Assignment 1 - CS 4071 - Spring 2018

Due: 2018-01-29 Group #13: Robert DiMartino (dimartrt), Hayden Schiff (schiffha), Jeremiah Leak (leakjz)

1. Exercise 1.1

Problem: Trace the action of the left-to-right binary method to compute:

a. x^{123}

$$\begin{gathered}123 = 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 \\ 123 = 1111011_2 \\ x \to x \times (x)^2 = x^3 \to x \times (x^3)^2 = x^7 \to x \times (x^7)^2 \\ = x^{15} \to (x^{15})^2 = x^{30} \to x \times (x^{30})^2 = x^{61} \to x \times (x^{61})^2 = x^{123}\end{gathered}$$

b. x^{64}

$$\begin{gathered}64 = 2^6 \\ 64 = 1000000_2 \\ x \to (x)^2 = x^2 \to (x^2)^2 = x^4 \to (x^4)^2 \\ = x^8 \to (x^8)^2 = x^{16} \to (x^{16})^2 = x^{32} \to (x^{32})^2 = x^{64}\end{gathered}$$

c. x^{65}

$$\begin{gathered}65 = 2^6 + 2^0 \\ 65 = 1000001_2 \\ x \to (x)^2 = x^2 \to (x^2)^2 = x^4 \to (x^4)^2 \\ = x^8 \to (x^8)^2 = x^{16} \to (x^{16})^2 = x^{32} \to x \times (x^{32})^2 = x^{65}\end{gathered}$$

d. x^{711}

$$\begin{gathered}711 = 2^9 + 2^7 + 2^6 + 2^2 + 2^1 + 2^0 \\ 711 = 1011000111_2 \\ x \to (x)^2 = x^2 \to x \times (x^2)^2 = x^5 \to x \times (x^5)^2 = x^{11} \to (x^{11})^2 = x^{22} \to (x^{22})^2 \\ = x^{44} \to (x^{44})^2 = x^{88} \to x \times (x^{88})^2 = x^{177} \to x \times (x^{177})^2 = x^{355} \to x \times (x^{355})^2 = x^{711}\end{gathered}$$

2-old. Exercise 1.7

Problem: Trace the action of the algorithm *GCD* for the following input pairs:

a. (24, 108)

$$\begin{gathered}\mathrm{NaiveGCD}(24,108)\\ =\gcd(24,84)=\gcd(24,60)=\gcd(24,36)\\ =\gcd(24,12)=\gcd(12,12)=12\end{gathered}$$

b. (23, 108)

$$\begin{gathered}\mathrm{NaiveGCD}(23,108)\\ =\gcd(23,85)=\gcd(23,62)=\gcd(23,39)\\ =\gcd(23,16)=\gcd(7,16)=\gcd(7,9)=\gcd(7,2)\\ =\gcd(5,2)=\gcd(3,2)=\gcd(1,2)=\gcd(1,1)=1\end{gathered}$$

c. (89, 144)

$$\begin{gathered}\mathrm{NaiveGCD}(89,144)\\ =\gcd(89,55)=\gcd(34,55)=\gcd(34,21)\\ =\gcd(13,21)=\gcd(13,8)=\gcd(5,8)=\gcd(5,3)\\ =\gcd(2,3)=\gcd(2,1)=\gcd(1,1)=1\end{gathered}$$

d. (1953, 1937)

$$\begin{gathered}\mathrm{NaiveGCD}(1953,1937)\\ =\gcd(16,1937)=\gcd(16,1921)=\gcd(16,1905)=\gcd(16,1889)\\ =\cdots=\gcd(16,49)=\gcd(16,33)=\gcd(16,17)=\gcd(16,1)\\ =\gcd(15,1)=\gcd(14,1)=\cdots=\gcd(3,1)=\gcd(2,1)=\gcd(1,1)=1\end{gathered}$$

2-new. Exercise 1.8

Problem: Trace the action of the algorithm *EuclidGCD* for the following input pairs:

a. (24, 108)

$$\begin{gathered}\mathrm{EuclidGCD}(24,108)\\ =\gcd(108,24)=\gcd(24,12)=\gcd(12,0)=12\end{gathered}$$

b. (23, 108)

$$\begin{gathered}\text{EuclidGCD}(23,108)\\ =\gcd(108,23)=\gcd(23,16)=\gcd(16,7)\\ =\gcd(7,2)=\gcd(2,1)=\gcd(1,0)=1\end{gathered}$$

c. (89, 144)

$$\begin{gathered}\text{EuclidGCD}(89,144)\\ =\gcd(144,89)=\gcd(89,55)=\gcd(55,34)=\gcd(34,21)\\ =\gcd(21,13)=\gcd(13,8)=\gcd(8,5)=\gcd(5,3)\\ =\gcd(3,2)=\gcd(2,1)=\gcd(1,0)=1\end{gathered}$$

d. (1953, 1937)

$$\begin{gathered}\text{EuclidGCD}(1953,1937)\\ =\gcd(1937,16)=\gcd(16,1)=\gcd(1,0)=1\end{gathered}$$

