


	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 1



GUÍA DE LABORATORIO

(formato docente)

INFORMACIÓN BÁSICA					
ASIGNATURA:	ESTRUCTURA DE DATOS Y ALGORITMOS				
TÍTULO DE LA PRÁCTICA:	TÉCNICAS Y DISEÑO DE ALGORITMOS				
NÚMERO DE PRÁCTICA:	02	AÑO LECTIVO:	2025 – A	NRO. SEMESTRE:	TERCERO III
TIPO DE PRÁCTICA:	INDIVIDUAL	X			
	GRUPAL	—	MÁXIMO DE ESTUDIANTES	00	
FECHA INICIO:	13/05/2025	FECHA FIN:	17/05/2025	DURACIÓN:	90 minutos.
RECURSOS A UTILIZAR: <ul style="list-style-type: none"> • Github. • Lenguaje de Programación Java. • Ide Java Eclipse/Visual Studio Code. 					
DOCENTE(s): <ul style="list-style-type: none"> • Mg. Ing. Rene Alonso Nieto Valencia. 					

OBJETIVOS/TEMAS Y COMPETENCIAS	
OBJETIVOS: <ul style="list-style-type: none"> • Aprenda técnicas y diseño de algoritmos. • Aplicar conceptos elementales de programación (condicionales, bucles, arreglos, etc.) a resolver en recursividad en problemas de algoritmos. • Desarrollar pruebas. 	
TEMAS: <ul style="list-style-type: none"> • Introducción. • Recursividad. • Técnicas y Diseño de Algoritmos. 	
COMPETENCIAS	C.a
	C.b
	C.c
	C.d

CONTENIDO DE LA GUÍA

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 2</p>

I. MARCO CONCEPTUAL

INTRODUCCIÓN

Continuando con la serie de guías de laboratorio, ahora que hemos aprendido sobre estructuras básicas de programación, es hora de aprender un poco sobre Recursión. En esta guía será presentada una introducción al tema, de una manera más objetiva y aún más simple, pero lo necesario para que puedas programar utilizando este recurso.

RECURSIVIDAD

En términos generales, la recursividad puede considerarse como un proceso de repetición de una rutina. Por tanto, de forma muy simplista, se puede definir como una rutina (procedimiento o función) que se llama a sí misma, directa o indirectamente.

Bueno, si la rutina se llama a sí misma muchas veces, entonces debes tener mucho cuidado con el **BUCLE**. Creo que todo el mundo ha oído hablar de este término, pero aclarando para los que no están acostumbrados. Un BUCLE se produce cuando una parte del código se sigue repitiendo eternamente, lo que hace que todo el sistema se bloquee.

Para que no se produzca un BUCLE, es necesario definir correctamente una condición de finalización, ya que, en ocasiones, incluso la condición de finalización puede ser la causa del BUCLE. Por lo tanto, es necesario analizar y evaluar bien el problema que se está resolviendo recursivamente y cómo debe terminar, sin entrar en **LOOP**.

Una rutina que está formada por un conjunto de comandos y una llamada a sí misma se llama Recursión Directa. Una rutina que contiene una llamada a otra rutina que, a su vez, tiene una llamada a otra rutina y así sucesivamente, se llama Recursión Indirecta, por tanto, rutinas diferentes.

Se llama a una función recursiva para resolver un problema, que sabe resolver solo la “parte” más simple, el “caso” más trivial. Por lo tanto, la solución a un problema recursivo generalmente se puede dividir en dos partes: una donde la solución es trivial y otra donde la solución es más general.

De esta manera, la recursividad aplica una técnica llamada divide y vencerás, que funciona más o menos así: si el problema a resolver es muy grande, divídelo en dos; si aún queda grande, divídelo en dos nuevamente; y así sucesivamente, hasta llegar a algo lo más simple posible.

En el caso de la recursividad, sabe resolver una parte del problema, pero otra parte de la que no está muy seguro, por lo que la recursividad divide el problema en dos partes sucesivamente: la primera parte en la que sabe resolverlo y la segunda parte que no sabe cómo resolverlo, siendo este similar al problema original que está tratando de resolver, pero debe ser una versión más simple o más pequeña. De esta forma tenemos:

- **Caso Trivial:** Fácil de resolver y devuelve un resultado, por lo general es el caso más básico o simple del problema, incluso sin necesidad de aplicar la recursividad.
- **Caso General:** En este caso, el problema es esencialmente el mismo que el problema original, pero debe ser una versión más general. Debido a que este nuevo problema es similar al original, la rutina llama a una nueva copia de sí misma para trabajar en el problema más pequeño.

Mientras sea necesario dividir el problema en problemas más pequeños, la función recursiva seguirá llamándose a sí misma, para seguir dividiendo, hasta llegar al caso más básico, y cuando eso sucede, la función deja de dividir y comienza a generar los resultados. Todos los datos de todas las variables involucradas en la

función recursiva deben almacenarse en cada llamada, esto significa que se debe crear una pila de llamadas de la función.

