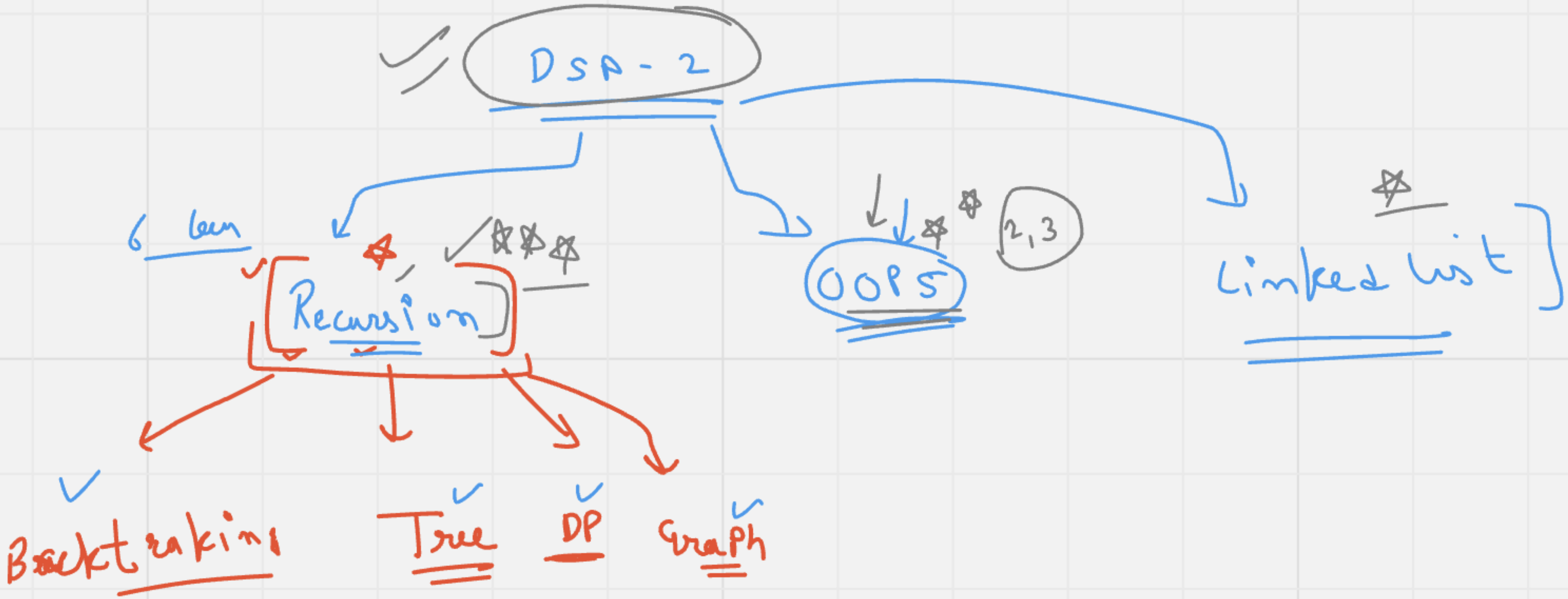
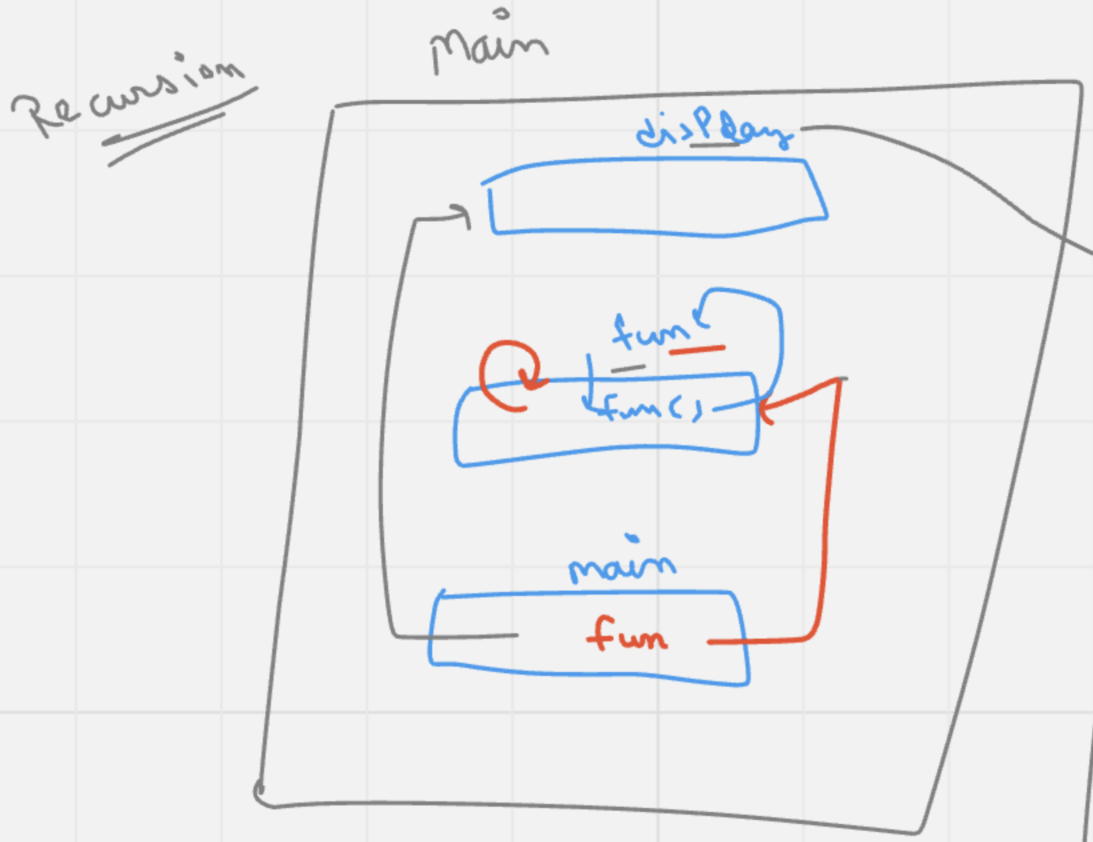
