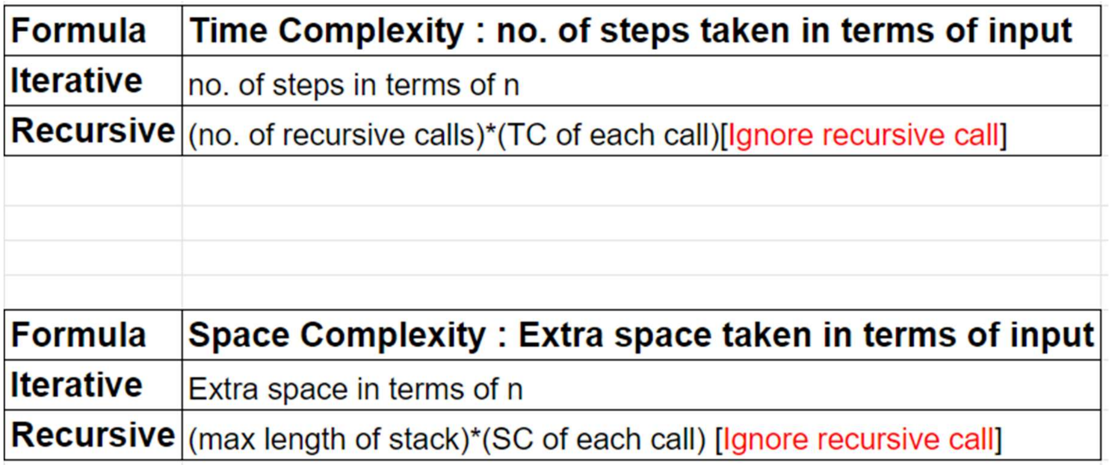
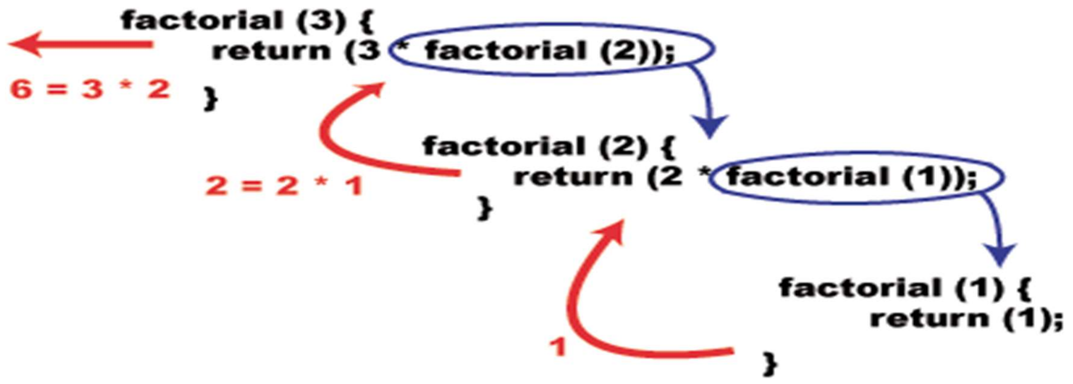
Formulae: -



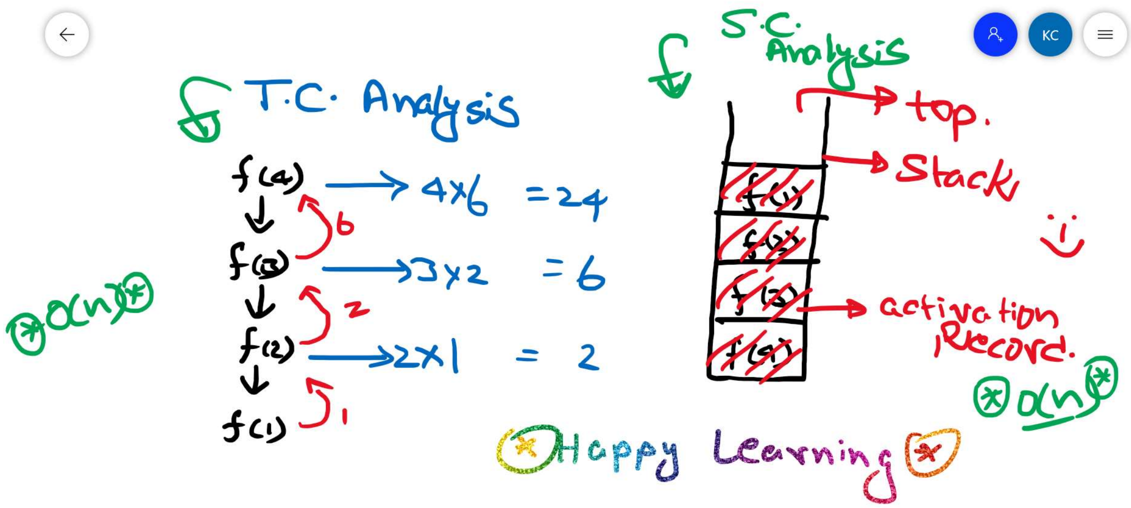
Formula	Time Complexity : no. of steps taken in terms of input
Iterative	no. of steps in terms of n
Recursive	(no. of recursive calls)*(TC of each call)[Ignore recursive call]

