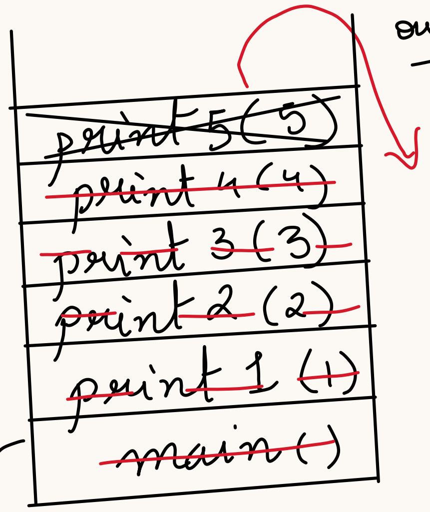
Requisites → functions, stack memory, heap memory

Q) Write a function that prints a message

→ A function is calling another function

→ main is called in the stack memory

How function calls works in languages.




output
1
2
3
4
5

VVIP:
While the function is not finished executing, it will remain in stack.

---

Point to be noted: when a function finishes executing it is removed from the stack and the flow of program is restored to where that function was called

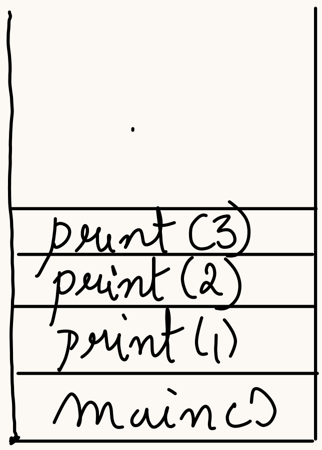
---

→ After printing all the numbers the print function will be removed from the stack.

* what is recursion? (A function that calls itself)

```
→ public static void main(String[] args){
    print(1);
  }
  static void print(int n){
      if (n == 5){
          System.out.println(5);
      return;
      }
      System.out.println(n);
      print(n+1);
  }
}
```

Recursive function for the same:




* Base condition in recursion:

condition where our recursion will stop making new calls

* If we are calling a function again and again, we can treat it as a separate call in the stack.

* No base condition → function calls will keep happening, stack will be filled again and again. We know that every call of function will take some memory.

* Memory of computer will exceed the limit. This will give us Stack Overflow error.

---

* Why Recursion ?

→ Function calling itself. It helps us in solving bigger and complex problems in a simple way.

→ You can convert recursion solutions into iteration and vice versa.

→ space complexity is not constant because of recursive calls

→ It helps in breaking down bigger problems into smaller problems

---

* Visualizing recursions: VVIP

main ( ) ↑ programmer

↓

print (1) // Recursion Tree

↓

print (2)

↓

print (3)

↓

print (4)

↓

print (5)

Q) Fibonacci numbers : Find $n^{th}$ fibonacci numbers

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| index → | 0th | 1st | 2nd | 3rd | 4th | 5th | 6th | 7th | 8th | 9th | . . |
| element → | 0, | 1, | 1, | 2, | 3, | 5, | 8, | 13, | 21, | 34, | . . |