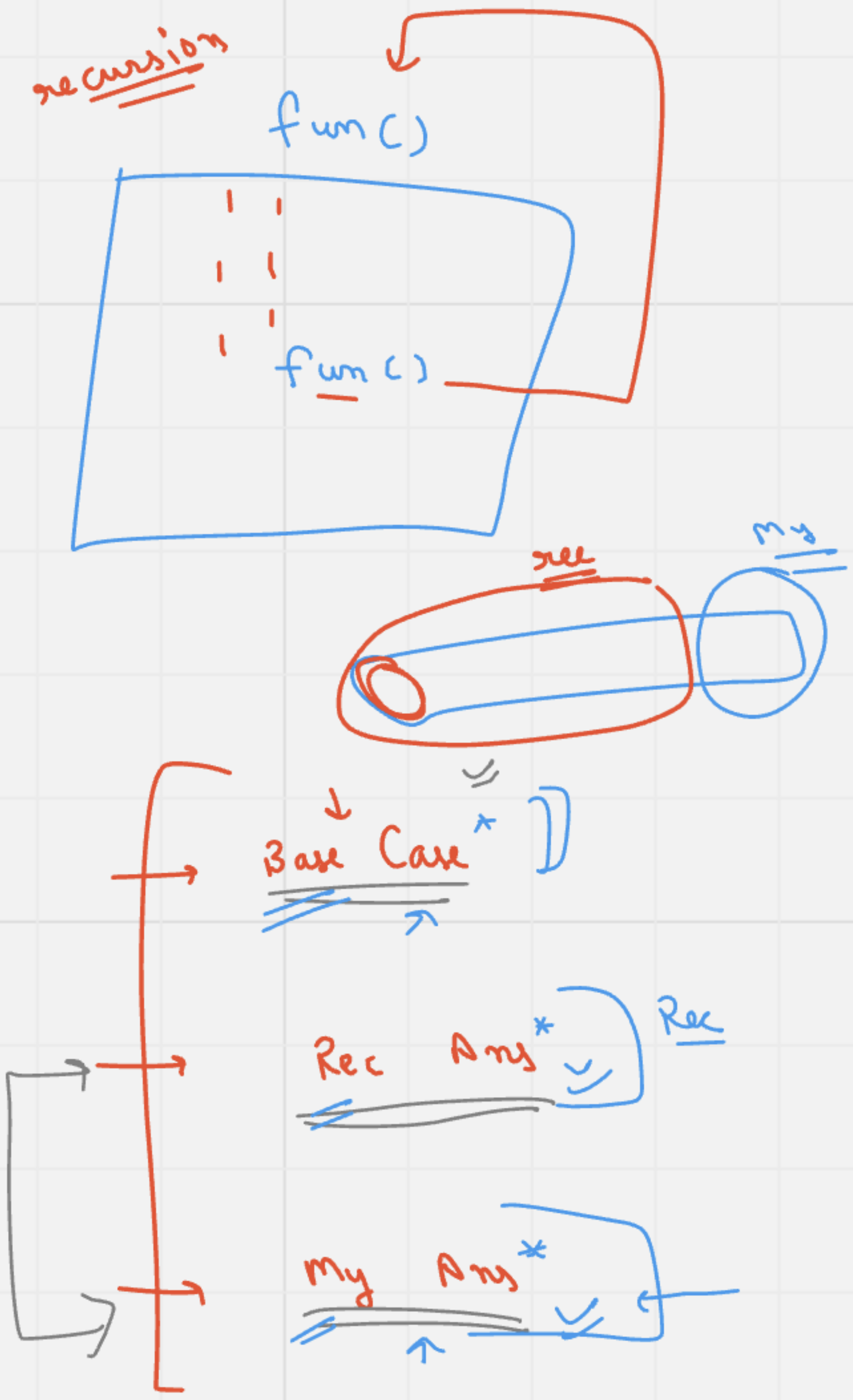
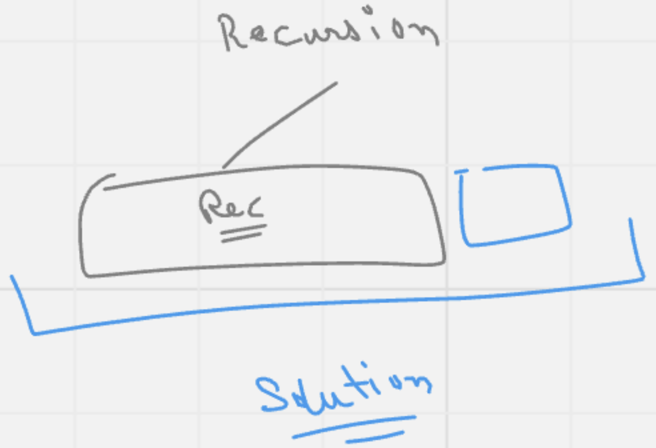

