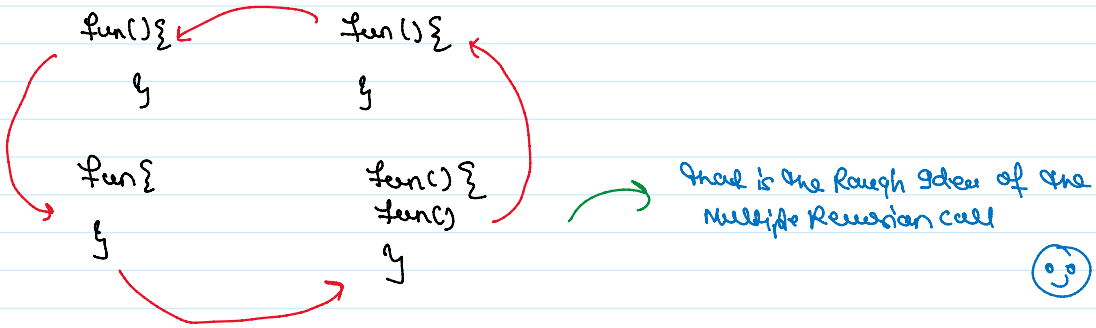
L5 Multiple Recursion Calls

Sunday, December 11, 2022 9:09 PM

Multiple Recursion calls that means more than one Recursion call at a single time that know as Multiple Recursion call at a time

→ This concept more Important for
→ Dynamic Programming
→ N Queen Solution etc



→ that is the Rough Idea of the Multiple Recursion call

Example Now to understand the Multiple Recursion is **Phebonacchi Numbers**

Now understand the concept of phebonacchi Numbers
→ the addition of previously two Numbers is known as phebonacchi Numbers

```
firstNumber = 0
secondNumber = 1
```

→ Always that constant

example 0 1 1 2 3 5 8 13 21

first two element constant

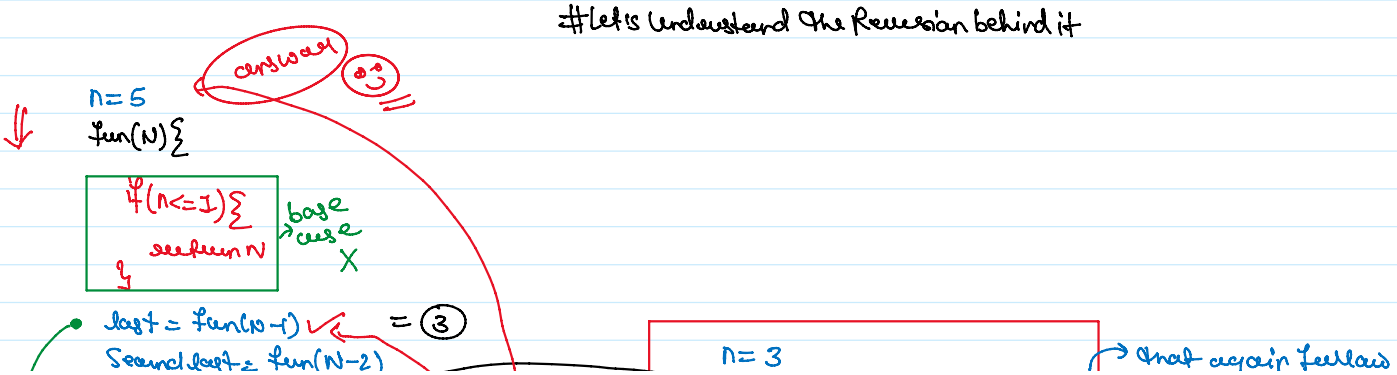
→ that is the concept of the phebonacchi Numbers

