Zachary Thompson
October 2, 2013

CS 315 Lab 5

1)

recf(46) = 2 * recf(23) + 1 = 2 * 35 + 1 = **71**
recf(23) = recf(11) + recf(12) = 16 + 19 = 35
recf(11) = recf(5) + recf(6) = 7 + 9 = 16
recf(12) = 2 * rec(6) + 1 = 2 * 9 + 1 = 19
recf(5) = recf(2) + recf(3) = 3 + 4 + 7
recf(6) = 2 * rec(3) + 1 = 2 * 4 + 1 = 9
recf(3) = recf(1) + recf(2) = 1 + 3 = 4
recf(2) = 2 * recf(1) + 1 = 2 * 1 + 1 = 3
recf(1) = 1

2)

```
long exp1(long x, int n) {
   if (n == 0) return 1.0;
   else if (n % 2 == 0) return exp1(x * x, n/2);
   else return x * exp1(x, (n-1)/2) * exp1(x, (n-1)/2);
}

long exp2(long x, int n) {
   if (n == 0) return 1.0;
   else if (n % 2 == 0) return exp2(x * x, n/2);
   else return x * exp2(x * x, (n-1) / 2);
}
```

3)

a)

f(1000) = 1 + f(500) = 1 + 13 = **14**
f(500) = 1 + f(250) = 1 + 12 = 13
f(250) = 1 + f(125) = 1 + 11 = 12
f(125) = 2 + f(62) = 2 + 9 = 11
f(62) = 1 + f(31) = 1 + 8 = 9
f(31) = 2 + f(15) = 2 + 6 = 8
f(15) = 2 + f(7) = 2 + 4 = 6
f(7) = 2 + f(3) = 2 + 2 = 4
f(3) = 2 + f(1) = 2 + 0 = 2
f(1) = 0

b) k, each time n can be divided evenly by 2 increment 1 so 2^k will naturally return k

c) 2(k – 1), each time the number is odd when divided by 2, increment 2, this will occur k-1 times.

4)

F(3) = 1 call to F(1)
F(4) = 1 call to F(1)
F(5) = F(3) + F(4) = 2 calls to F(1)
F(6) = F(4) + F(5) = 3 calls to F(1)
F(7) = F(5) + F(6) = 5 calls to F(1)
F(8) = F(6) + F(7) = 8 calls to F(1)
F(9) = F(7) + F(8) = 13 calls to F(1)
F(10) = F(8) + F(9) = **21 calls to F(1)**

The recursive calls form a tree structure, with every resulting grouping being a leaf node of the tree. Since m > n will never occur in this situation, the number of leaf nodes will be equal to the binomial coefficient. The tree will be full so we can conclude the number of parent nodes is equal to the number of leaf nodes – 1. Each node of the tree represents a call to the C function so we can conclude the total number of calls (including the first call) will be 252 (leaf nodes) + 251 (parent nodes) = 503.

10! / (5! * 5!) = 252 (leaf nodes)
252 -1 = 251 (parent nodes)

C(10, 5) = **503 calls**

5)

```
IntSLList::IntSLList(IntSLLNode* node) {
   head = node;
}

IntSLList* IntSLList::mult(const IntSLList* multList) const {

   if (multList -> head == NULL) {
      return new IntSLList();
   }
   else {
      IntSLList* current = smult(multList -> head -> info);

      IntSLList* smallerList = new IntSLList(multList -> head -> next);

      IntSLList* next = mult(smallerList);
      next -> addToHead(0);

      IntSLList* resultList = current -> add(next);

      delete next, current;

      return resultList;
   }
}

IntSLList* IntSLList::exp(IntSLList* x, long long n) {
   if (n == 0) {
```

```
        IntSLList* resultList = new IntSLList();
        resultList -> addToHead(1);
        return resultList;
    }
    else if (n % 2 == 0) {
        //return exp(x -> mult(x), n/2);
        IntSLList *resultList, *temp;
        temp = exp(x, n/2);
        resultList = temp -> mult(temp);
        delete temp;
        return resultList;
    }
    else {
        //return x -> mult(exp(x -> mult(x), (n-1) / 2));
        IntSLList *resultList, *temp, *temp2;
        temp = exp(x, (n-1) / 2);
        temp2 = temp -> mult(temp);
        resultList = temp2 -> smult(x -> head -> info);
        delete temp, temp2;
        return resultList;
    }
}
```

Recursive implementation of 12345^12345:
Wall Time = 4.68424
CPU Time = 4.61

Iterative implementation of 12345^12345:
Wall Time = 5.24798
CPU Time = 5.05

6)

```
float f(int n) {
    if (n == 1) return 1.0;
    else if (n % 2 == 0) return 1.0 / n + f(n-1);
    else return -1.0 / n + f(n-1);
}
```