DSA - 2
6 lec
Recursion
OOPS
2,3
Linked list
Backtraking
Tree
DP
Graph

Recursion } → function calls itself again again



```
static void display(int a, int b)
{

    // body

}
```





$\Sigma 1 \equiv \boxed{1}$ Base Case

$\Sigma 5 = \underbrace{1 + 2 + 3 + 4}_{} + 5 \;]\; 15$

$\Sigma 5 = \overset{rec}{\underset{10}{\Sigma 4}} + \overset{My\,Ans}{5} = 15$

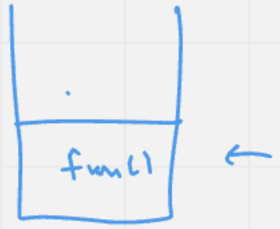
$\Pi_1 \rightarrow 1$ Base

$\Pi_5 = 1 * 2 * 3 * 4 * 5 = 120$

$\Pi_5 = \overset{Rec}{\underset{24}{\Pi_4}} * \overset{My\,Ans}{5} = 120$

call function

stored into a stack



```
static int fun(int n) {

    // base case
    if(n == 1)  {
      return 1;
    }

    // rec Ans
    int recAns = fun(n-1);

    // my Ans
    int myAns = recAns + n;
    return myAns;

}
```