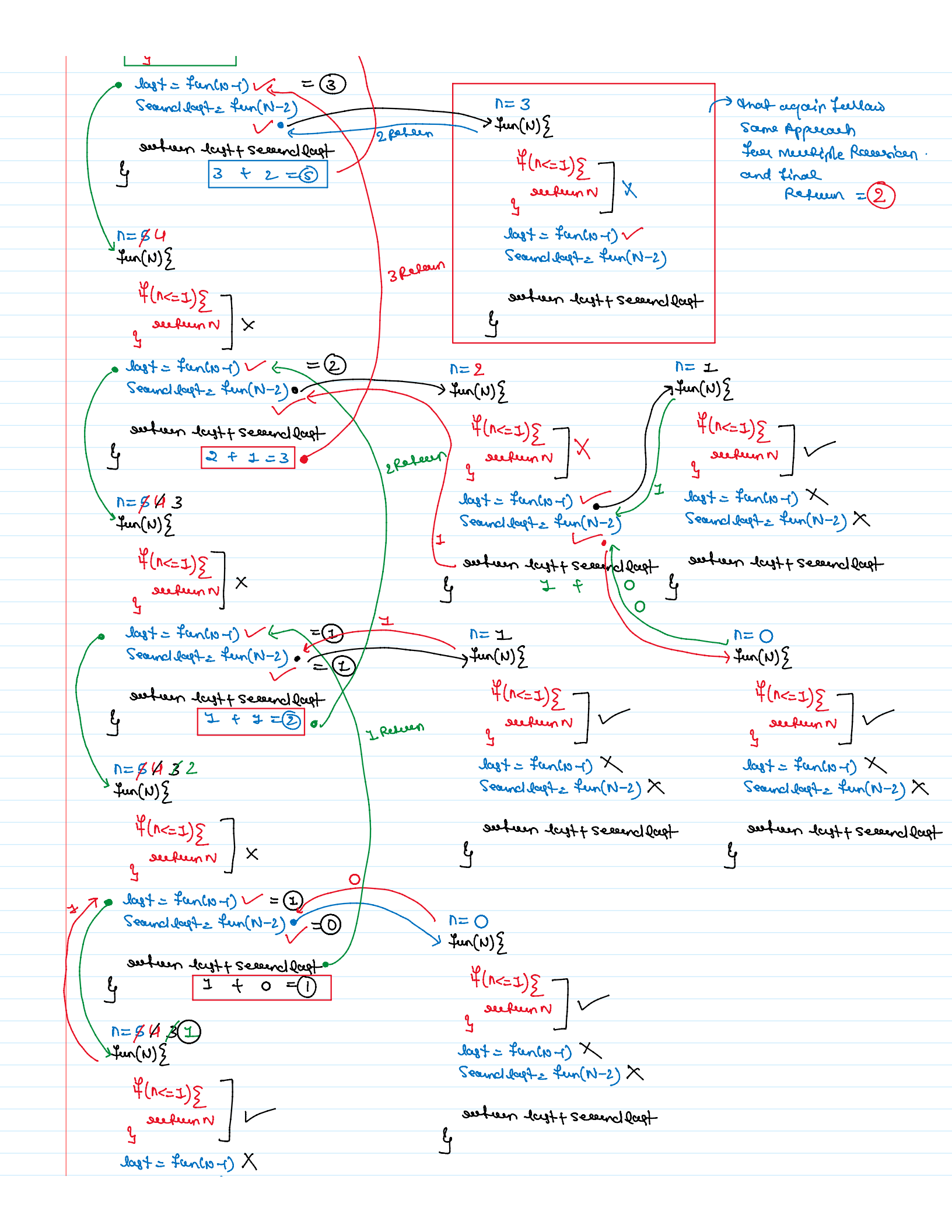
→ So what we observed that the current output depend on the last previously two Numbers

```
fun(N-1) + fun(N-2)
```

→ our Observation

#Let's understand the Recursion behind it




last = fun(N-1) = 3
SecondLast = fun(N-2)
return last + secondLast
3 + 2 = 5
n = 4
fun(N){
if(n<=1){
return N
}
3 Return
n = 3
fun(N){
if(n<=1){
return N
}
last = fun(N-1)
SecondLast = fun(N-2)
return last + secondLast
2 Return
That again follows
Same Approach
for Multiple Recursion.
and final
Return = 2
last = fun(N-1) = 2
SecondLast = fun(N-2)
return last + secondLast
2 + 1 = 3
n = 3
n = 2
fun(N){
if(n<=1){
return N
}
last = fun(N-1)
SecondLast = fun(N-2)
return last + secondLast
1 + 0
2 Return
n = 1
fun(N){
if(n<=1){
return N
}
last = fun(N-1)
SecondLast = fun(N-2)
return last + secondLast
1
last = fun(N-1) = 1
SecondLast = fun(N-2) = 1
return last + secondLast
1 + 1 = 2
n = 2
n = 1
fun(N){
if(n<=1){
return N
}
last = fun(N-1)
SecondLast = fun(N-2)
return last + secondLast
1 Return
n = 0
fun(N){
if(n<=1){
return N
}
last = fun(N-1)
SecondLast = fun(N-2)
return last + secondLast
last = fun(N-1) = 1
SecondLast = fun(N-2) = 0
return last + secondLast
1 + 0 = 1
n = 3
n = 0
fun(N){
if(n<=1){
return N
}
last = fun(N-1)
SecondLast = fun(N-2)
return last + secondLast