Formula	Space Complexity : Extra space taken in terms of input
Iterative	Extra space in terms of n
Recursive	(max length of stack)*(SC of each call) [Ignore recursive call]

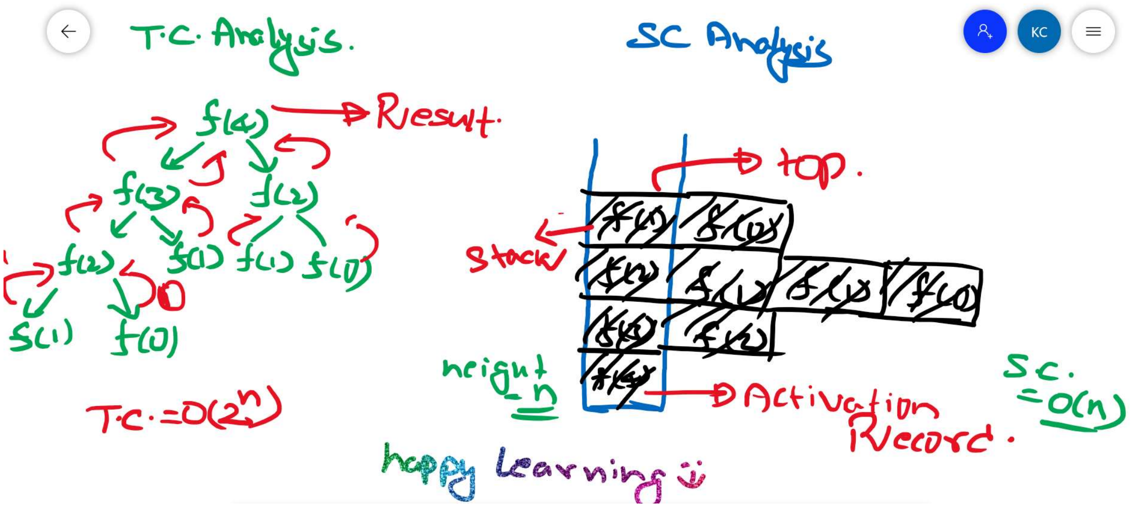
Factorial Using Recursion: -



```
public class FactorialRecursive {
     public static void main(String[] args) {
          System.out.println(fact(5));
     }
     static int fact(int n) {
          if (n == 0 || n == 1) {
               return 1;
          }
          return n * fact(n - 1);
     }

}
```

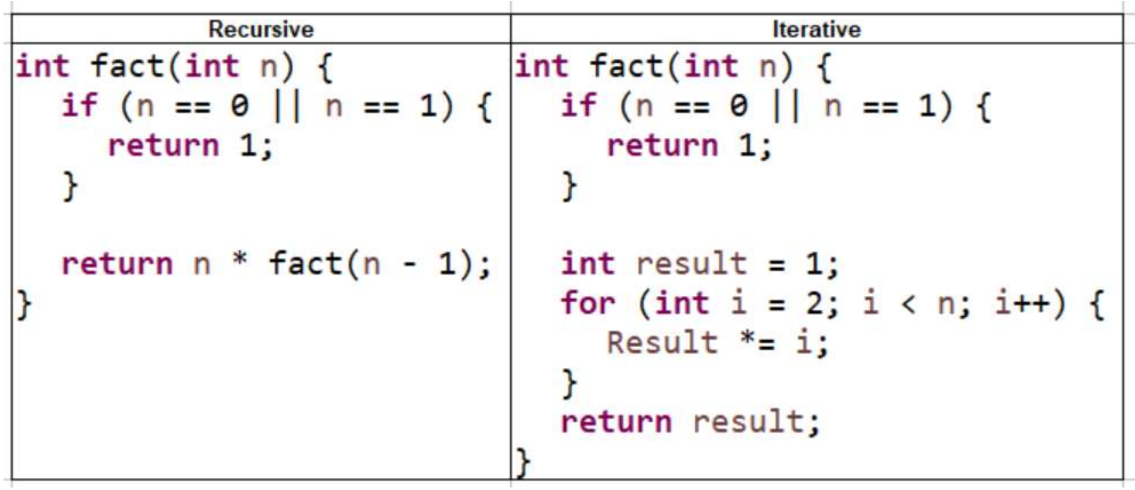


```
public class FibonacciSeriesRecursive {
     public static int fibonacci(int n) {
          if (n <= 0) {
               return 0;
          }
          if (n == 1) {
               return 1;
          }
          return fibonacci(n - 1) + fibonacci(n - 2);
     }
     public static void main(String[] args) {
          System.out.println(fibonacci(4));
     }

}
```



