Recursion :- The most important

Note :- Function/method & Memory management Knowledge is must

① Functions/Method :- A Function/method is a collection of code that you can use again and again

② Memory Management :- There are two types of memory, "Stack & heap"

* Stack memory :- When we declare a variable

eg :- int a = 10 ;
stack heap

reference variable are stored in stack

* Heap memory :- Variable which is pointing to the object of that variable are stored in heap are called heap memory.

1) Function calling another function
2) All of the function will have one thing in common the body and the defination of this function

eg :- Taking one parameter and doing something just name is different

* Internal working of (recursion code) Call stack + Debugging

```
//Code

public static void main (String[] args){

    print1(1);
}
static void print1 (int n) {
    System.out.println(n);
    print2(2);
}
static void print2(int n){
    System.out.println(n);
    print3(3);
    .
    .
    .
    .
    .
static void print5(int n){
    System.out.println(n);
}
```

function body change here because it is not calling anything

Flow of program:-

print5(5)
↳ print4(4)
↳ print3(3)
↳ print2(2)
↳ print1(1)
↳ main → in last program exits

it includes extra questions like

1) How function calls work in programming language?

steps

1) First it's calling main function
"all the function call that happen in a programming language they go into the stack memory."

2a) while the function is not finished executing it will remain in stack memory

Q Which is the first function that is called in programming language like JAVA, C, c++ etc?

ans Main function !!!

b) Main function is the first function that will go in stack and the last function that will came out of the stack.

c) when a function is staying inside the stack it basically means that function call is currently going on.

3) So the main function is called & then main function itself calls the printf function (given) So, printf function is called & main function is currently in progress, which say hey printf please give me the answer that I've asked you to do.
So, after printf function finished execution it will end, but it will rest in stack memory so, the printf will get called first

& then it will also go in stack memory & it is going to have a primitive (primitive are also stored in a stack memory) so, in the stack memory only there will be a variable primitive one because we have passed that. Now the print function is called so it'll print1(one)

3 What does print function do when it is called?
ans It prints whatever the values is passed.

4) Now, the print1 is going to call print2, so now the print1 will say to print2 that main actully wants us to print all the numbers from 1,2,3,4,5 I've printed 1, print2 could you, please print2 and asks the other functions to print the rest of the numbers? So, now print2 will take care for the rest of the numbers & it will print2 Now, print2 will be inside the stack

5) Now, print2 is like "hey" print3 I've not finished executing print1, main are waiting on me to print 2,3,4,5
As I've printed 2, print3 can you please make sure that no 3,4,5 are also get printed
So, while you do that I'll wait inside the stack memory.
So the print3 will be like fine you wait I'll take care of the no 3,4,5 Now print3 function is called it will print 3 & go

inside stack memory but before it will call print4 "Every function is in the stack memory is currently ongoing".

6) As print3 called print4 function it will perform same as above and then call another function and wait in stack

7) Now print4 will call print5, so 4 is printed and the function is obviously in the stack memory which is saying print5 we have printed 1, 2, 3, 4 and all of us are waiting in the stack could you please the last number? So, print5 will print last number and it'll also gonna say that "do you want me to call print6 or something else or am I the last one? print4 will be like you've the last one do not call anyone else. So print5 will be called & it will go in the stack memory Because any function that is currently running will go in stack memory. So print5 is in stack memory & it'll print 5. So all 1, 2, 3, 4, 5 has been printed

8 Now, print5 is going to be like My work is done and I don't need to call anyone else So my function call will be over.

Note:- When a function finishes executing it is removed from the stack and the flow of program is restored to where that function was called

eg int a = b + c ;

↓

it is calling the sum function & the sum function will return a value & it will return a value from where it was called.

9) So, print5 has finished executing so, from where will be our program executing right now from where it was called & the print5 will be removed from the stack. So now it will came outside & print5 is going to finish the executing its going to say to print4 hey I'm finished executing and I don't see that you're doing anything else. So you too can also finish executing.

10) Now, print4 is going to be like fine - print5 did it's work and nothing new needs to be done so, I'll also finish executing and I'll also leave the stack memory

11) Now, print4 will like print3 I'm done with my work and the rest is upto you. print4 will leave the stack. So Now the program flow is with print3

12) print3 will do the same thing and leave the stack print2 will also do the same & leave print1 will be like I'm also done & leave. So it will go from where it was called "main" function.