3. Exercise 1.9

Problem: Give a formula for $\text{lcm}(a,b)$ in terms of $\gcd(a,b)$.

n is a multiple of x if there exists some integer p such that $x \times p = n$. Let $m = \text{lcm}(a,b)$. m must contain all of the prime factors of both a and b. Clearly, ab is a multiple of both a and b because $a \times b = ab = b \times a$ and ab contains all of the prime factors of both a and b. Is there a smaller common multiple? Yes! ab is double-counting all of the prime factors that a and b share. The product of these shared factors is $\gcd(a,b)$. We can divide those shared factors out of the product and see that this is still a common multiple of both a and b because:

$$a \times \frac{b}{\gcd(a,b)} = \frac{ab}{\gcd(a,b)} = b \times \frac{a}{\gcd(a,b)}$$

And $\gcd(a,b) \geq 1$ therefore $\frac{ab}{\gcd(a,b)} \leq ab$. So we arrive at our formula for the $\text{lcm}(a,b)$:

$\text{lcm}(a,b) = a \times b \div \gcd(a,b)$

4. Exercise 1.17

Problem: Trace the action of Horner's rule for the polynomial $7x^5 - 3x^3 + 2x^2 + x - 5$.

Horner's rule for 5th degree polynomial:

$((((a_5 \times x + a_4) \times x + a_3) \times x + a_2) \times x + a_1) \times x + a_0$

For the polynomial $7x^5 - 3x^3 + 2x^2 + x - 5$ the coefficients are:

$a_5 = 7, \quad a_4 = 0, \quad a_3 = -3, \quad a_2 = 2, \quad a_1 = 1, \quad a_0 = -5$

So Horner's rule is:

$((((7 \times x + 0) \times x + -3) \times x + 2) \times x + 1) \times x + -5$

For $x = 7$:

$$\begin{gathered} ((((7 \times 7 + 0) \times 7 + -3) \times 7 + 2) \times 7 + 1) \times 7 + -5 \\ ((((49) \times 7 + -3) \times 7 + 2) \times 7 + 1) \times 7 + -5 \\ (((340) \times 7 + 2) \times 7 + 1) \times 7 + -5 \\ ((2,382) \times 7 + 1) \times 7 + -5 \\ (16,675) \times 7 + -5 \\ 116,720 \end{gathered}$$

5. Exercise 2.13

Problem: Design an algorithm that tests whether or not two input lists of size n have at least one element in common. Give formulas for $B(n)$ and $W(n)$ for your algorithm.

function $\text{CheckShareElements}(a[0:n-1], b[0:n-1])$
Input: $a[0:n-1], b[0:n-1]$ (two arrays of real numbers)
Output: `true` if a and b have at least 1 element in common, else `false`

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        if (i == j) return true
return false
```

If $a[0] = b[0]$, then it'll return true after the very first comparison. Thus, $B(n) = 1$.

If $a \cap b = \emptyset$ (or if the only common element is the last element of a and the last element of b), then it'll have to compare every element of a to every element of b. Thus, $W(n) = n^2$.

6. Exercise 2.16

Part a.

Problem: Write a procedure that finds both the largest and second-largest elements in a list $L[0:n-1]$ of size n.

function $\text{FindTwoLargest}(L[0:n-1])$
Input: $L[0:n-1]$ (an array of real numbers)
Output: a and b (the largest two numbers from L)

```
# start by assuming the first two elements are the largest
a = L[0]
b = L[1]
if (b > a) # make sure a is bigger than b
    temp = a
    a = b
    b = temp