$$\boxed{\text{Fibonaci}(N) = \text{Fibonaci}(N-1) + \text{Fibonaci}(N-2)}$$

→ Example: 3rd index

$2 \Rightarrow \underset{(N-1)}{1} + \underset{(N-2)}{1}$

$\Rightarrow 2$

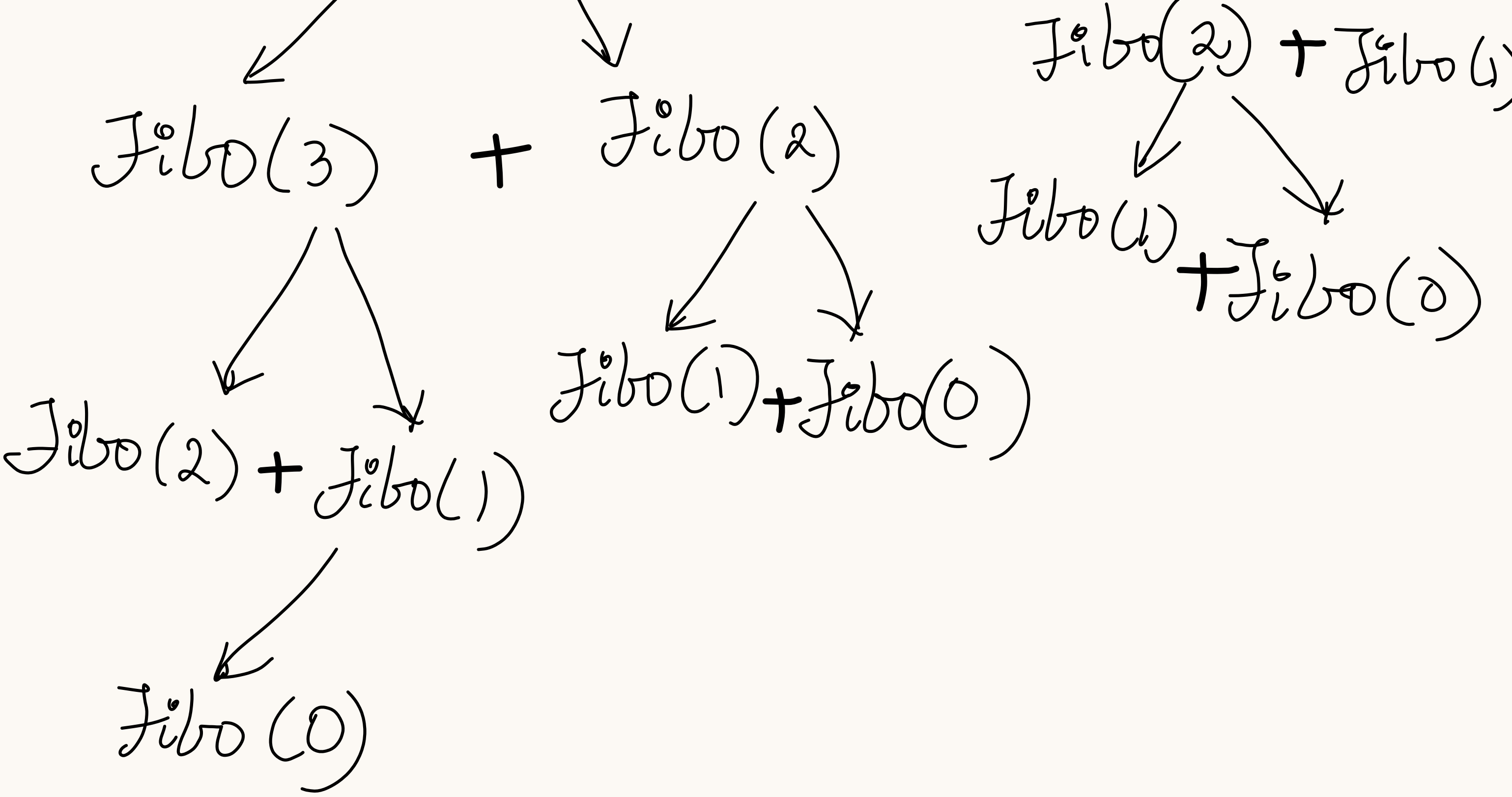
Similarly for $(N-1)$ Fibo it would be $(N-2) + (N-3)$ index elements

$$\boxed{\text{Fibo}(N-1) = \text{Fibo}(N-2) + \text{Fibo}(N-3)}$$

It will keep going on so if we try to build recursive tree for fibo(5)

