical error in the proof. Please assume a world where every book has exactly one author (no co-authors, no anonymous publications).

> **Theorem 1. (Wrong!)** In any collection of n books, all the books have the same author.
>
> **Proof of Theorem 1 (Wrong!)**: The proof is by induction on n.
>
> Base case (for $n = 1$): If the collection consists of a single book, then every book in the collection has this author.
>
> Induction step: $(n = k)$. Assume the theorem holds for $n = k - 1$. Then, consider a collection of k books. Consider any group of $k - 1$ of them. By the induction hypothesis, all books in the group have the same author (let's say author A). Consider any other group of $k-1$ of the k books. Similarly, the induction hypothesis has all of these sharing the same author. Since the two "size $k - 1$ out of k" groups must have $k - 2$ books in common, and since a book cannot change its author, all the books in second group have A as their author too. Between them, all k books are in one group or the other (or both). So, we see that all k books have the same author. The proof then holds by mathematical induction.

Problem 2. Induction gone right

Consider the following algorithm for counting the number of 1s in the binary representation of an unsigned integer:

```
unsigned integer countOnes(unsigned integer x)
  if x = 0
    return 0
  else
    return (x mod 2) + countOnes(x/2)
```

Prove using induction that `countOnes` correctly returns the number of 1s in the binary representation of `x`.

Problem 3. A few number games

(a) Evaluate $\sum_{i=0}^{\infty} \frac{1}{4^i}$

(b) Evaluate $\sum_{i=0}^{\infty} \frac{i}{4^i}$

(c) Prove $\sum_{i=1}^{N} i^3 = (\sum_{i=1}^{N} i)^2$

(d) Evaluate $2^{1123} - 6^{58} (\text{mod } 5)$

Make sure to not just evaluate the sum, but show us how you performed this evaluation. For the modulus problem, you should not need to use a calculator at all.

Problem 4. Algorithm analysis

For each of the following six programs, give an analysis of the running time (Big-Oh will do).

(a)
```
sum = 0;
for( i = 0; i < n; i++ )
  sum++;
```

(b)
```
sum = 0;
for( i = 0; i < n; i++ )
  for( j = 0; j < n; j++ )
    sum++;
```

(c)
```
sum = 0;
for( i = 0; i < n * n * n; i++ )
  for( j = 0; j < n * n; j++ )
    sum++;
```

(d)
```
sum = 0;
for( i = 0; i < n; i++ )
  for( j = 0; j < i; j++ )
    sum++;
```

(e)
```
sum = 0;
for( i = 0; i < n; i++ )
  for( j = 0; j < i * i; j++ )
    for( k = 0; k < j; k++ )
      sum++;
```

(f)
```
sum = 0;
for( i = 0; i < n; i++ )
  for( j = 0; j < i * i; j++ )
    if (j % i == 0)
      for( k = 0; k < j; k++ )
        sum++;
```

(g)
```
sum = 0;
for( i = 0; i < n; i++ )
{
  for( j = 0; j < n; j++)
    sum++;
  for( k = 0; k < i; k++)
    sum++;
}
```

Problem 5. Big-O, Big-Θ

Big-O, Big-Θ, and Big-Ω are the ubiquitous language of the analysis of algorithms. Getting you head around what these notations mean is essential for understanding pretty much any theoretical analysis of an algorithm.

Prove true or explain why the following statements are incorrect:

(a) If $f(n) = O(g(n))$ and $h(n) = O(k(n))$, then $f(n) - h(n) = O(g(n) - k(n))$.

(b) If $f(n) = O(g(n))$ and $h(n) = O(k(n))$, then $f(n) + h(n) = O(g(n) + k(n))$.

(c) $(2^n)^{1/3} = \Theta(2^n)$

(d) $(2^n)^{1/3} = 2^{\Theta(n)}$

Problem 6. Horner's Rule

The classic way to evaluate a polynomial is called Horners rule which can be stated recursively as follows. Let $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$. To compute $p(c)$ for some constant c, first evaluate $q(c)$ where $q(x) = a_1 + a_2x + \cdots + a_nx^{n-1}$ recursively, then $p(c) = a_0 + cq(c)$.

(a) Provide a base case for this method, and show the method works mathematically.

(b) For a polynomial of degree n, as a function of n, how many additions and how many multiplications are used to evaluate the polynomial in Horner's rule.

(c) Provide an elegant pseudocode function for Horner's rule where the data coefficients are stored in an array A[0 . . . n], with A[i] containing a_i.