rec

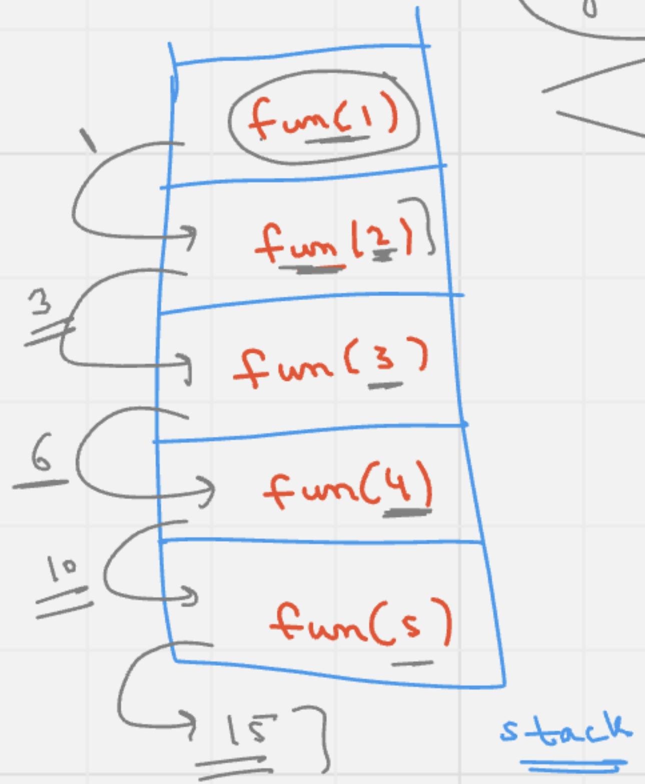
my

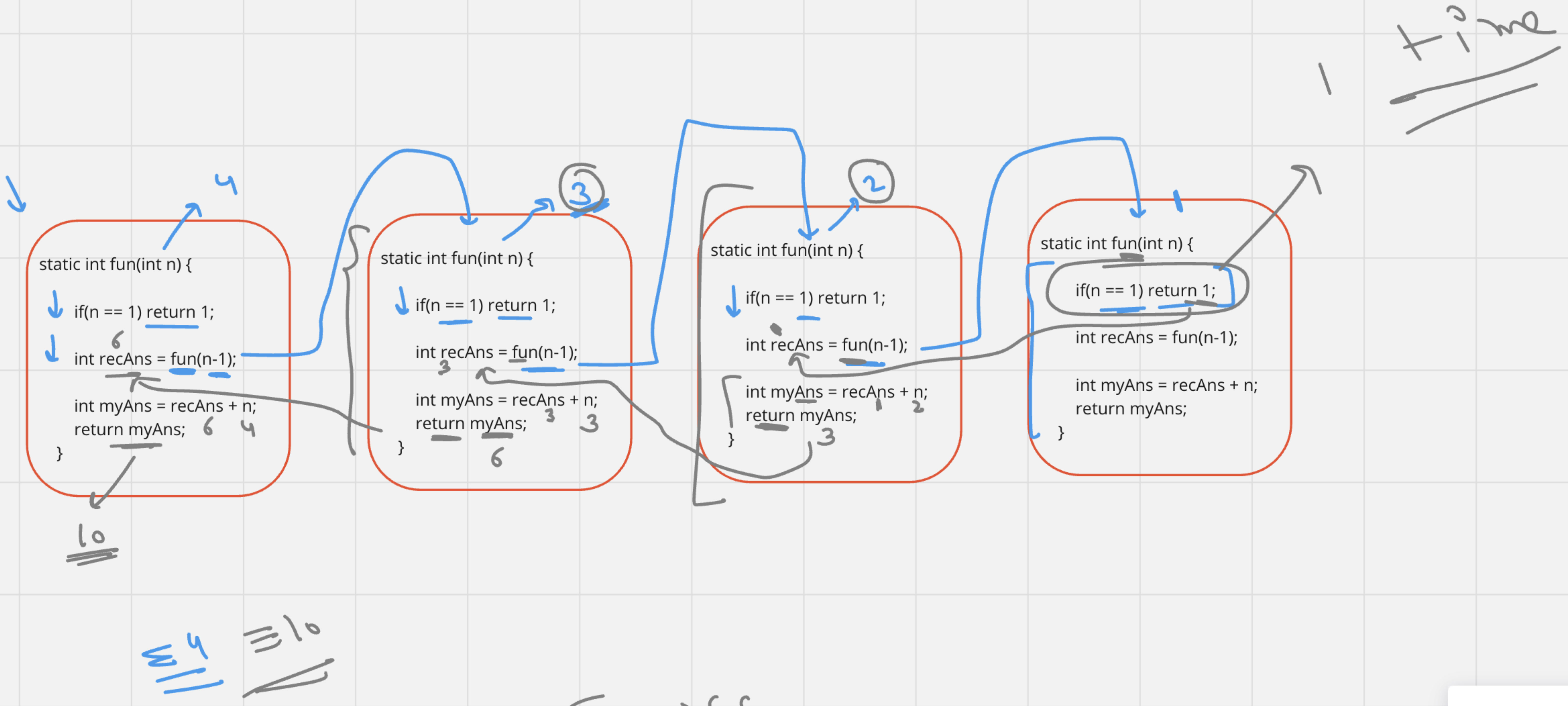
return i

get out from stack

return

function finish





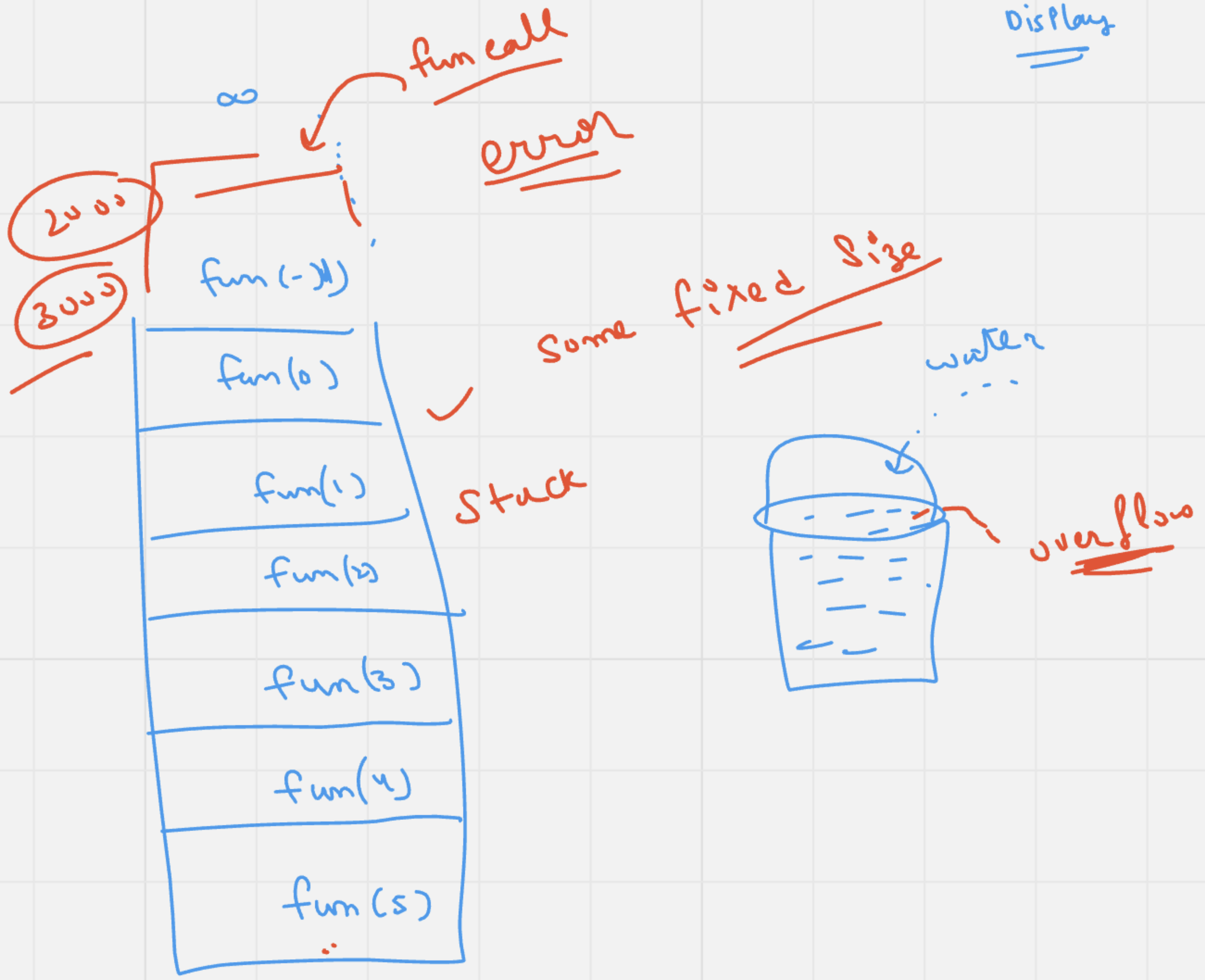
```
static int fun(int n) {
    if(n == 1) return 1;
    int recAns = fun(n-1);
    int myAns = recAns + n;
    return myAns;
}
```

```
static int fun(int n) {
    if(n == 1) return 1;
    int recAns = fun(n-1);
    int myAns = recAns + n;
    return myAns;
}
```

```
static int fun(int n) {
    if(n == 1) return 1;
    int recAns = fun(n-1);
    int myAns = recAns + n;
    return myAns;
}
```

```
static int fun(int n) {
    if(n == 1) return 1;
    int recAns = fun(n-1);
    int myAns = recAns + n;
    return myAns;
}
```

```
static int fun(int n) {

    // base case
    if(n == 1)  {

    }

    // rec Ans
    int recAns = fun(n-1);

    // my Ans
    int myAns = recAns + n;
    return myAns;

}
```



