# Programming Assignment 1

## Pseudo Code

### Multiple Recursion

Function MultipleOddo(int n)

If ( n is **0** or **1** or **2)**

Return 1

Else

Return MultipleOddo(n-1) + MultipleOddo(n-2) + MultipleOddo(n-3)

## Linear Recursion

Function LinearOddo(int n)

Return LinearOddoHelper(n,1,1,1)

Function LinearOddoHelper(int n, int a, int b, int c)

If n is 0

Return a

Return LinearOddoHelper(n-1, b, c , a+b+c)

## B)

The Multiple recursion version of the Oddonacci sequence is exponential because, for each value in the sequence, it must trigger 3 recursive calls to calculate the preceding oddonacci values. For each of these recursive calls, it triggers 3 more recursive calls, leading to a rapidly growing call stack. This results in a treelike structure where each call spawns 3 new branches, resulting in an exponential increase in calls. This growth will rapidly increase time complexity and consume a significant amount of memory due to the deep call stack.

On the other hand, the Linear recursion variant is much more efficient because it avoids recalculating the same values. It calculates each oddonacci value once and carries the result forward to the next calculation. This eliminates the need for recomputation, reducing time and space complexity.

C)

Yes, a tail recursive algorithm can be created for the oddonacci sequence. For the algorithm to be tail recursive, the recursive call must be the last line executed in the function. We can do this by passing the last 3 oddonacci values as parameters to the recursive function. Instead of relying on recomputed values from recursive calls to perform further computation, we update these parameters at each step and carry forward until we reach base case.