A)

Linear Oddonacci

```
Algorithm LinearOddonacci (n, one, two, three)
Input: A positive integer n, initial values one, two, three
Output: The nth oddonacci number
If n =1:
    Return one
If n =2:
    Return two
If n =3:
    Return three
Else:
 Return LinearOddonacci(n-1, two , three ,one + two + three)
```

Multiple Oddonacci

```
Algorithm MultipleOddonacci (n)
Input: A positive integer n
Output: The nth oddonacci number
If n ≤ 2:
   Return 1
Else:
   Return MultipleOddonacci (n-1) + MultipleOddonacci (n-2) + MultipleOddonacci (n-3)
```

B)

The linear recursion has a linear time complexity of time O (n) since it calculates the oddonacci numbers by taking the sum of three preceeding numbers which helps avoid repetition and redundant calculations. This results in a linear growth in runtime even when the number of inputs increases. However, the multiple recursion has a time complexity of $O(n^3)$ since it calculates the same oddonacci numbers multiple times which leads to redundant calculations. Due to this bottleneck, the time complexity becomes exponential, and the runtime becomes very long.

C)

Yes, my linear algorithm uses tail recursion because the recursive call is the last operation, however, the multiple recursion algorithm does not use tail recursion since the function needs to perform additional operations after the recursive call is returned.

A tail-recursive version of the Oddonacci calculator can be designed using a helper function that hold the state of the last three numbers in sequence as well as the current position. When the input is less than three, it will return the value, otherwise it will call the helper function recursively. When it does, it will move the parameters so it can carry on the next sequence while ensuring that the recursive call was the last operation done.