N

5

5
4
3
2
1

using

Prod

N → 5

1
2
3
4
5

void

(n==1)

Base 1 → return;

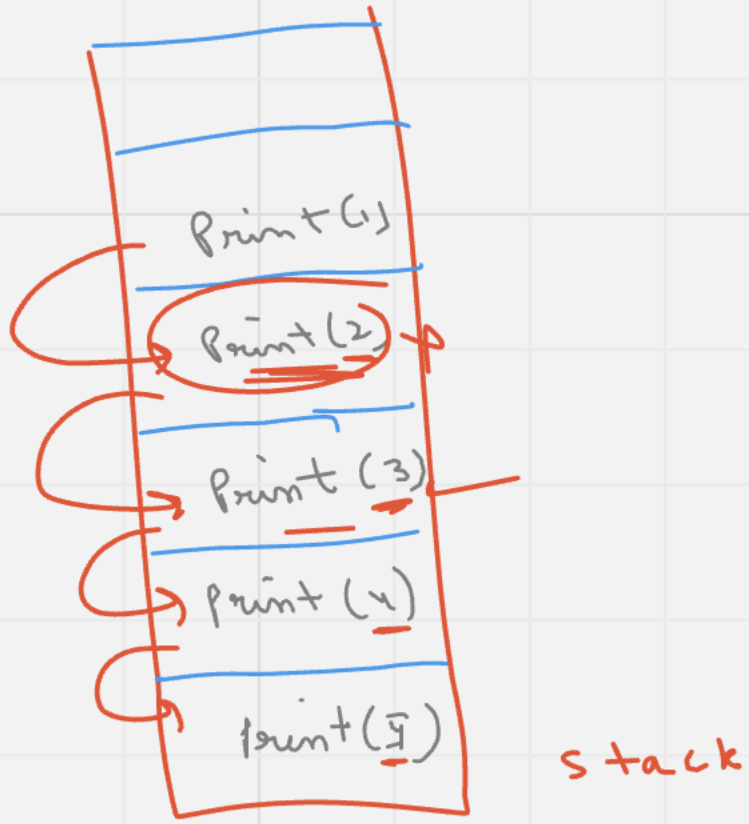
n-1

rec fun(4)