13) Now, main function will be like I'm also done & leave main is the last function that is removed from the stack & then the program will be over.

O/P 1, 2, 3, 4, 5

Recursion

Q Write a function that takes in a number and prints it (1st five no)

```
public class NumberExample{
    public static void main(String[] args){
        printnum(1);
    }

    static void printnum(int n){
        // base condition
        if(n==5){
            System.out.println(5);
            return;
        }
        // body :- System.out.println(n);
        printnum(n+1); (a tail recursion)
    }
}
```

it is a base condition

recursive call

this is the last statement that is executed in this recursive function
∴ It is a tail recursion.

Note :- 1) without check (a base condition) if the function is being called then it will run infinity i.e stackoverflow.

2) If you are calling a function again and again you can treat it as a separate call in the stack.

3) each call you can treat it as a separate function in the stack.

4) here, In this code the same function is being called again & again But, every function call is taking the memory seperately.

Notes :-

1) While a function call is not finished executing it will stay in the stack memory

Q What is Recursion?

Ans

Recursion means a function that calls itself

Q what is a base condition? (in recursion)

Ans

1) It is a condition where our recursion will stop making new calls
2) It is a simple if condition
3) It needs to be returned.

Q What is stack overflow? error?

Ans

1) Suppose there was no base condition we will get error.
2) It means that function calls will keep happening
3) Stack will be keep getting filled again and again.
4) we know that, every call takes a memory even though its the same function or different one doesn't matter
5) If you've calling a function more than one times simultaneously so, again and again every function call will take some memory in stack.
6) So, If there is no base condition it will keep going on and one time will come where memory exceeds of a computer's limit.
7) Hence, This is going to give or throw an error
8) That error is termed as stack overflow error

Q Why recursion?

Ans

1) It helps us in solving bigger/complex problems in a simple way.
2) You can convert recursion solutions into iteration and vice-versa.

Q What is iteration?

Ans 1) It basically means not using any function calls
2) It means loops & for loops.

3) Recursion takes a space as well
So, space complexity :- not constant because each function call is taking some memory
∴ recursive calls.

4) It helps us in breaking down the bigger problem into smaller problems

Since, there are recursion calls or function call linked to one another is visualizing recursion

* Visualization Recursion :- Imp

Q print a no from 1 to 5

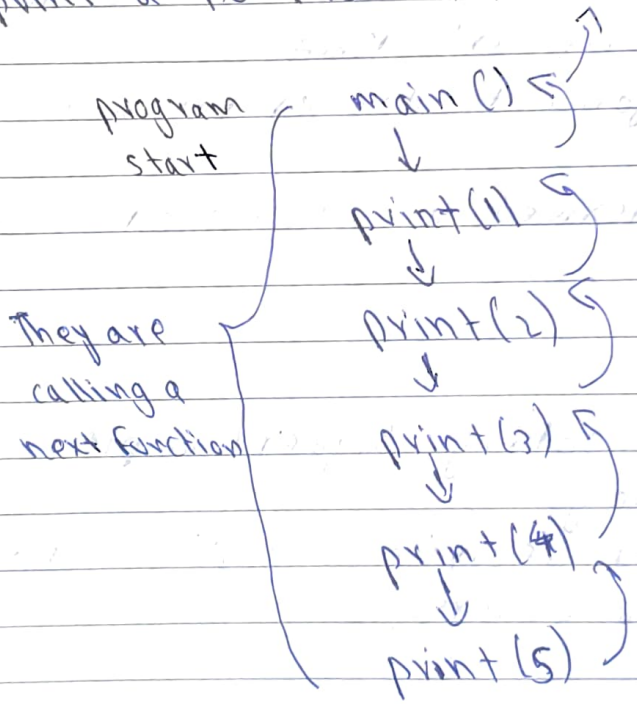


fig :- Recursion Tree / Recursive tree

Q. Find n^{th} fibonacci number using recursion.

→

Q. How to identify whether a problem can be solved in recursion or not?

Ans

① Practice

② Try to see if there's a smaller version of the problem that we can solve

"check if you can break the problem in to smaller problem"

