Date: / / Page no:

Recursion :-

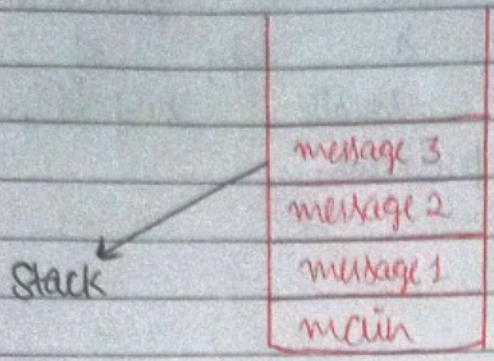
"A function that calls itself is known as recursion."

Before starting the recursion let's see some workings of function.

Example :-

```
public class Message
{
    public static void main (String[] args)
    {
        message1 ();
    }
    Static void message1 ()
    {
        Sout ("hello");
        message2 ();
    }
    Static void message2 ()
    {
        Sout ("hello");
        message3 ();
    }
    static void message3 ()
    {
        Sout ("hello");
    }
}
```

- How function calls work in languages:-

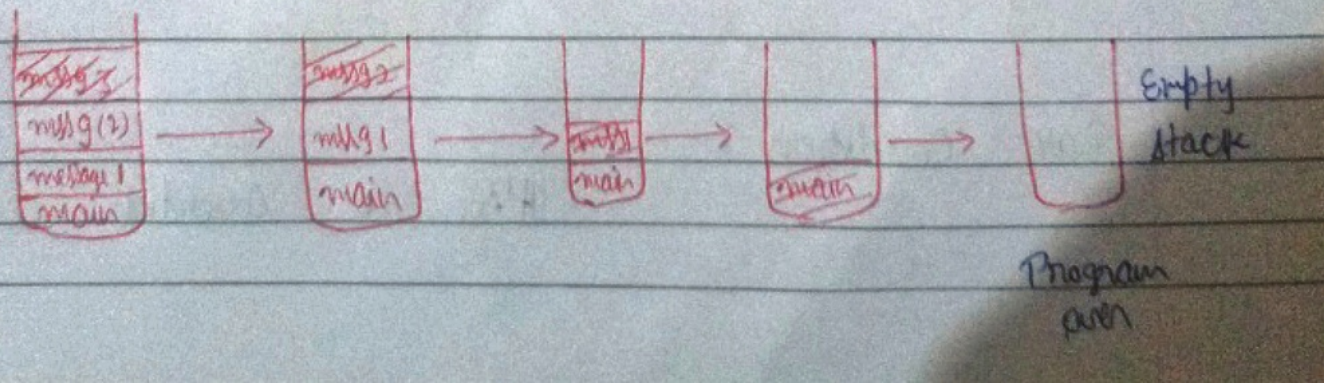


output -
hello
hello
hello

* While the function is not finished executing it will remain in stack.

- Main function goes first in the stack and comes out in the last.
  Because it will call message1() and message2() and so on. message3() function will go in the end and will finish first. then message2() will finish and in the end

* When a function finish executing it is removed from stack and the flow of program is restored to from where that function was called.



ans ⇒ what is recursion ?

ans ⇒ A recursive function solve a particular program by calling a copy of itself and solving smaller subproblems of the original problems.

Example :-

```
public class NumberExampleRecursion
{
    public static void main (String[] args)
    {
        print (1);
    }
    Static void print (int n)
    {
        if (n==5)    // Base Condition
        {
            Sout (5);
            return;
        }
        Sout (n);
        print (n+1);    // Recursive call
    }
}
```

- Base Condition :-

  It's a Condition

where our recursion stop making new call.

- **Stack overflow error :-**

This error occurs when a computer program tries to use more memory space in the call stack than has been allocated to the stack.