Comparison: -

Program for Factorial:-



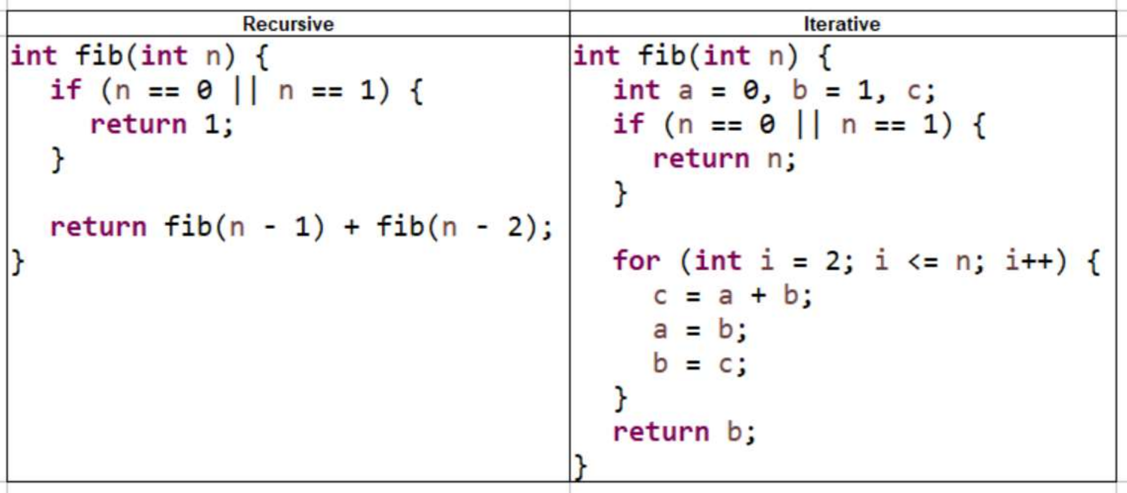
Recursive

```
int fact(int n) {
   if (n == 0 || n == 1) {
      return 1;
   }

   return n * fact(n - 1);
}
```

Iterative

```
int fact(int n) {
   if (n == 0 || n == 1) {
      return 1;
   }

   int result = 1;
   for (int i = 2; i < n; i++) {
      Result *= i;
   }
   return result;
}
```

Program for Fibonacci series: -

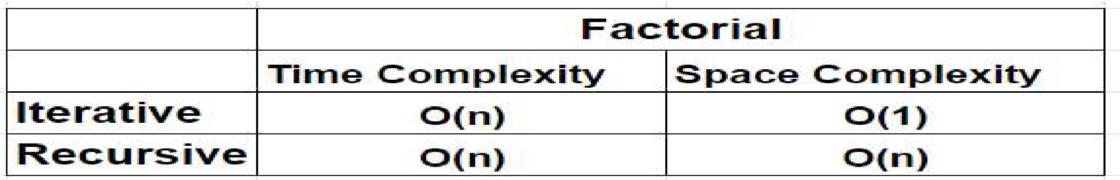


Recursive

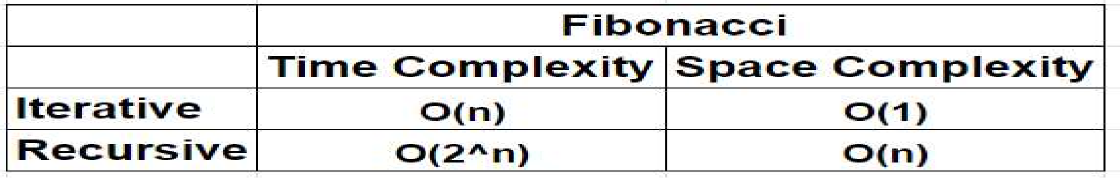
```
int fib(int n) {
   if (n == 0 || n == 1) {
      return 1;
   }

   return fib(n - 1) + fib(n - 2);
}
```

Iterative

```
int fib(int n) {
   int a = 0, b = 1, c;
   if (n == 0 || n == 1) {
      return n;
   }

   for (int i = 2; i <= n; i++) {
      c = a + b;
      a = b;
      b = c;
   }
   return b;
}
```



	Factorial	
	Time Complexity	Space Complexity
Iterative	O(n)	O(1)
Recursive	O(n)	O(n)



	Fibonacci	
	Time Complexity	Space Complexity
Iterative	O(n)	O(1)
Recursive	O(2^n)	O(n)