Formula :- $fibo(N) = fibo(N-1) + fibo(N-2)$

in abb^{n} :- $fn = fn-1 + fn-2$ $(\because f_1 = f_2 = 1)$

Now let's see the working of fibonacci no. using formulae :-

Q. Find F_3?

Now as we know

$fn = fn-1 + fn-2$ - (formula)

$f_3 = f_3-1 + f_3-2$

$= f_2 + f_1$ } given

$= 1+1$

$= 2$

$\boxed{f_3 = 2}$

2 $f_5 = ?$

$f_n = f_{n-1} + f_{n-2}$ (formulae)

$f_5 = f_{5-1} + f_{5-2}$

$= f_4 + f_3$

$= 3 + 2$

$\boxed{f_5 = 5}$

So, the fibonacci series :-

f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	
0	1	1	2	3	4	5	6	...
0	1	1	2	3	5	8	13	...

So basically the entire problem is divided into two smaller problem

$f_n = f_{n-1} + f_{n-2}$

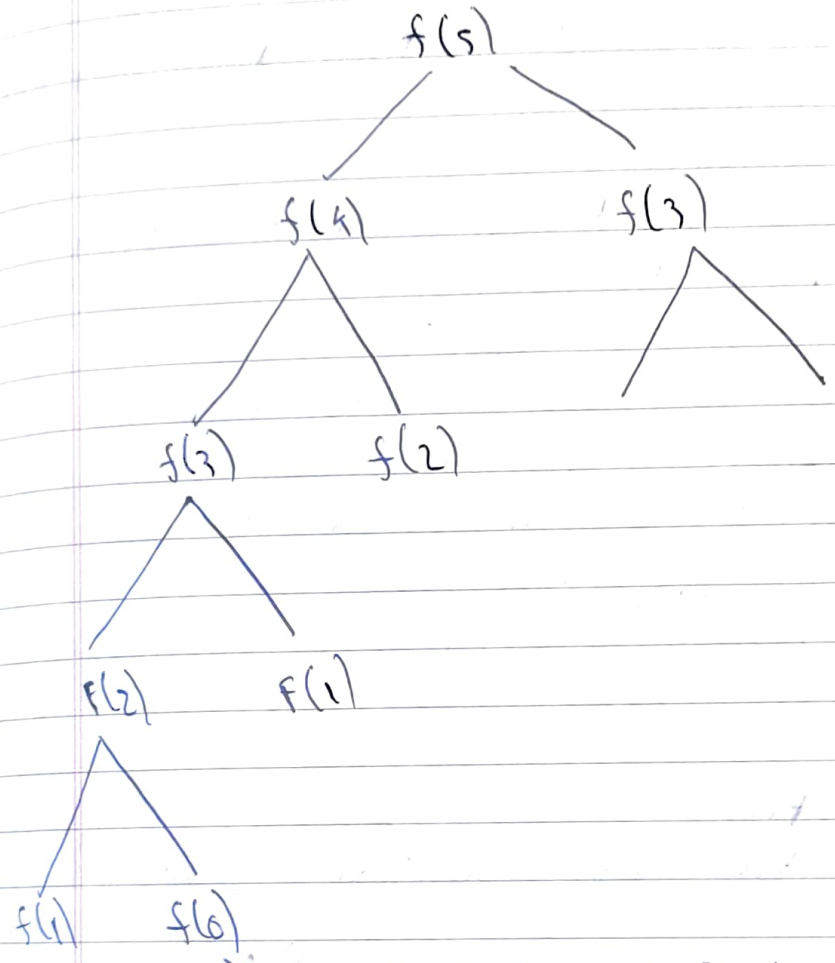
Here, Hence you can apply recursion

- Now the ~~dd~~ divided part itself can be broken down into two more smaller problem

$f_{n-1} = f_{n-2} + f_{n-3}$

Hence it will keep going on.....

9 Find the fibo no.



→ Now both the call finished executes hotwit will add the ans of these calls

fig:- Recursive tree for fibonacci

1) So, this is how you identify the solution can be solved via recursion or not. you try to see if you can break it down into a smaller problem

2) when you write the recursion in a formula it is known as recurrence relation.

3) The base condition is represented by answer we already have
eg :- In this case we know that f(0) = 0, f(1) = 1
this is base condition

Base conditions are those whose answers are already provided to you

Note :- Important steps for solving the recursion prob.

