# Solutions to Math 416 Homework Assignment 1

September 20, 2004

## 1 Problem 2.1-3

Linear search pseudocode:

```
LINEAR-SEARCH(A,v)
for( i ← 1; i ≤ length[A]; i++){
    if( A[i] == v)
        return(i);
}
return( NIL);
```

Loop invariant:

At the start of a loop iteration, $v$ is not among $A[1], \ldots, A[i-1]$.

(That is, if $1 \leq j$ and $j < i$, then $v \neq A[j]$.)

Initialization: There is no $j$ with $1 \leq j < 1 = i$, so the the statement holds.

Maintenance: Suppose the statement is true at a particular iteration $k$. We consider two cases. If $A[i] == v$, then the code will return $i$ and there is no next $(k+1)$'st iteration. If $A[i] \neq v$, then, using the loop invariant, we know $v$ is not among $A[1], \ldots, A[i_k]$, where $i_k$ is the value of $i$ during the $k$'th iteration. At the start of the next iteration, we have $v$ is not among $A[1], \ldots, A[i_k] = A[i_{k+1} - 1]$, so the loop invariant holds.

Termination: The loop may terminate for two reasons. If it terminates early, then it returns $i$ with $A[i] == v$, which is correct. Otherwise, the code returns NIL. In that case, by the loop invariant, $v$ is not among $A[1], \ldots, A[length[A]]$, so the NIL output is correct.

## 2 Problem 2.3-5

Binary search pseudocode:

```
BINARY-SEARCH(A,v) //A indexed from 0 to length[A] - 1
if length[A] = 0
    return( NIL);
m ← ⌊length[A] / 2⌋;
if (v > A[m])
    return BINARY-SEARCH(A[0..m − 1], v);
if (v == A[m])
    return(m);
// if (v < A[m])
return BINARY-SEARCH(A[m + 1..length[A]], v); // indices get relabeled to start from zero
```

If $T(n)$ is the worst-case cost of the algorithm on input sequences of length $n$, then, for some $c$,

$$T(n) \leq \begin{cases} T(\lfloor n/2 \rfloor) + c, & n > 0; \\ c, & n = 0. \end{cases}$$

This is because, for $n > 0$, a call on an array of length $n$ results in one recursive call on an array of length at most $n/2$ and, by induction, one can show that $T$ is monotonically increasing.

We now show by induction that $T(n) \leq c(\lg(n) + 2)$ for $n > 0$. (We avoid $\lg(0)$.)

First, $T(0) \leq c$. It follows that $T(1) \leq T(0) + c \leq 2c = c(\lg(n) + 2)$.

Suppose that, for all $m > 1$ and all $n < m$, we have $T(n) \leq c(\lg(n) + 2)$. Now consider $T(m)$. Note that $m/2 < m$, so this is covered inductively. We have $T(m) \leq T(m/2) + c \leq c(\lg(m/2) + 2) + c = c(\lg(m) + 1) + c = c(\lg(m) + 2)$.

Informally, we are allowed to assume that a binary tree of height $h$ has $2^h$ leaves and $2^{h+1} - 1$ nodes. Assuming that the original string has $2^{h+1} - 1$ elements, we can consider a binary tree of height $h$, whose nodes are associated with input elements, and whose in-order traversal enumerates the elements in sorted order. Then binary search follows a path from root downwards to some node (not necessarily a leaf). Thus the worst-case time cost is the height of the tree. As long as the degree of the tree is at least 2 and at most $O(1)$, the height of the tree is $O(\log(n))$ and we don't need to be any more careful about how the tree branches.

# 3 Problem 2-3

a. The asymptotic running time is $\Theta(n)$, since there are $n$ loop iterations and each takes some constant amount of time.

b. Naive polynomial pseudocode:

```
NAIVE-POLY(a_0, a_1, ..., a_n; x)
y ← 0
for( i ← 0; i ≤ n; i++){
    z ← 1
    for( j ← 0; j < i; j++)
        z *= x;
    y += a_i · z;
}
// y is set to the output
```

Note we are evaluating the polynomial from lowest degree to highest degree.

For some constant, the runtime is $\sum_{0 \leq i \leq n} \sum_{0 \leq j < i} c$. This is

$$\begin{aligned} \sum_{0 \leq i \leq n} \sum_{0 \leq j < i} c &= \sum_{0 \leq i \leq n} ci \\ &= c\frac{n(n+1)}{2} \\ &= \Theta(n^2), \end{aligned}$$

which we can informally see from looking at the loops. (E.g., consider the runtime for the last $n/2$ iterations of the outer loop. There, $i \geq n/2$ so the inner loop executes at least $n/2$ times for each iteration of the outer loop, so the inner loop iterates at least $(n/2)^2$ times altogether. On the other hand, $i$ and $j$ are at most $n$, so the inner loop iterates at most $n^2$ times.)

c. Initialization. At the start, $i = n$, so $n - (i + 1) = -1$. The sum $\sum_{k=0}^{-1}$ is the empty sum, zero, which equals $y$.

Maintenance. Suppose the invariant is true at the start of some loop. During that loop, $y$ gets

$$\begin{aligned}
a_i + x \cdot y &= a_i + \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^{k+1} \\
&= \sum_{k=-1}^{n-(i+1)} a_{k+i+1} x^{k+1} \\
&= \sum_{k'=0}^{n-i} a_{k'+i} x^{k'}, \quad \text{for } k' = k+1 \\
&= \sum_{k'=0}^{n-((i-1)+1)} a_{k'+(i-1)+1} x^{k'} \\
&= \sum_{k'=0}^{n-(i'+1)} a_{k'+i'+1} x^{k'},
\end{aligned}$$

where $i' = i - 1$ is the value of $i$ at the start of the next iteration.

Termination. At termination, we have $y = \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k$ for $i = -1$; *i.e.*, $y = \sum_{k=0}^{n} a_k x^k$.