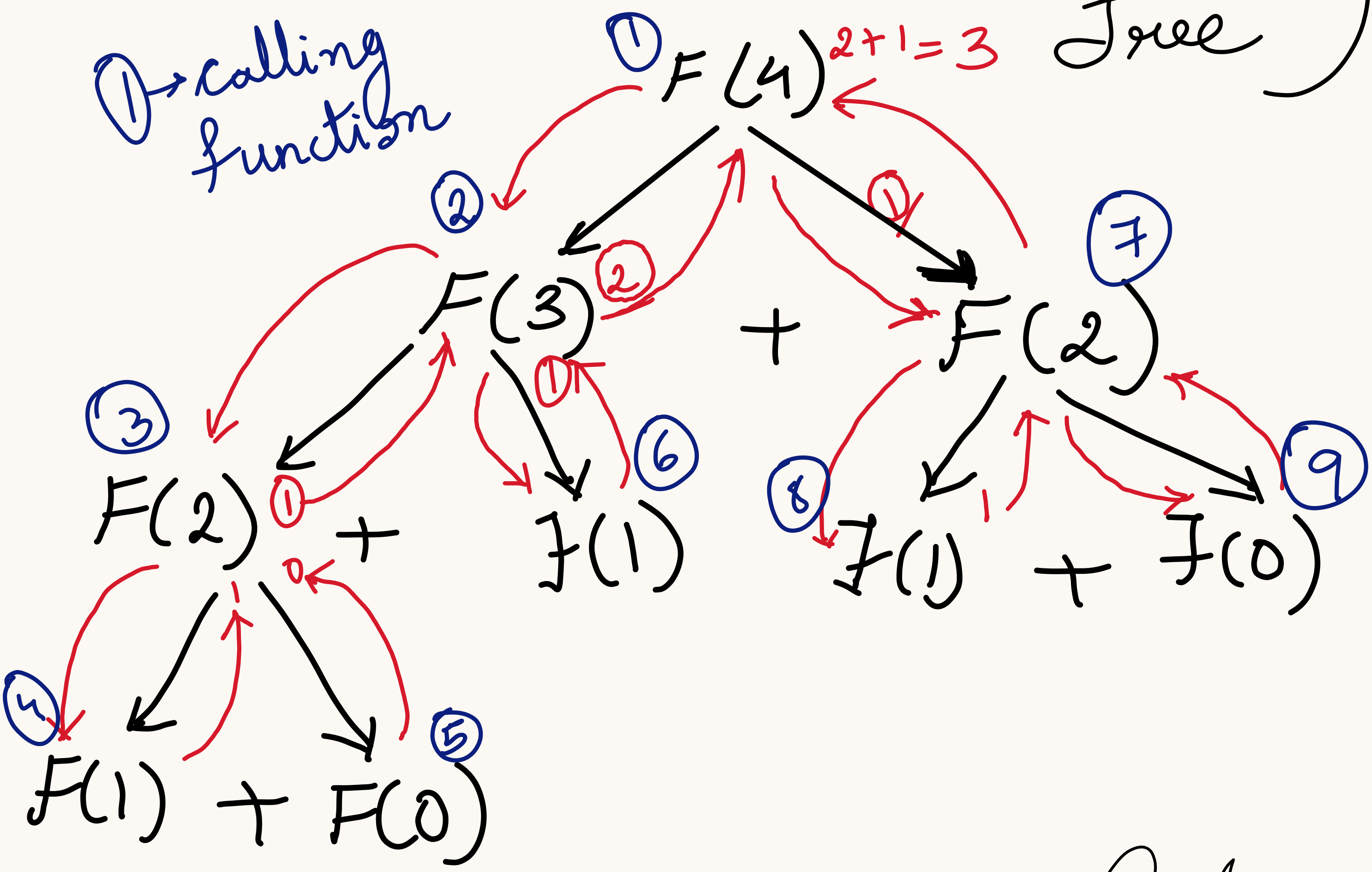
Fibo(5)

Fibo(4) + (Fibo(3))




## → Similarly for F(4)

F(4) would be (Recursive Tree)




How to apply recursions? Ans

* Break it down into smaller problems. The above formula is known recurrance relation.

* The base condition is represented by answers we already have. In the above case we know that $F(0) = 0$ and $F(1) = 1$. This is base condition.

---

Code:

```
Psvm (String[] args){
    int ans = fibo(4)
    System.out.println(ans);
}

static int fibo (int n){
    // base condition
    if (n < 2){
        return n;
    }
    return fibo(n-2) + fibo(n-1);
}
```

How to understand and approach a problem?

(VVIP)

1 → Identify if you can break down problem into smaller problems

2 → Write the recurrance relation if needed.

