TYPES OF RECURSION

1. Head Recursion
2. Tail Recursion
3. Indirect Recursion
4. Nested Recursion
5. Tree Recursion

Notes:

1. Head Recursion

Head Recursion: Recursive call is the first statement in the function then it is known as Head Recursion.

There is no statement, no operation before the call. The function does not have to process or perform any operation

at the time of calling and all operations are done at returning time.

Time Complexity: O(n)

Space Complexity: O(n)

Without recursion:

Time Complexity: O(n)

Space Complexity: O(1)

```
class test {

    // Recursion function
    static void fun(int n)
    {
        if (n > 0) {

            // First statement in the function
            fun(n - 1);
```

```
                System.out.print(n + " ");
        }
    }

    static void fun2(int n)
    {
        int i = 1;
        while (i <= n) {
            System.out.print(" "+ i);
            i++;
        }
    }

    public static void main(String[] args)
    {
        int x = 4;
        fun(x);
    }
}
```

2. Tail Recursion

Tail Recursion: Recursive call is the last statement in the function. After that call the recursive function

performs nothing. The function has to process or perform any operation at the time of calling

and it does nothing at returning time.

Time Complexity: O(n)

Space Complexity: O(n)

Without recursion:

Time Complexity: O(n)

Space Complexity: O(1)

```
class test {

	// Recursion function
	static void fun(int n)
	{
		if (n > 0)
		{
			System.out.print(n + " ");

			// Last statement in the function
			fun(n - 1);
		}
	}

	static void fun2(int n)
	{
		while (n > 0) {
			System.out.print(n + " ");
			n--;
		}
	}

	public static void main(String[] args)
	{
		int x = 4;
		fun(x);
	}
```

```
}
```

3. Indirect Recursion

Mutual/Indirect Recursion: In this recursion, there may be more than one functions and they are calling one another in a circular manner.

```
class test {

    static void funA(int n)
    {
        if (n > 0) {
            System.out.print(n + " ");

            funB(n - 1);
        }
    }

    static void funB(int n)
    {
        if (n > 1) {
            System.out.print(n + " ");

            funA(n / 2);
        }
    }

    public static void main(String[] args)
    {
        funA(10);
    }
```

```
}
```

4. Nested Recursion

Nested Recursion: In this recursion, a recursive function will pass the parameter as a recursive call. That means 'recursion inside recursion'

```
class test {

	// Recursion function
	static int fun(int n)
	{
		System.out.print(n + " " );
		if (n > 100)
			return n - 10;

		// Recursion inside the recursion
		return fun(fun(n + 11));
	}

	public static void main(String[] args)
	{
		int result;
		result = fun(99);
		System.out.print(result);
	}
}
```

5. Tree Recursion

Tree Recursion: If a recursive function calling itself for one time then it is known as Linear Recursion.

If a recursive function calling itself for more than one time then it ls known as Tree Recursion.

Time Complexity: O(2^n)

Space Complexity: O(n)

```
class test {
	static int count;

	// Recursion function
	static void fun(int n)
	{
		count++;
		if (n > 0) {
			System.out.print(n + " ");

			fun(n - 1);
			fun(n - 1);
		}
	}

	public static void main(String[] args)
	{
		int x = 4;
		fun(x);
		System.out.println("\nCount " + count);
	}
}
```