Lesson:

Recursion In JAVA

Pre requisites :

- Loops
- Basic java syntax

List of concepts involved :

- Introduction to recursion
- Need of recursion
- Base case and recursive calls
- Given an integer, find its factorial
- Given an integer n , find nth fibonacci number
- Given two integers a and b. Find the value of a^b using recursion.
- Efficient way of power calculation
- Staircase problem

Introduction to recursion

A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems. Many more recursive calls can be generated as and when required. It is essential to know that we should provide a certain case in order to terminate this recursion process. So we can say that every time the function calls itself with a simpler version of the original problem. The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

It has basically two parts:

- A precondition that is used to stop this recursive call (base Condition).
- A function that is capable of calling itself (recursive call).

Need of Recursion

Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write. It has certain advantages over the iteration technique which will be discussed later. A task that can be defined with its similar subtask, recursion is one of the best solutions for it. For example; The Factorial of a number.

Terminating Condition:

Just like how we have a condition in an iterative statement to terminate the loop similarity, it must have a terminating condition to terminate the recursive call; otherwise, it will result in an overflow error as it will go inside an infinite recursive call.

Steps of any generic recursive function:

- **Base case :** this is the smallest subproblem whose answer is known or can be calculated. Beyond this point one needs not to make further calls to the function.
- **Recursive calls:** this is dividing the major problem into smaller chunks to assemble answers for the main, the original problem.

Syntax of any generic recursive code is as follows:

```
methodName(paramters){
    if(baseConditions)return;
    methodName(changedParameters);
}
```

This will get more clear when we will solve some problems based on recursion.

Q1. given an integer 'n'. Find the factorial of 'n'.

Input 1: n = 5
Output 1: 120
Explanation : 5! = 5*4*3*2*1 = 120

Input 1: n = 3
Output 1: 6
Explanation : 5! = 3*2*1 = 6

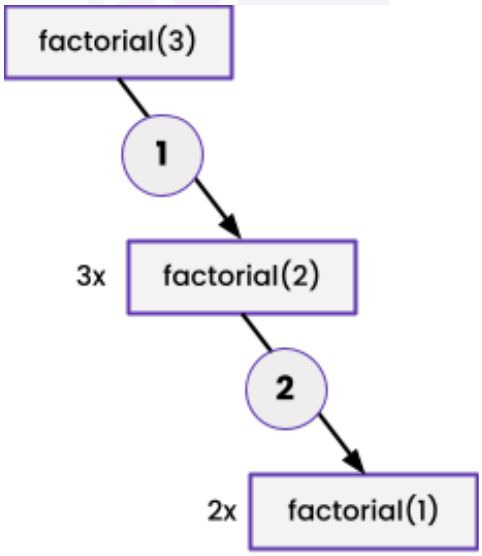
Solution :

Code : LP_Code1.java

Output:

```
Enter the number : 6
The factorial is : 720
```

Approach :



- For any number 'n', we can write its factorial as 'n' multiplied by the factorial of its previous number.
- This is a statement we know to be true. So we recursively call our function to return to us the factorial of the previous number in order to calculate the factorial of the current number by the help of this statement.
- Lets' calculate the factorial of 5 using the recursion code we wrote:
  factorial(5) will call for factorial(4)
  factorial(4) will call for factorial(3)
  factorial(3) will call for factorial(2)

factorial(2) will call for factorial(1)
Now we know the value of factorial(1) to be 1. So we simply return the value i.e. our base case.
Now factorial(1) will return 1. Using this value factorial(2) will be calculated.
factorial(2) = 2 * factorial(1)= 2 * 1 = 2
Similarly,
factorial(3) = 3 * factorial(2) = 3 * 2 = 6
factorial(4) = 4 * factorial(3) = 4 * 6 = 24
factorial(5) = 5 * factorial(4) = 5 * 24 = 120

At the end we have factorial(5)= 120, the desired output.

Q2: given an integer n. Find the nth fibonacci number.

Input1 : n = 7
Output : 13
Explanation : 0 1 1 2 3 5 8 13 21. . . . . .

