Answer for SET 3

(1)

Algorithm: numDigits(n)

Requires: a positive integer n

Return: the number of its digits

```
1. if n < 10 then
2.     return 1
3. else
4.     return 1 + numDigits(n / 10)
5. endif
```

(2)

Algorithm: mySqrt(n)

Requires: a positive integer n

Return: an integer

```
1. return mySqrtHelper(1,n)
```

Algorithm: mySqrtHelper(m,n)

Requires: 2 positive integer m and n

Return: an integer

```
1. if (m*m <= n) && ((m + 1)*(m + 1) > n) then
2.     return m
3. else
4.     return mySqrtHelper(m + 1,n)
5. endif
```

(3)

Algorithm: isPrime(p)

Requires: a positive integer p

Return: true or false

```
1. return isPrimeHelper(mySqrt(p),p)
```

Algorithm: isPrimeHelper(m,p)

Requires: 2 positive integer m and p

Return: true or false

```
1. if m == 1 then
2.     return true
3. elseif p % m == 0 then
4.     return false
5. else
6.     return isPrimeHelper(m - 1,p)
7. endif
```

(4)

Algorithm: reverse(list)

Requires: a list

Return: a new list with the elements placed in reversed order

```
1. return reverseHelper(list,[])
```

Algorithm: reverseHelper(L1,L2)

Requires: 2 list

Return: a new list

```
1. if isEmpty(L1)
2.     return L2
3. else
4.     return revereHelper(tail(L1),cons(head(L1),L2))
5. endif
```

(5)

Algorithm: num2list(n)

Requires: a positive integer n

Return: a list that contains the digits of the input integer in the correct order

```
1. return num2listHelper(n,[])
```

Algorithm: num2listHelper(n,L)

Requires: a positive integer n and a list L

Return: a list that contains the digits of the input integer in

```
1. if n == 0 then
2.     return L
3. else
4.     return num2listHelper(n/10,cons(n%10,L))
5. endif
```

trace for 35181

```
num2list(35181)
   return num2listHelper(35181,[])
```

```
num2listHelper(35181,[])
   n != 0
   return num2listHelper(3518,[1])
num2listHelper(3518,[1])
   n != 0
   return num2listHelper(351,[8,1])
……
num2listHelper(3,[5,1,8,1])
   n != 0
   return num2listHelper(0,[3,5,1,8,1])
num2listHelper(0,[3,5,1,8,1])
   return [3,5,1,8,1]
```

(6)

Algorithm: fiboList(n)

Requires: a positive integer n

Return: a list whose elements are the first n Fibonacci numbers

```
1. return fiboListHelper(n,[])
```

Algorithm: fiboListHelper(n,list)

Requires: a positive integer n and a list

Return: a list whose elements are the first n Fibonacci numbers

```
1. if n == 0 then
2.     return list
3. else
4.     return fiboListHelper(n-1,cons(Fibo(n),list))
5. endif
```

Algorithm: Fibo(n)

Requires: a positive integer n

Return: the nth Fibonacci number

```
1. if n == 1 then
2.     return 1
3. elseif n == 2 then
4.     return 1
5. else
6.     return Fibo(n-1) + Fibo(n-2)
7. endif
```

trace for 5

```
fiboList(5)
   return fiboListHelper(5,[])
fiboListHelper(5,[])
   5 != 0
   return fiboListHelper(4,[3])
fiboListHelper(4,[3])
   4 != 0
   return fiboListHelper(3,[2,3])
……
fiboListHelper(1,[1,1,2,3])
   1 != 0
   return fiboListHelper(0,[0,1,1,2,3])
fiboListHelper(0,[0,1,1,2,3])
   return [0,1,1,2,3]
```

(7)

Algorithm: splitEO(list)

Requires: a list of integers

Return: a list of odd-positioned elements and another of even-positioned elements

```
1. return splitEOHelper(list,[],[])
```

Algorithm: splitEOHelper(list,list1,list2)

Requires: 3 lists

Return: a list of odd-positioned elements and another of even-positioned elements

```
1. if isEmpty(list) then
2.     return list1, list2
3. elseif isEmpty(tail(list)) then
4.     return concat(list1,cons(head(list),nil)),list2
5. else
6.     return
   splitEOHelper(tail(tail(list)),concat(list1,cons(head(list),
   nil)),concat(list2,cons(head(tail(list)),nil)))
7. endif
```

Algorithm: concat(L1,L2)

Requires: 2 lists L1 L2

Return: a new list with L1 attached to the head of L2

```
1. if isEmpty(L1) then
2.     return L2
3. else
```

```
4.     return cons(head(L1),concat(tail(L1),L2))
5. endif
```

trace for [2,5,6,8,7,3,4,0]

```
splitEO([2,5,6,8,7,3,4,0])
   return splitEOHelper([2,5,6,8,7,3,4,0],[],[])


splitEOHelper([2,5,6,8,7,3,4,0],[],[])
   isEmpty([2,5,6,8,7,3,4,0])??                NO!
   isEmpty([5,6,8,7,3,4,0])??                  NO!
   return splitEOHelper([6,8,7,3,4,0],[2],[5])
splitEOHelper([6,8,7,3,4,0],[2],[5])
   isEmpty([6,8,7,3,4,0])??                    NO!
   isEmpty([8,7,3,4,0])??                      NO!
   return splitEOHelper([7,3,4,0],[2,6],[5,8])
splitEOHelper([7,3,4,0],[2,6],[5,8])
   isEmpty([7,3,4,0])??                        NO!
   isEmpty([3,4,0])??                          NO!
   return splitEOHelper([4,0],[2,6,7],[5,8,3])
splitEOHelper([4,0],[2,6,7],[5,8,3])
   isEmpty([4,0])??                            NO!
   isEmpty([0])??                              NO!
   return splitEOHelper([],[2,6,7,4],[5,8,3,0])
splitEOHelper([],[2,6,7,4],[5,8,3,0])
   isEmpty([])??                               YES!
   return [2,6,7,4],[5,8,3,0]
```

Algorithm: splitEO(list)

Requires: a list of integers

Return: a list of odd-positioned elements and another of even-positioned elements

```
1. if isEmpty(list) then
2.     return [],[]
3. elseif isEmpty(tail(list)) then
4.     return list,[]
5. else
6.     let(L1,L2) =
```