06/12/2021

SDS | Page No. | Date / /

⇒ ~~Recursive algorithms~~

⇒ Recursion

Consider an example below

```
public class NumbersExample {
    public static void main(String[] args) {
        print1(1);          // task is to print first five number
    }
    static void print1(int n) {
        System.out(n);
        print2(2);
    }
    static void print2(int n) {
        System.out(n);
        print3(3);
    }
    static void print3(int n) {
        System.out(n);
        print4(4);
    }
    static void print4(int n) {
        System.out(n);
        print5(5);
    }
    static void print5(int n) {
        System.out(n);
    }
}
```

Two point to be noticed here is

1) Function calling another function
2) All these function's have one thing in common, the body and defination of function is same.

O/P

1 2 3 4 5

⇒ How function calls works

First the main function is called, then print1(1) function was called by main function main function will remain in the stack till print1(1) has completed it's work.

But here print1(1) function calls print2(2) function so now print1(1) will also have to wait in call stack until print2(2) has completed its work, now print2 function calls print3(3) and this continues till print5(5) function.

print5(5)
print4(4)
print3(3)
print2(2)
print1(1)
main()

print5(5) function has to only print 5 so after printing it, it will return the flow of control to the function by whom print5 was called i.e print4

Now since print4 has completed its work the flow will be returned to print3 and this will continue till main function.

As the flow is returned back to main function the program will end and the call is empty now.

Imp point

1) While the function has not finished its execution it will remain in the stack
2) When a function finishes execution, it is removed from stack and the flow of program is restored to where the function was called.

⇒ Recursion

Function calling itself is called as recursive function and the mechanism is called as recursion.

Base condition is an important thing in recursion it is a condition where our recursion will stop making calls.

If the base condition is not written or not handled properly function call will keep happening and stack will be kept filling till a point when memory of computer exceeds the limit and it throws stack overflow error

Why Recursion?

1) It helps us in solving bigger / complex problem in a simpler way
2) You can convert recursion solution into iteration and vice-versa
3) Space complexity is not constant because of recursive calls
4) It helps us in breaking down bigger problems into smaller problems.

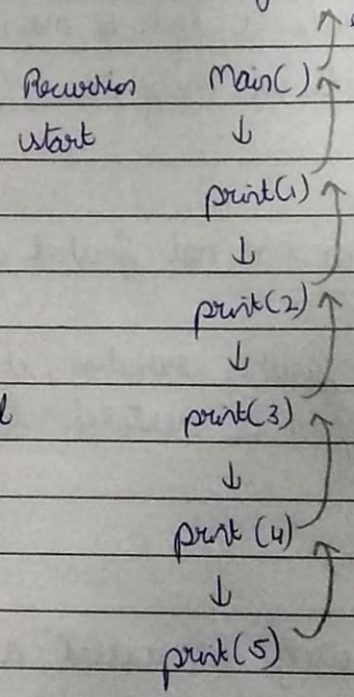
Writing the above iterative code in recursive manner

```
void print(int n){
    if(n==5){
        return;
    }
    System.out(n);
    print(n+1);
}
```

base cond^n

This above recursion is also called as tail recursion as the last statement of the function is a function call.