3 → Draw the recursive tree.

4 → About the tree

* See the flow of functions, how they are getting in stack
* Identify and focus on left tree calls and then right tree calls.
* Draw the tree and pointer again and again using pen and paper
* Use a debugger to see the flow of program.

5 → See How the values are returned at each step. See where the function will come out of. In the end it will come out of the main function.

---

* Variables : 1 Arguments (will go in the next funt call)

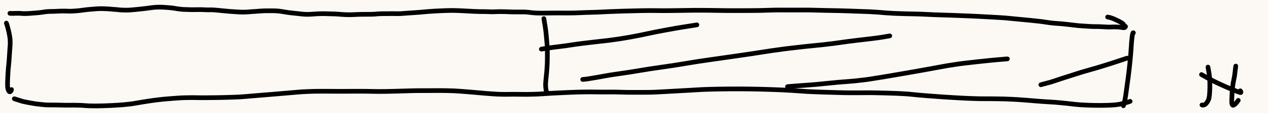
VVIP

2 Return Type

3 Body of the function

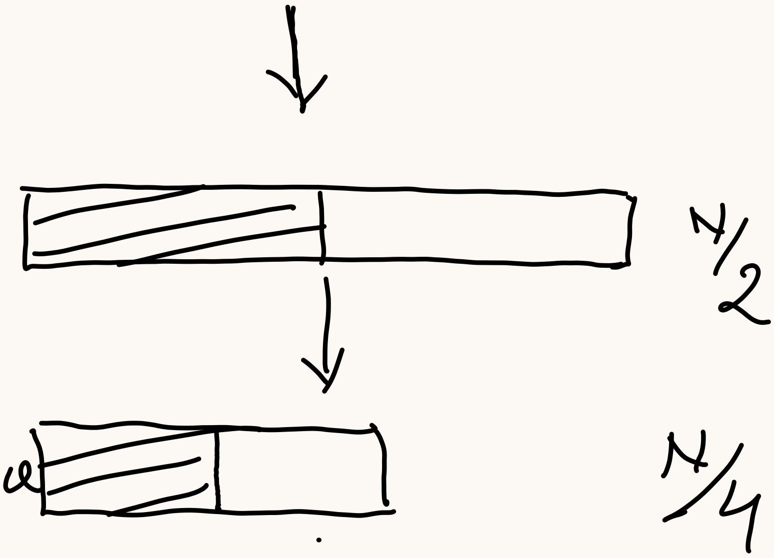
How to use variables in the functi ?

★ Let's take a question of Binary Search with recursions:



① comparing → O(1)

② Dividing it into 2 half.




③ Write the recurrance relation

$$\boxed{F(N) = \underbrace{O(1)}_{\text{comparison}} + \underbrace{F(N/2)}_{\text{Dividing Array in half}}} \rightarrow \text{Recurrance relation}$$

Types of recurrance relation :-

① Linear reccurrance relation → Fibo

② Divide and conquer recurrance relation. (B S) (Reduced by a factor) like ( F(N/2)) or (N/3) or (N/4)

★ If in the recursion calls two or more doing the & work for example

F(50) → $50^{th}$ fibonacci number in that F(2), F(1) and F(0) will be called again and again F(2) and this

$$
\begin{array}{ccc}
 & F(2) & \\
\swarrow & & \searrow \\
F(1) & & F(0)
\end{array}
$$

will create a repeatation in the function. And the ans will take much longer to compute.

How can we solve this issue? The ans is Dynamic programming (If in the recursion calls two or more recursion calls are doing the same work don't compute it again and again

Tip: Do not overthink

---

Make sure to return the result of a function call of the return type.