1) Try to break it down into a smaller problem then you can apply recursion
2) Try to figure out the recursive tree.

Ans

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

        System.out.println(f(7));
    }
    static int f(int n){
      // base condition
        if(n<2){
            return n;
        }

        return f(n-1) + f(n-2);
    }
} // this is not the last statement in this function call because f(n-1) gives the ans & f(n-2) gives the ans
```

∴ this is not a tail recursion because it will call F(n-1) then F(n-2) then it will do the addition of it, So this extra step of adding & return are not tail recursion.

Note :- So when you have the last statement in the function call it is known as tail recursion

5 How to understand and approch a problem?

Ans

1) Identify if you can break down a problem into a smaller problem.
2) form the recurrence relation or write if needed
3) Draw the recursive tree
4) about the tree
a) see the flow of function
~~#~~ How they're getting in stack
b) Identify and focus on left tree calls & right tree calls.
c) Draw the trees & pointer again & again using pen & paper
d) Use a debuger to see the flow

5) See how the values are returned at each step & what type of values are returned (int, string etc) of see where the function call will came out & in the end you will come out of the main function

* Binary search using recursion :-
because in binary search we are dividi
the problem into half ,so the problem is divid
into sub-problems so definitely we can appl
Binary search in recursion.

***

1) So, there are two key areas of focus when you've starting out with recursion programs and you wanna have strong foundation of recursion so the two key areas are :-

a) How the function are getting called? tree? using fig?

b) Variables and datatypes and point a)
eg :- of fibo :- there were three type of variable one is at the argument/ parameter one is being returned and one is something you may be having in the body

which variable type to use in which place and what to return is very important and if some one knows this then the entire recursion is easy for them

3 How to figure out what to pass and what to create and what to returned?

* Working with variables :-

a) Arguments whatever you put in the arguments it's going to go in the next function call

b) return type (simple) whatever you wanna return that'll be ~~in~~ it

c) Body of the funct^n

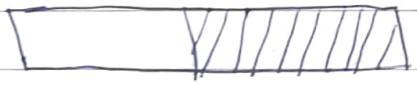
So variable which are specific to that funct^n call that will go inside ~~body~~ of the body

---

* Binary search with recursion

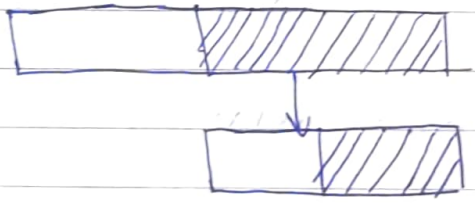
Q So what is binary search?

ans Suppose you've given an array like this



size N

Now it will be like okay let compare the middle element with the target element.



N

$\frac{N}{2}$

So on & So forth

At every steps it's doing two things

1) Comparing :- ~~It~~ is of single step so it takes constant amount of time O(1)

you just need to check whether a no is greater than or equal to or less than the middle no and

It does not depend on the size of an array $O(1)$

2) Dividing into two half

Suppose we're taking a no to find a no in a func^{tn} N = size of array

$$F(n) = O(1) + F\left(\frac{N}{2}\right)$$

- $F(n)$ → Func^{tn} / or binary search
- $O(1)$ → comparison in constant time
- $F\left(\frac{N}{2}\right)$ → Now, search in the array of size $n/2$ (dividing array in to two parts)

This is the recurrence relation.

- If you want to apply binary search on the array of size n, do a step that takes constant amount of time + search in the half of the array

Note Types of recurrence relation