1
2
3
4
5

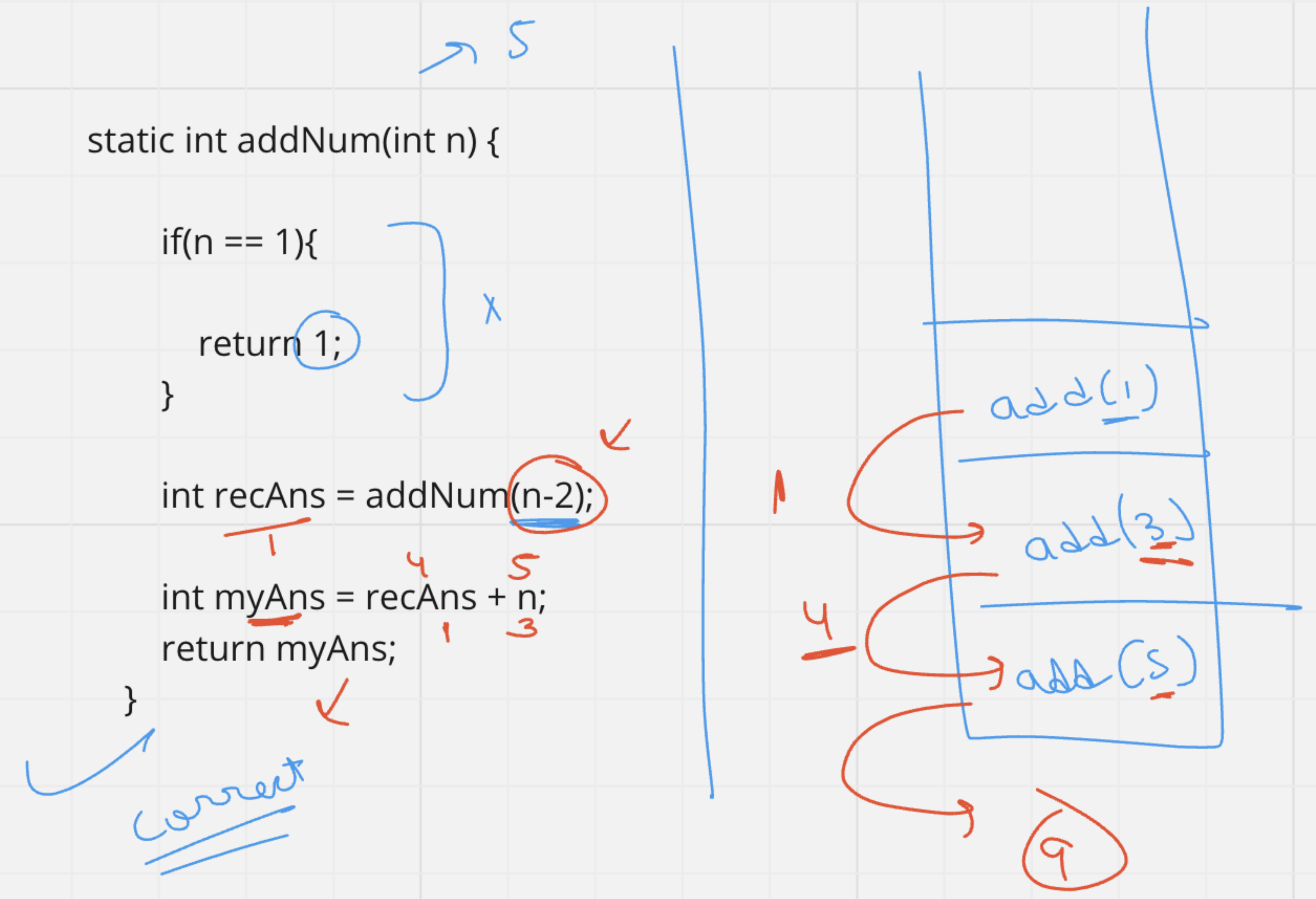
my System.out.Println(n);

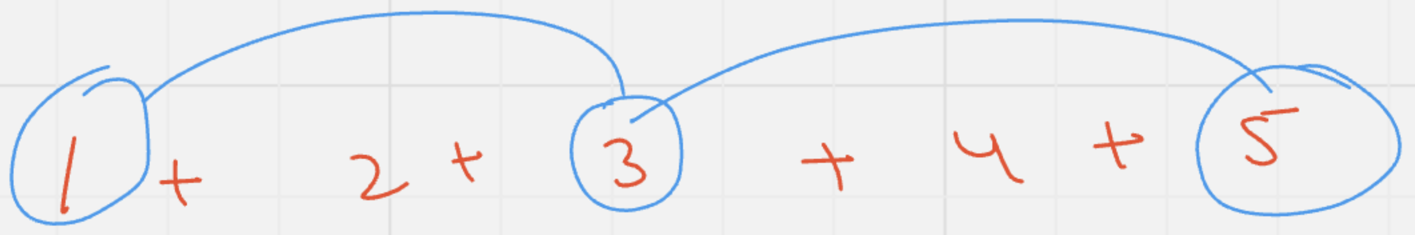
```java
static void print1ToN(int n) {

    if(n == 1) {
        System.out.println(1);
        return ;
    }

    print1ToN(n - 1);

    System.out.println(n);
}
```



```
static int addNum(int n) {

    if(n == 1){

        return 1;
    }

    int recAns = addNum(n-2);

    int myAns = recAns + n;
    return myAns;
}
```

