a) Multiple Recursion:

```
Algorithm multipleOdd(n):
   Input: A nonnegative integer n
   Output: The nth Oddonacci number $F_n$
      if ($n \le 3$) then
         return 1
      else
         return multipleOdd(n-1) + multipleOdd(n-2) + multipleOdd(n-3)
```

Linear Recursion:

```
Algorithm linearOdd(n):
   Input: A nonnegative integer n
   Output: Pair of Oddonacci numbers ($F_n$, $F_{n-1}$)
       if ($n \le 3$) then
          return (1,1,1)
       else
          (i, j, k) = linearOdd(n-1)
          return (i + j + k, i, j)
```

Observations: The multiple-recursion Oddonacci version takes significantly more time to compute than the linear-recursion version since the time complexity of the multiple-recursion ($O(3^n)$) is larger than the time complexity of the linear-recursive ($O(n)$).

b) The multiple-recursion Oddonacci method is of exponential time complexity since the number of recursive calls triples every other time. The linear-recursion Oddonacci method resolves specific bottlenecks of the multiple-recursion by reducing the number of recursive calls. It is of linear time complexity since the method makes n-3 recursive calls.

c) None of the two methods use tail recursion. For the multiple-recursion Oddonacci method, the last action of the function is an addition. For the linear-recursion Oddonacci method, the last action of the function returns an array. A tail-recursive version of Oddonacci can be implemented by adding three additional parameters representing three Oddonacci numbers and shifting them to the next Oddonacci numbers in the sequence every time the method calls itself.

```
Algorithm tailOdd(n, i, j, k):
  Input: A nonnegative integer n and the last three computed Oddonacci numbers i, j and k
  Output: The nth Oddonacci number $F_n$
     if (n = 1) then
        return i
     else if (n = 2) then
        return j
     else if (n = 3) then
        return k
     else
        return tailOdd(n-1, j, k, i + j + k)
```