Visualising Recursion

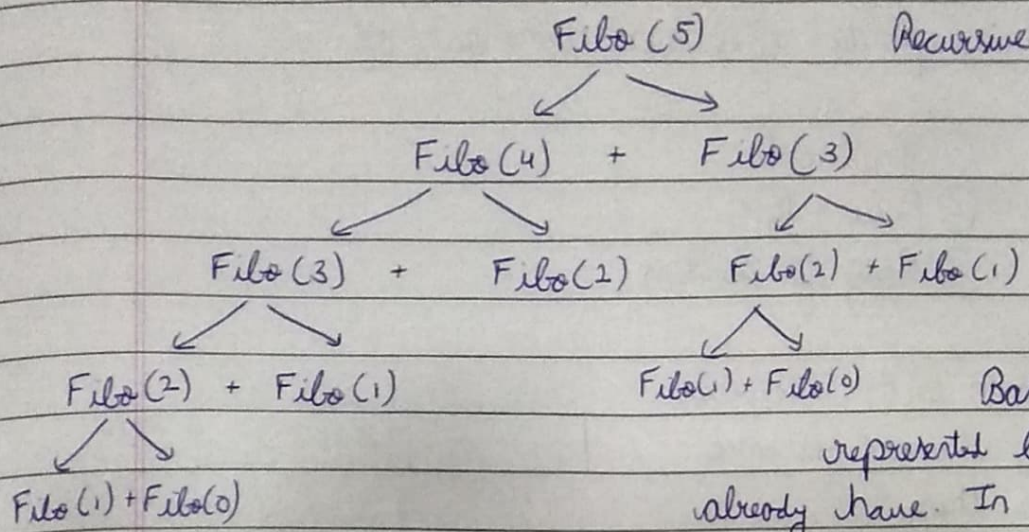


It is also known as recursion tree.

Q) Find n^{th} fibonacci number

0^{th}	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}	7^{th}	8^{th}	.....
0	1	1	2	3	5	8	13	.....	

Fibo(n) = Fibo(n-1) + Fibo(n-2) → Recurrence Relation



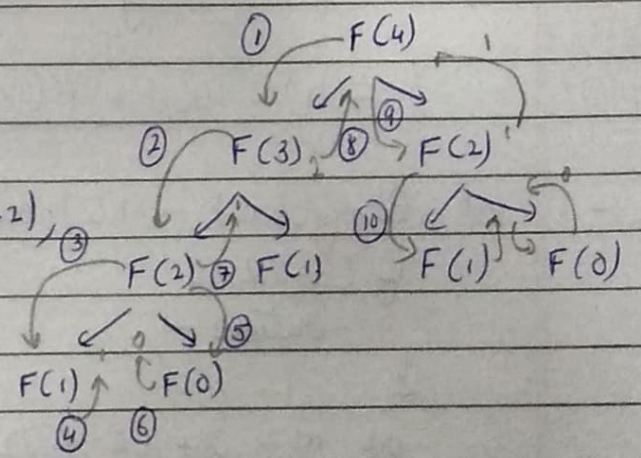
Recursive Tree

Base condition is represented by answers we already have. In this case, we know F(0) = 0 and F(1) = 1

This is the base condition

Code :-

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



⇒ Understanding and approaching a Recursion Problem

1) Identify if you can break your problem into smaller problems
2) Write the recurrence relation if needed
3) Draw the recursion tree
4) Abort the tree

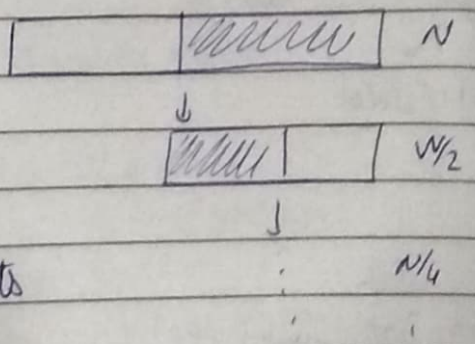
a) See the flow of functions, how they are getting in stack
b) Identify and focus on left tree calls and right tree calls
c) Draw the tree and pointers again and again using pen and paper
d) Use a debugger to see the flow
5) See how the values are returned at each step.
See where the function call will come out of.

Variables : ① Arguments
② Return type
③ Body of function

Q) Binary Search with recursion

In binary search we are doing two things

1) Comparing mid with target which takes O(1) time.
2) Dividing the array in two parts



$$F(n) = O(1) + F(n/2)$$ Recurrence rel^n

Code :-

```
int BinarySearch(int[] arr, int target, int s, int e){
    if (s>e){
        return -1;
    }
    int m = s+ (e-s)/2;
    if (arr[m] == target){
        return m;
    }
    if (target < arr[m]){
        return search(arr, target, s, m-1);
    }
    return search(arr, target, m+1, e);
}
```

e.g arr = [1, 2, 3, 4, 55, 66, 78] target = 78