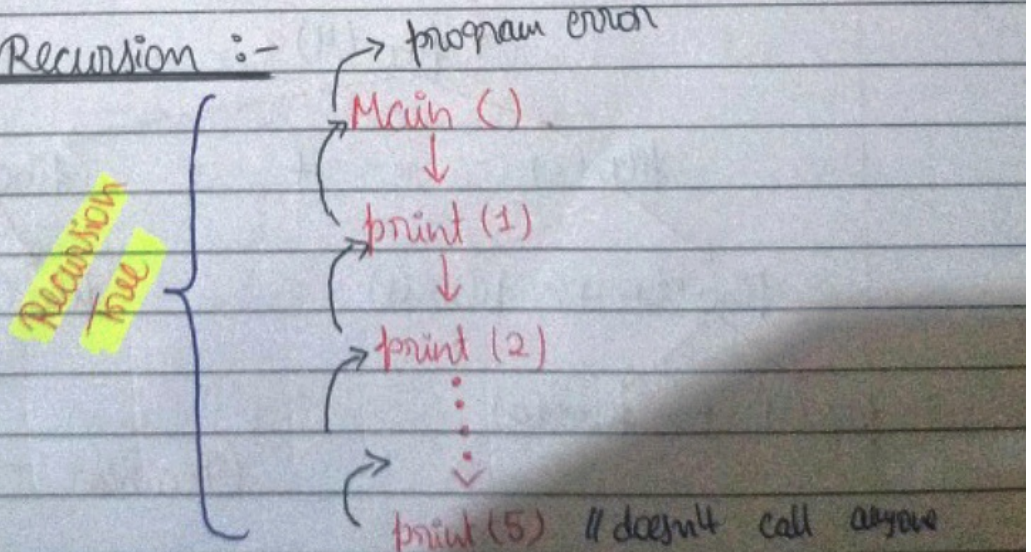
- **Why Recursion :-**

i) It helps us in solving bigger complex problem in a simpler way.

ii) You can convert recursion solution into iteration and vise versa.

iii) Space complexity is not constant because of recursive calls.

- **Visualize Recursion :-**



Date: / / Page no:

Fibonacci no :-

find n^{th} fibonacci no.

0, 1, 1, 2, 3, 5, 8, 13, 21, ... n

* How do we know that the problem can be solved through recursion?

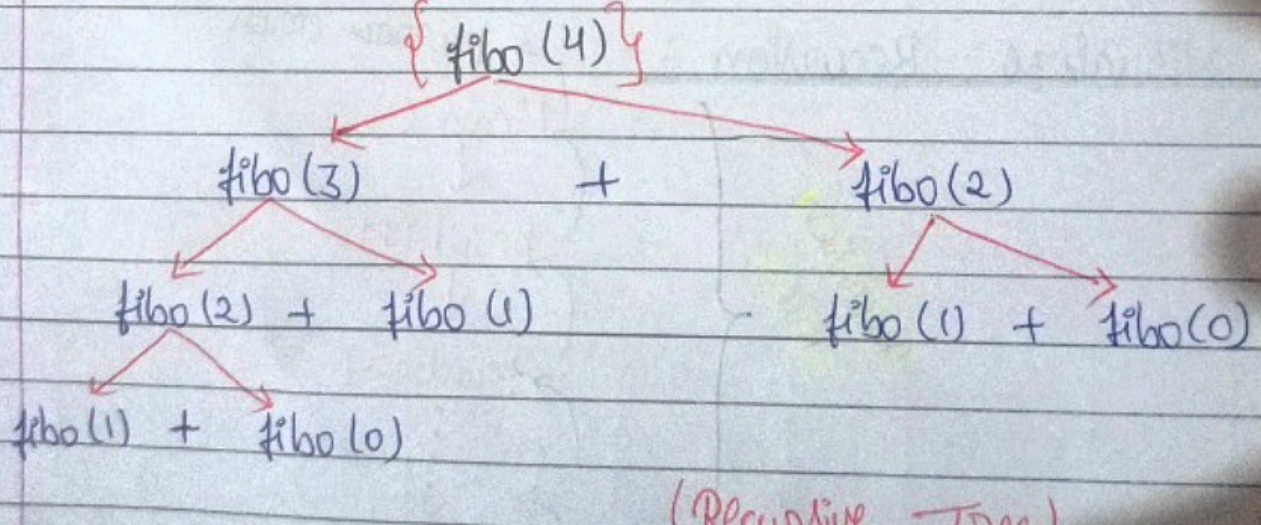
⇒ We can see that the problem can be broke into smaller parts.

How? →

Fibonacci(n) = fibo(N-1) + fibo(N-2)

Recurrance Relation :-

When we write recursion in formula it is known as recurrance relation.



(Recursive Tree)

* Program :-

```
public class fibonacci
{
public static void main (String[] args)
{
    Sout (fibo(8));
}
static int fibo (int n)
{
  if (n<2)  // Base Condition
  {
    return n;
  }
return fibo(n-1) + fibo(n-2); // Recursive call
}
}
```

- Steps to understand the problem :-

i) identify if you can break down problem into smaller one.
ii) Reccurance relation.
iii) draw the recursive tree.
iv) identify and focus on left tree calls and right tree calls.
v) See how the values are returned at each step.
vi) You will come out of the main function.

Date: / / Page no:

- Area to focus in recursion :-

  variable :- i> arguments
  ii> Return type
  iii> Body of function.

Ques ⇒ Binary search with recursion.

ans ⇒ Steps :- i> comparing
ii> dividing into 2 half

$$f(n) = O(1) + f(N/2)$$ — Recurrence Relation.

($O(1)$: constant time; $f(N/2)$: divide array in half)

- Code :-

Function :-

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

```
if (arr[mid] == target)
{
    return mid;
}
if (target < arr[mid])
{
    return search(arr, target, start, mid-1);
}
else {
    return search(arr, target, mid+1, start);
}
}
```

Types of Reccurance relation :-

i) Linear Reccurance relation -> fibonacci
ii) divide & conquer relation -> binary search.