# search the list for bigger numbers, and update a and b as needed
for (i = 2; i < n; i++)
    if (L[i] > a)
        b = a
        a = L[i]
    elif (L[i] > b)
        b = L[i]

return a, b
```

Part b.

Problem: Determine $B(n)$ and $W(n)$ for the algorithm in a.

Our basic operation is comparisons. We do one comparison at the beginning, before the loop. Then on each iteration of the loop, we do one comparison if $L[i] > a$; if not, we end up doing two comparisons. Thus, our best case would be if the list sorted from smallest to largest (meaning $L[i] > a$ will be true on every iteration). Our worst case would any list where none of the other elements are larger than $L[0]$ or $L[1]$ (meaning $L[i] > a$ will be false on every iteration). Thus:

$$B(n) = 1 + (n - 2) = n - 1$$

$$W(n) = 1 + 2(n - 2) = 2n - 3$$

7. Exercise 2.30

Part a.

Problem: Design a recursive version of *InsertionSort*.

function $\text{recursiveInsertionSort}(a[0 : n - 1], k)$
Input: $L[0 : n - 1]$ (an array of n elements), k (an index of list L)
Output: $L[0 : n - 1]$ with elements $L[0 : k]$ sorted in ascending order

```
# initial condition
# a list of one element is already sorted
if k==0:
    return
```

```
endif

# recursively call function to sort up to index k-1
recursiveInsertionSort(L[0:n-1], k-1);

insertElement ← L[k] # element to sort
i ← k-1
# shift all elements greater than the
# insertElement up one position in L
while i >= 0 and L[i] > insertElement:
    L[i+1] ← L[i]
    i ← i-1
endwhile
L[i] ← L[k]
```

Part b.

Problem: Design a recursive linked list version of *InsertionSort*.

function $\text{recDLLInsertionSort}(a[0:n-1], k)$
Input: $head$ (a pointer to the head node of a doubly linked list), $tail$ (a pointer to a node in the doubly linked list pointed to by $head$)
Output: A doubly linked list where the nodes from $head$ to $tail$ are sorted in ascending order

```
# initial condition
# a list of one node is already sorted
if head == tail:
    return
endif

# recursively call function to sort up to the
# node before tail
recursiveInsertionSort(head, tail.previous);

insertNode ← tail # node to sort

# set cursor and remove insertNode
cursor ← insertNode.previous
cursor.next ← insertNode.next
if tail.next is not null:
    tail.next.prev ← cursor
endif

while cursor is not null and cursor.data > insertNode.data:
    cursor ← cursor.previous
endwhile
```

```

# move insertNode into sorted position
if cursor is null: # insertNode is the smallest
    head.prev ← insertNode
    insertNode.next ← head
    insertNode.prev ← null
    head = insertNode
else:
    cursor.next.prev ← insertNode
    insertNode.next ← cursor.next
    insertNode.prev ← cursor
    cursor.next ← insertNode
endif
```

8. Exercise 1.19

Problem: Describe a modification of *HornerEval* that solves this particular evaulation problem using only $n+1$ multiplications and $n+1$ additions.

TA Note: This solution was found before the hint was given in class.

If we modified the polynomial to just look at the even powers of x, i.e. $P_{even}(x) = a_0 + a_2x^2 + a_4x^4 + \cdots$, we noticed that $P_{even}(x) = P_{even}(-x)$. This is because all even powers x^{2k} an be re-written $(x^2)^k$ and $(-x)^2 = x^2$.

Similiary, for a polynomial of just the odd powers of x, i.e. $P_{odd}(x) = a_1x + a_3x^3 + a_5x^5 + \cdots$, we noticed that $P_{odd}(-x) = -P_{odd}(x)$. This is because for all odd powers

$$(-x)^{2k+1} = (-x) \times (-x)^{2k} = (-x) \times (-x)^{2k} = -(x \times x^{2k}) = -x^{2k+1}$$

Using these two observations, we modified Horner's algorithm to keep track of two sums, one for the even powers of x and one for the odd powers of x. The extra multiplication comes from keep track of v^2 and the extra addition comes from the return when we have to sum and difference the even and odd sums.

function $\text{ModifiedHornerEval}(a[0:n], v)$
Input: $a[0:n]$ (an array of real numbers), v (a real number)
Output: the values of polynomial $P(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ at $x = v$ and $x = -v$

```
vSquared ← v*v
if n is even:
    EvenSum ← a[n]
    OddSum ← 0
```

```
else:
    EvenSum ← 0
    OddSum ← a[n]
endif
for i ← n-1 downto 0 do:
    if i is even:
        EvenSum ← EvenSum * vSquared + a[i]
    else:
        OddSum ← OddSum * vSquared + a[i]
    endif
endfor
OddSum ← OddSum*v
return(P(v) is EvenSum+OddSum and P(-v) is EvenSum-OddSum)
```