Cuando la función comienza a devolver los resultados (devoluciones de llamada), entonces tenemos las llamadas de función saliendo de la pila de control de llamadas y regresa la función recursiva, que devolverá a la persona que llama al resultado solicitado. Este resultado se agrupa con el siguiente y así sucesivamente, hasta formar el resultado final. A medida que se extraen, los valores también se liberan de la memoria, tal como están en una pila dinámica. El siguiente algoritmo paso a paso ilustra el proceso recursivo descrito hasta ahora:

¿Acabado?

Si sí:

Devolver los resultados.

Sí no:

Simplifique el problema;

Resolver los problemas más simples;

Ajustar los resultados a la solución del problema original;

Devuelve la solución.

Ejemplo de recursividad: Factorial

El factorial de un número natural “n”, representado por n!, es el producto de todos los números enteros positivos menores o iguales a “n”, como se define en la siguiente ecuación matemática:

$$n! = n * (n - 1) * (n - 2) * (n - 3) * ... * 1$$

- **Caso trivial (problema menor):** $0! = 1$ y $1! = 1$
- **Caso general (problema complejo):** $n! = n * (n - 1)!$

Ejemplos:

$$2! = 2 * 1 = 2$$

$$3! = 3 * 2! = 3 * (2 * 1) = 6$$



$$4! = 4 * 3! = 4 * (3 * 2 * 1) = 24$$

Recursión X Iteración

Hay una discusión sobre cuándo debemos usar la recursividad en nuestra solución. Algunos problemas son naturalmente recursivos y otros pueden definirse en términos recursivos. Una función se puede escribir como una función recursiva sin usar la iteración y, por lo tanto, a la inversa, una función recursiva se puede describir a través de iteraciones sucesivas. Sin embargo, es necesario, en primer lugar, comprender la diferencia entre recursividad e iteración, y luego elegir cuál usar.

Tanto la iteración como la recursión utilizan la repetición. La iteración usa la repetición en forma de instrucciones de repetición (**for**, **while**, **do-while**), mientras que la recursividad usa la repetición en forma de llamadas repetitivas a una rutina.

Ambos necesitan una prueba de terminación. La iteración termina cuando falla la condición de prueba y la recursión termina cuando se alcanza el caso trivial. Ambos pueden entrar en un bucle infinito, en el caso de

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 4</p>

iteración si la prueba nunca se vuelve falsa y en el caso de recursividad si el problema no se reduce de manera que converja al caso trivial.

Algunos lenguajes de programación también son del tipo recursivo, son lenguajes de programación funcional (LISP) y programación lógica (**PROLOG**), muy utilizados incluso en inteligencia artificial, robótica y automatización. Las estructuras de datos dinámicas, como árboles, colas, pilas y listas, también utilizan la recursividad.

Una vez clara la diferencia entre el uso de estos dos recursos computacionales, tendrás más claridad para decidir cuál usar en tu proyecto. También es importante decir que se debe hacer un análisis de la complejidad de un algoritmo recursivo. A menudo, el tiempo computacional dedicado a una función recursiva en particular no vale la pena y es menos costoso implementar la misma función de forma iterativa.

El factorial es una de esas funciones que se pueden implementar tanto de forma recursiva como iterativa, y varios otros problemas caerán en esta situación. Dependerá del diseñador del algoritmo implementar y evaluar lo que realmente valdrá la pena.

II. EJERCICIO/PROBLEMA RESUELTO POR EL DOCENTE

En un editor Java, realizar la integración de los siguientes ejercicios, revisar los resultados obtenidos y realizar una explicación del funcionamiento de forma concreta y clara.

1. Ejercicio 1: Implementación de un método recursivo.

```
public class Recursividad {

    void repetir() {
        repetir();
    }

    public static void main(String[] ar) {
        Recursividad re = new Recursividad();
        re.repetir();
    }
}
```

2. Ejercicio 2: Implementación de un método recursivo que reciba un parámetro de tipo entero y luego llame en forma recursiva con el valor del parámetro menos 1.

```
public class Recursividad {

    void imprimir(int x) {
        System.out.println(x);
        imprimir(x - 1);
    }

    public static void main(String[] ar) {
        Recursividad re = new Recursividad();
        re.imprimir(5);
    }
}
```

3. **Ejercicio 3:** Implementar un método recursivo que imprima en forma descendente de 5 a 1 de uno en uno.

```
public class Recursividad {  
  
    void imprimir(int x) {  
        if (x > 0) {  
            System.out.println(x);  
            imprimir(x - 1);  
        }  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re = new Recursividad();  
        re.imprimir(5);  
    }  
}
```

4. **Ejercicio 4:** Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

```
public class Recursividad {  
  
    void imprimir(int x) {  
        if (x > 0) {  
            imprimir(x - 1);  
            System.out.println(x);  
        }  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re = new Recursividad();  
        re.imprimir(5);  
    }  
}
```