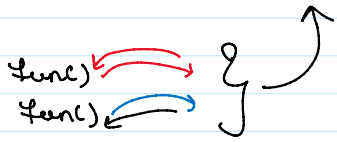
last = fun(n-1) X

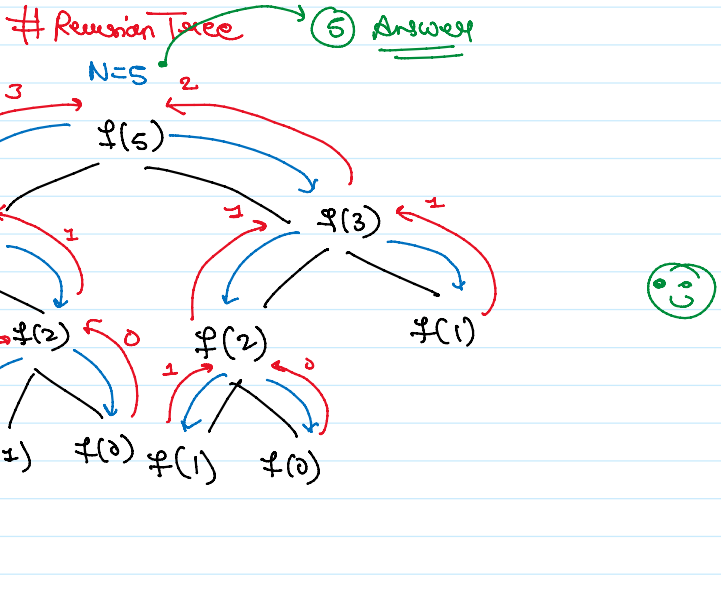
SecondLast = fun(N-2) X

return last + secondlast

Important Point

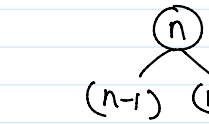
→ Always Remember that if multiple Recursion do then first complete and then second executed





Time Complexity = ?

→ Now we clearly seen in previously for every (n) we call (2) Recursion



that Means every calling 2^n ≈ Exponential

$= O(2^n)$

Space Complexity = O(N)

→ Recursion Stack Space

Implementation Python

```
class MultipleRecursionCalls:


    def recursivePhebo(self,n):
```

Implementation Java

```
public class L5_Multiple_Recursion_Calls {
    public static void main(String[] args) {

        int n = 10;
        int ans1 = rescursivePhebo(n);
```

```
    def recursivePhebo(self,n):

        if n <= 1:
            return n

        last = self.recursivePhebo(n-1)
        secondlast = self.recursivePhebo(n-2)

        return last + secondlast

    def iterativephebo(self,n):

        first = 0
        second = 1
        third = 0

        for i in range(n):
            first = second
            second = third
            third = first+second
        return third

ans = MultipleRecursionCalls()
print(ans.recursivePhebo(15))
print(ans.iterativephebo(15))
```

```
    public static void main(String[] args) {

        int n = 10;
        int ans1 = rescursivePhebo(n);
        System.out.println(ans1);
        int ans2 = iterativePhebo(n);
        System.out.println(ans2);
    }

    private static int iterativePhebo(int n) {

        int first = 0;
        int second = 1;
        int third = 0;

        for(int i = 0;i < n; i++){
            first = second;
            second = third;
            third = first+second;
        }

        return third;
    }

    private static int rescursivePhebo(int n) {
        if(n <= 1){
            return n;
        }

        int last = rescursivePhebo(n-1);
        int secondlast = rescursivePhebo(n-2);

        return last + secondlast;
    }
}
```