Zachary J. Bell

Dr. Roy

CS 1401

Lab 8

**Activity 1**

Pseudo-code iterFactorial(number)

1. if number is less than 0 give 0 and print “Number is undefined!”
2. else if number = 0 or number = 1 give 1
3. else
   1. for (i = number – 1; i > 0; --i)
      1. number 🡨 number \* i
   2. give number

Pseudo-code recFactorial(number)

1. if number is less than 0 give 0 and print “Number is undefined!”
2. else if number = 0 or number = 1, give 1
3. else give number \* recFactorial(number – 1)

**Activity 2**

Pseudo-code iterFibonacci(number)

1. if number less than or equal to 1, give 0
2. else if number less than or equal to 2, give 1
3. else
   1. A[0] = 0
   2. A[1] = 1
   3. for (i = 2; i less than number; ++i)
      1. A[i] 🡨 A[i – 1] + A[i – 2]
   4. give A[number – 1]

Pseudo-code recFibonacci(number)

1. if number is less than or equal to 1, give 0
2. else if number is less than or equal to 2, give 1
3. else give recFibonacci(number – 1) + recFibonacci(number – 2)

Challenge Question:

1. The method returns -811192543.
2. The problem is that at **n = 46** the return value is **1,134,903,170** and at **n = 47** the return value is **1,836,311,903**. **n = 48** should return **method(46) + method(47)** which is **2,971,215,073** however that number overflows the maximum allowed stored integer value of **2,147,483,647**. A simple fix would be to have n store and return as a long so it can reach a higher value without overflowing.

**Activity 3**

Pseudo-code iterSIOL(seed, number)

1. for (i = 0; i less than number; ++i)
   1. if i is equal to 0; result[0] 🡨 seed
   2. else; result[i] 🡨 SIOL(result[i – 1])
2. give result

Pseudo-code SIOL(number)

1. convert number to string A
2. for (i = 0; i less than A length; ++i)
   1. charCount <- 1
   2. for (j = i + 1; j less than A length; ++j)
      1. if A character at j is equal to A character at i
         1. ++charCount
         2. i 🡨 j
      2. else break loop
   3. B 🡨 B + charCount + A character at i
3. give A converted as an integer

**Activity 4**

Pseudo-code iterPalindrome (input)

1. convert input to all lower case
2. for (i = 0; i less than input length; ++i)
   1. if input character at i is not equal to input length – 1 – i
      1. give false
3. give true

Pseudo-code recPalindrome(input)

1. convert input to all lower case
2. if input length is equal to 0 or equal to 1
   1. give true
3. else if input character at 0 is equal to input character at input length – 1
   1. give recPalindrom(input substring from 1 to length – 1)
4. else give false

**Activity 5**

1. The method goes through all indexes in the array and finds the maximum integer and switches it with the last index integer. It then changes the parameters of the first loop so that it doesn’t evaluate the last index that has been set. The purpose of the method is to take an array and sort it lowest to highest.
2. Pseudo-code iterSelectionSort(A)
   1. for (size = length of A – 1; size greater than 0; --size)
      1. maximum 🡨 A[0]
      2. for (i = 1; i is less than or equal to size; ++i)
         1. if A[i] is greater than maximum
            1. maximum 🡨A[i]
            2. index 🡨i
      3. A[index] 🡨🡪 A[size]
3. Trace
   1. {5,4,3,2,1}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A[0] | A[1] | A[2] | A[3] | A[4] |
| Initial | 5 | 4 | 3 | 2 | 1 |
| 1st iteration | 1 | 4 | 3 | 2 | 5 |
| 2nd iteration | 1 | 2 | 3 | 4 | 5 |
| 3rd iteration | 1 | 2 | 3 | 4 | 5 |

* 1. {10,1,7,2}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A[0] | A[1] | A[2] | A[3] |
| Initial | 10 | 1 | 7 | 2 |
| 1st iteration | 2 | 1 | 7 | 10 |
| 2nd iteration | 1 | 2 | 7 | 10 |

1. Pseudo-code recSelectionSort(A)
   1. recSelectionSort(A, A length – 1)
2. Pseudo-code recSelectionSort(A, n)
   1. if n is not equal to 0
      1. for( i = 0; i less than n; ++i)
         1. if A[i] > A[n]
            1. A[i] 🡨🡪 A[n]
         2. recSelectionSort(A, n – 1)