Code:-

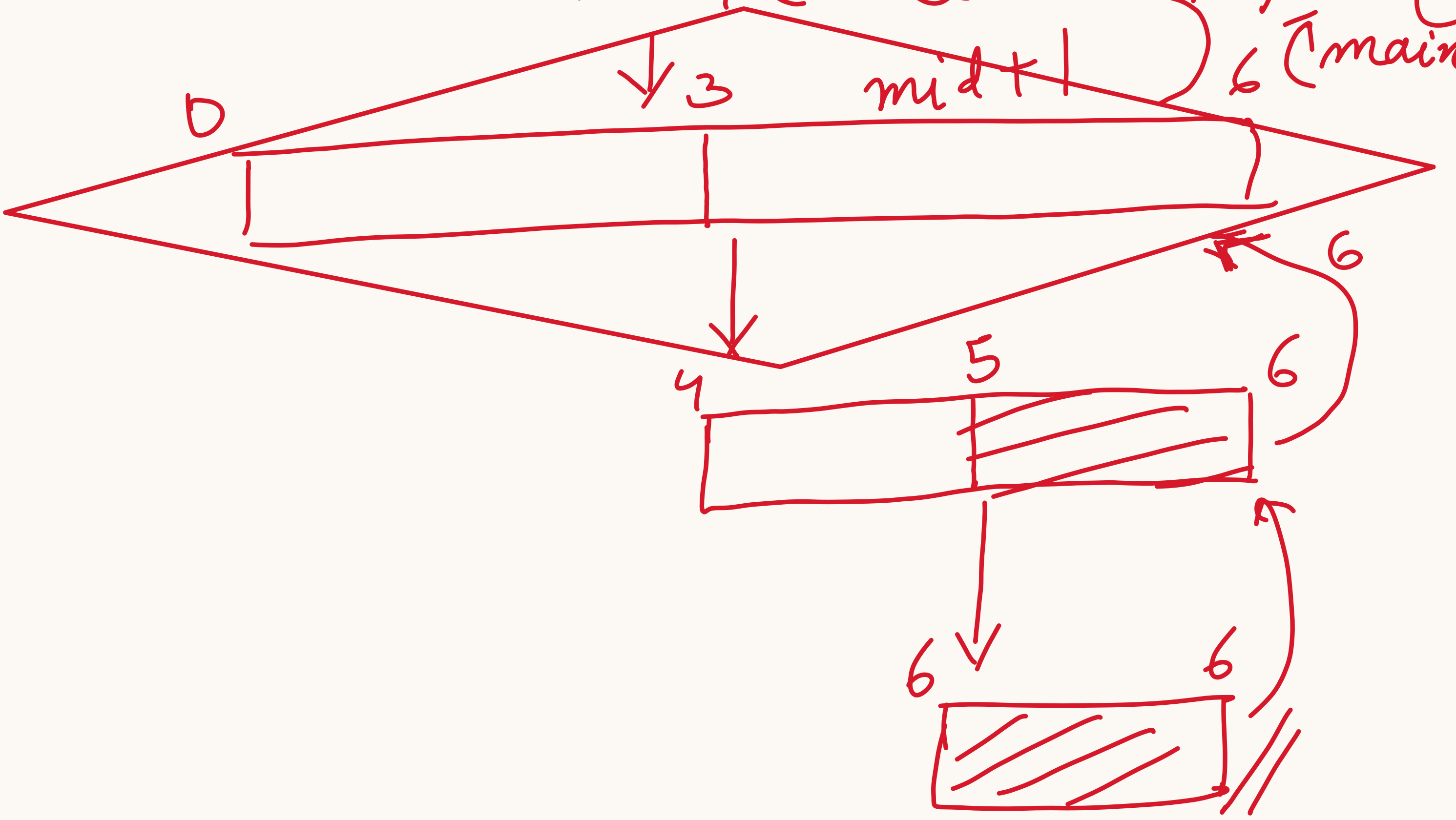
---

```
PSVM (String[], args){

int[] arr = {2,4,6,8,10,12};
```

```
    int target = 12;
    System.out.println(binarySearch(
            arr, target, 0, arr.length-1));
}

static int binarySearch(int[] arr,
int target, int start, int end){
    if (start > end){
        return -1;
    }
    int mid = start + (end-start)/2;
    if (arr[mid] == target){
        return mid;
    }
    if ( target < arr[mid]){
return binarySearch(arr, target, start,
                                mid-1);
    }
    return binarySearch(arr, target, mid+1,
                            end);
    }
}
```

All of these will be in stack :

main ( 6 // print(6)




return $6^{th}$ index