5. **Ejercicio 5:** Obtener el factorial de un número. Recordar que el factorial de un número es el resultado que se obtiene de multiplicar dicho número por el anterior y así sucesivamente hasta llegar a uno.
Ej. el factorial de 4 es $4 * 3 * 2 * 1$ es decir 24.

```
public class Recursividad {  
  
    int factorial(int fact) {  
        if (fact > 0) {  
            int valor = fact * factorial(fact - 1);  
            return valor;  
        } else  
            return 1;  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re = new Recursividad();  
        int f = re.factorial(4);  
    }  
}
```

```
        System.out.println("El factorial de 4 es " + f);  
    }  
}
```

6. Ejercicio 6. Implementar un método recursivo para ordenar los elementos de un vector.

```
class Recursividad {  
    static int[] vec = { 312, 614, 88, 22, 54 };  
  
    void ordenar(int[] v, int cant) {  
        if (cant > 1) {  
            for (int f = 0; f < cant - 1; f++)  
                if (v[f] > v[f + 1]) {  
                    int aux = v[f];  
                    v[f] = v[f + 1];  
                    v[f + 1] = aux;  
                }  
            ordenar(v, cant - 1);  
        }  
    }  
  
    void imprimir() {  
        for (int f = 0; f < vec.length; f++)  
            System.out.print(vec[f] + " ");  
        System.out.println("\n");  
    }  
  
    public static void main(String[] ar) {  
        Recursividad r = new Recursividad();  
        r.imprimir();  
        r.ordenar(vec, vec.length);  
        r.imprimir();  
    }  
}
```

III. EJERCICIOS/PROBLEMAS PROPUESTOS

De acuerdo a los ejercicios propuestos desarrollar los algoritmos y mostrar las siguientes indicaciones:



- Enunciado del ejercicio.
- Código en java desarrollado.
- Resultados obtenidos.
- Explicación breve y concreta del código implementado.

1. Invertir vector de enteros, permite ingresar tamaño y captura de valores del arreglo, el método **invertirArray** calcula y muestra el resultado.

N = 3

A = [1 2 3] -> Asalida=[3 2 1]

```
public int[] invertirArray(int[] A) {
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 7</p>

```
//Ingresar el código aquí
return Asalida;
}
```

2. **Rotar a la Izquierda**, permite ingresar tamaño y captura de valores del arreglo, el método **rotarIzquierdaArray** calcula y muestra el resultado.

Si d=2

A=[1 2 3 4 5] -> Ainvertido=[3 4 5 1 2]

```
public int[] rotarIzquierdaArray(int[] A) {
//Ingresar el código aquí
return Ainvertido;
}
```

3. **Triangulo recursivo 1**. El método **trianguloRecursivo1** calcula y muestra el resultado.

- Si b = 5
- Salida:

```
*
**
***
****
*****
```

```
public void trianguloRecursivo1(int base) {
//Ingresar el código aquí
}
```



4. **Triangulo recursivo 2**. El método **trianguloRecursivo2** calcula y muestra el resultado.

- Si b = 5
- Salida:

```
*
**
***
****
*****
```

```
public void trianguloRecursivo2(int base) {
//Ingresar el código aquí
}
```

5. **Triangulo recursivo 3**. El método **trianguloRecursivo3** calcula y muestra el resultado.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 8</p>

- Si b = 5
- Salida:

```
*
**
***
****
*****
```

```
public void trianguloRecursivo3(int base) {
//Ingresar el código aqui
}
```

6. Cuadrado recursivo. El método **cuadradoRecursivo** calcula y muestra el resultado.

- Si b = 5
- Salida:

```
*****
*      *
*      *
*      *
*      *
*****
```



```
public void cuadradoRecursivo(int base) {
//Ingresar el código aqui
}
```

IV. CUESTIONARIO

1. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.
2. Diferencias entre algoritmos de secuencialidad, decisión e iteración.
3. Que son las clases y métodos genéricos

V. REFERENCIAS Y BIBLIOGRAFÍA RECOMENDADAS:

- Weiss M., Data Structures & Problem Solving Using Java, 2010, Addison-Wesley.
- Weiss M., Data Structures and Algorithms Analysis in Java, 2012, Addison-Wesley.
- Cormen T., Leiserson C., Rivest R., Stein C., Introduction to Algorithms, 2022, The MIT Press
- The Java™ Tutorials - <https://docs.oracle.com/javase/tutorial/>
- Sedgewick, R., Algorithms in Java, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, Part 5:
- Graph Algorithms, Addison-Wesley.
- Malik D., Data Structures Using C++, 2003, Thomson Learning.

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p align="center">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 9</p>

- Knuth D., The Art of Computer Programming, Vol. 1 y 3, Addison - Wesley.

TÉCNICAS E INSTRUMENTOS DE EVALUACIÓN	
<p>TÉCNICAS: <i>Actividades Resueltas</i> <i>Ejercicios Propuestos</i></p>	<p>INSTRUMENTOS: <i>Rubricas</i></p>
<p>CRITERIOS DE EVALUACIÓN Los criterios de evaluación se encuentran en el silabo DUFA ANEXO en la sección EVOLUCIÓN CONTINUA</p>	