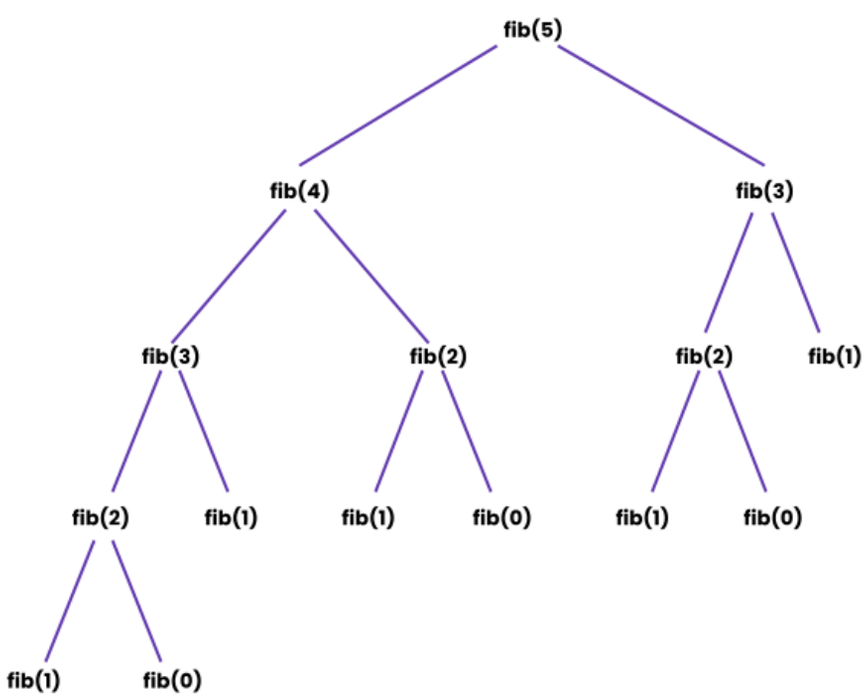
Input1 : n = 5
Output : 5
Explanation : 0 1 1 2 3 5 8 13 21. . . . . .

Solution:
Code : LP_Code2.java

Output :

```
Enter the number : 6
The nth fibonacci number is : 8
```

Approach :
In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation
Fn = Fn-1 + Fn-2
Here F0=0,F1=1, F2=1 .....
Lets find out the 5th fibonacci term :



Now we know that fib(0) = 0 and fib(1) = 1 we can substitute these values in the above tree diagram , then fib(2) = fib(1) + fib(0) = 0 + 1 = 1.
Now fib(3) = fib(2) + fib(1) = 1 + 1 = 2
fib(4) = fib(3) + fib(2) = 2 + 1 = 3
fib(5) = fib(4) + fib(3) = 3 + 2 = 5

Q3: given two integers a and b. Find the value of a^b using recursion.
Input 1: a = 3 , b = 4
Output 1: 81

Solution :
Code: LP_Code3.java

Output:

```
Required answer is 81
```

Approach :

- Create a recursive function say power(int a, int b)
- Base condition : if(b == 0) return 1 because we know anything to the base 0 = 1.
- Otherwise, return (a * power(a , b - 1))

Efficient approach:
Lets see a better approach on how we can improve the time complexity of our solution, lets define a function func which will take two numbers a and b as input and will return a^b. Now this function can we written in this way func(a, b) =func(a, b / 2) ×func(a, b / 2); Like humans do in real life for example a=2, b=4, so for calculating 2^4 = we can find (2^2), take square of it in case b is even. In case b is odd, for example a=2, b=5, so for calculating 2^5 = we can find (2^2), take square of it and multiply it with 2 like (2^5)= (2) * (2^2) * (2^2).

power(a, b) =power(a , b / 2) ×power(a , b / 2); // if b is even

func(a, b) = a ×power(a , (b-1) / 2) ×power(a , (b-1) / 2); // if b is odd

The generated recursion tree when above recurrence relation is implemented to calculate power(3 , 4) is shown below.

As you can observe, func(2,2,), func(2,1) are being computed multiple times. To avoid these redundant computations, we store the result in a result variable while making the recursive call and use this variable value subsequently.

Code : LP_Code4.java

Output :

```
Required answer is 81
```

Approach :

- This function has a base case: b = 0.
- This function has two recursive calls. Only one of them is made in any given call.
- Let's first consider the case when b is even: In that case, the recursive call is made with b / 2. I will store the value return by power(a , b / 2) in result variable and will return square of result variable i.e (result * result).

- In that case when b is odd, the recursive call is made with b / 2. I will store the value return by power(a , b / 2) in result variable and will return square of result variable multiplied by a i.e (a * result * result).

Q4 : There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1 stair or 2 stairs at a time. Count the number of ways the person can reach the top.

Examples:

Input: n = 1
Output: 1

There is only one way to climb 1 stair

Input: n = 2
Output: 2

There are two ways: (1, 1) and (2)

Input: n = 4
Output: 5

(1, 1, 1, 1), (1, 1, 2), (2, 1, 1), (1, 2, 1), (2, 2)

Solution:
Code : LP_Code5.java

Output :

```
Enter the number : 4
The number of ways to reach nth stair is : 5
```

Approach:

- We can easily find the recursive nature in the above problem. The person can reach the nth stair from either (n-1)th stair or from (n-2)th stair. Hence, for each stair n, we try to find out the number of ways to reach n-1th stair and n-2th stair and add them to give the answer for the nth stair. Therefore the expression for such an approach comes out to be :
- ways(n) = ways(n-1) + ways(n-2)
- The above expression is actually the expression for Fibonacci numbers, but there is one thing to notice, the value of ways(n) is equal to fibonacci(n+1).

  ways(1) = fib(2) = 1

  ways(2) = fib(3) = 2

  ways(3) = fib(4) = 3

  *Please refer to the approach of the fibonacci series.