① linear recurrence relation eg:- fibo (+ or −)

② Divide & conquer recurrence relation eg:- binary search

Here the search space is reduced by a factor in above case $\frac{N}{2}$

So, divide the answer into something via a factor

∴ our search space is getting much faster in Divide

and conquer recurrences.
{Dividing a number by 2 is definitely much more faster than subtracting it by 1 or 2 Divide & conquer is much more efficient}

Q Why linear recurrence relation is inefficient?
Ans: ~~to~~ If in the recursion calls two or more recursion calls are doing the same work
(fig:- recursive tree for fibonacci)
Don't compute it again and again

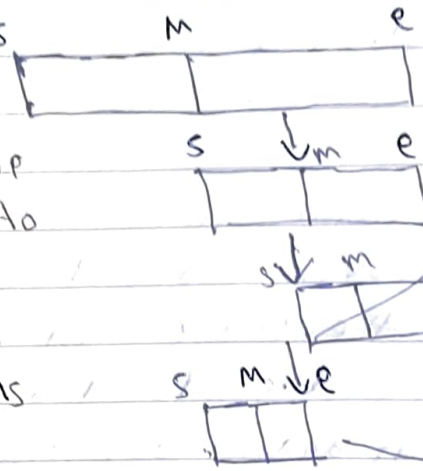
Q So, How can we solve this problem?
Ans Dynamic programming

Tip:- DO NOT OVERTHINK
Now, In binary search what variables do we have i.e start, end & mid
So, from this which will go in the body of the fun^n and which will go in the arguments.

1) So, we are trying to reduce the size of an array and what all the variables.
2) Are the variables that determine whether the size of the array is reduced:-
3) start and end.

a) Whatever you put in the arguments / parameter it's gonna go in the next funct^n call
b) So, variables which specify to the function call will go inside the body of the funct^n

* Continuation of where to take which variables



{Have 2 variables we're focusing on s & e & it will check for m in fn

{Now in this function call it have s & e & again checking m}

{Now in another fun^n call}

{Note:-
this middle value m is really benifical to the faster calls? No that middle value is only benifical to this call
∴ it will go inside the body of the funt^n}

The variable that only you consider to be valuable in that fun^n call put in body

a) Now, Can you see that these 's & e' variables these are actually being passed in the function call. that is why this will go in the arguments and it is very important, because :- we'll be passing these values when we call funtion call in the argument

* inshort:- If there are a few variables that you need to pass in the future function calls then put it inside the argument

b) The variable that you consider to be valuable in that funt^n call only that you don't need to pass inside the future recursion calls then put it inside the body of that functi

c) Make sure to return the result of a function call of the returned type because if you don't return it in the end it will be called after all the sub problem are solved and in the end it will be calling the main function if orignal is not returned the main one will also be not returned so your answer will not be returned so, Return the whatever ans you've getting

code

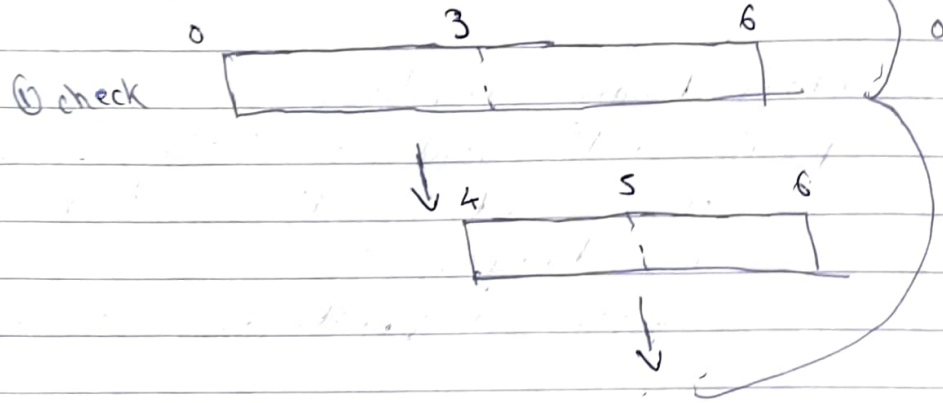
```
public class BinarySearchRec{
    public static void main(String[] args){
        int[] = {1, 2, 3, 13, 18, 98, 165};
        int target = 98
        System.out.println(search(arr, target, 0
                                  , arr.l - 1
    }

    static int search(int[] arr, int target, ints
                                  , inte)
        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, start, mid-1);
        }
```

```
        return search(arr, target, m+1, e);
    }
}
```

target = 98

0	1	2	3	4	5	6
1	2	3	13	18	98	165

main () — // ans ⟵ then functio



∴ start = m+1 = 5

Found the ans return 5

where it is going to return from where it was called

* if there is a return condition, make sure you're returning the type which is same as the return type.

* make sure your returning whatever the sub recursion calls are giving
sub recursion calls is happening make sure your return it.