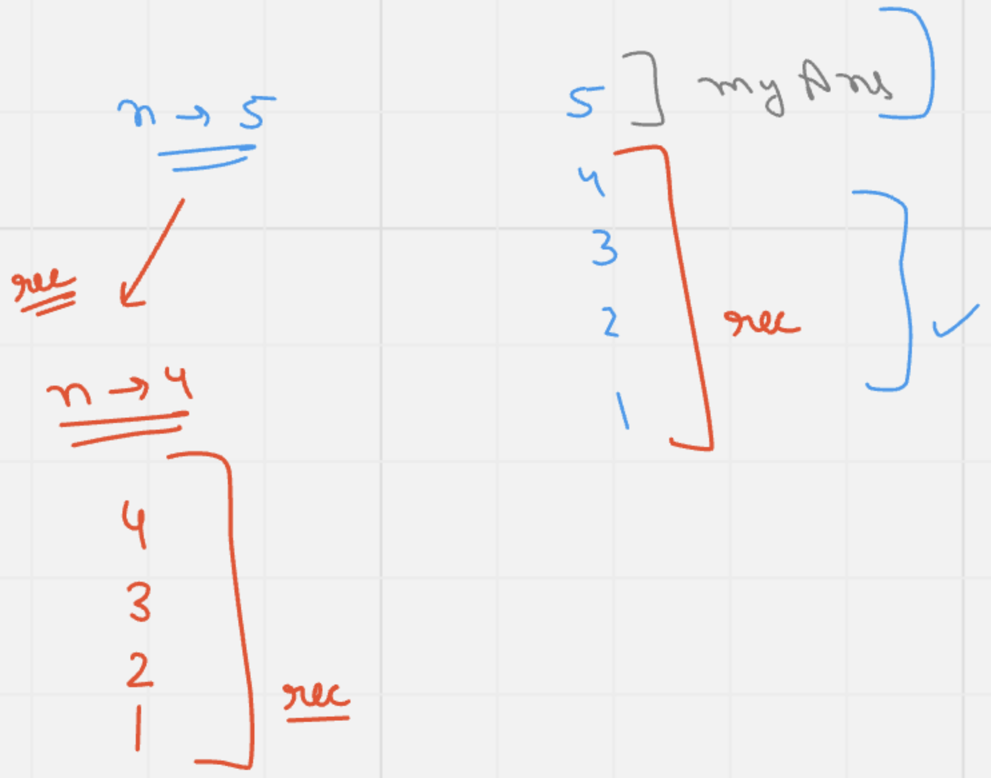
correct

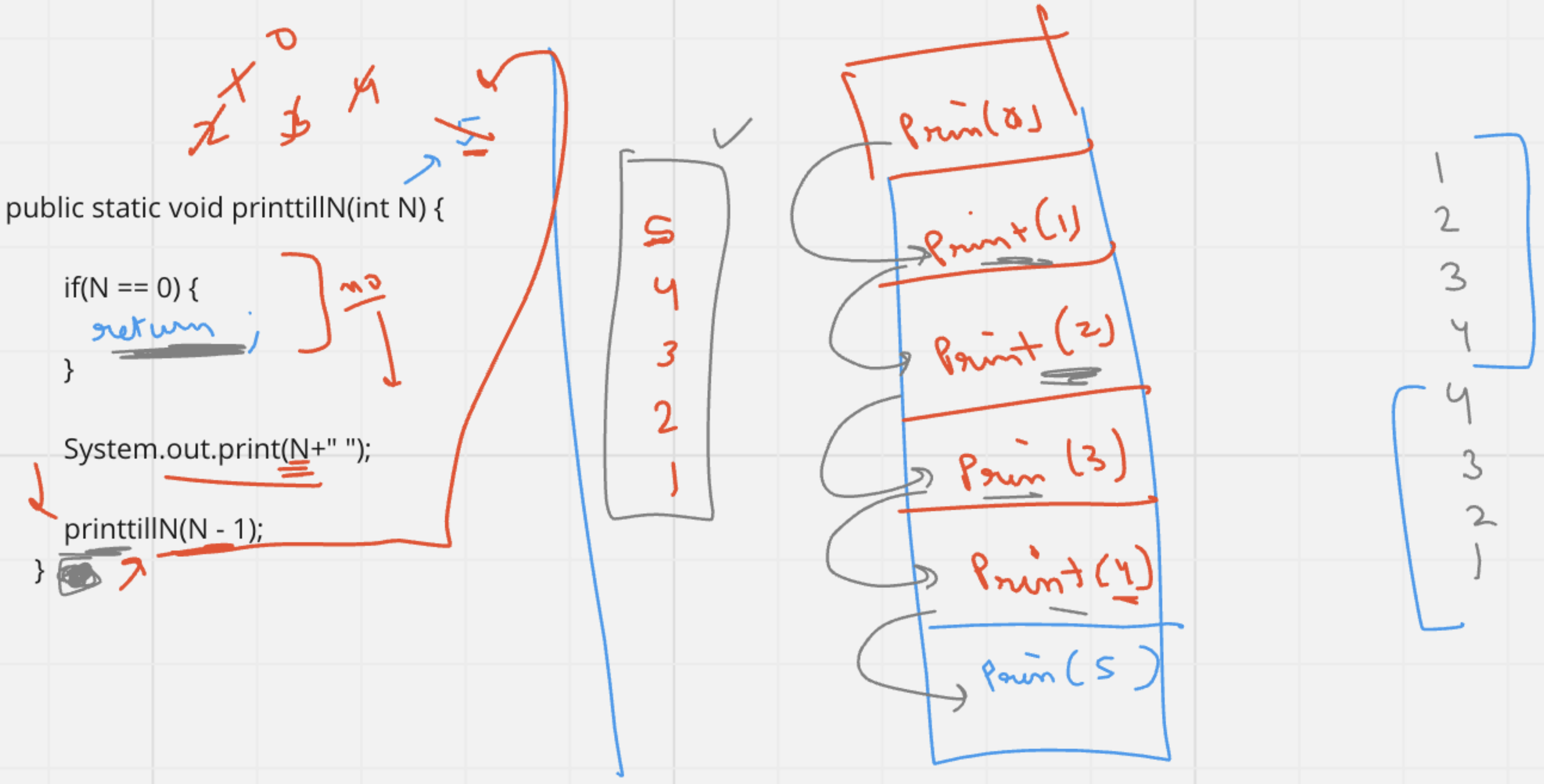




9

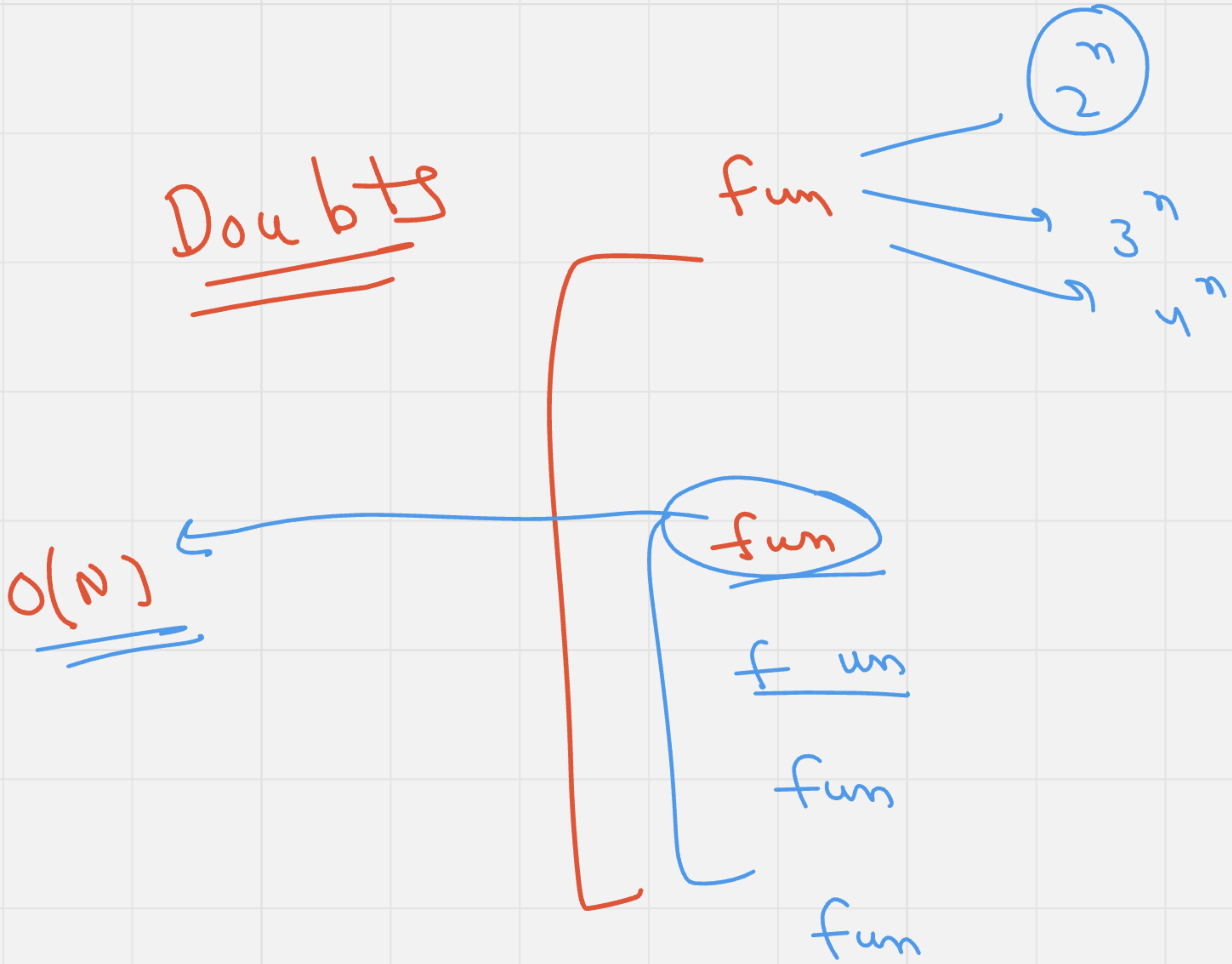
[find Sum of odd num 1 to n





```
public static void printtillN(int N) {

    if(N == 0) {
      return;
    }

    System.out.print(N+" ");

    printtillN(N - 1);
}
```

Doubts


fun
2^n
3^n
4^n
fun
fun
fun
fun
O(N)