

Sesión 01

Introducción a los algoritmos & Métodos de Ordenamiento

Unidad I



**Universidad
Tecnológica
del Perú**

Pautas de trabajo

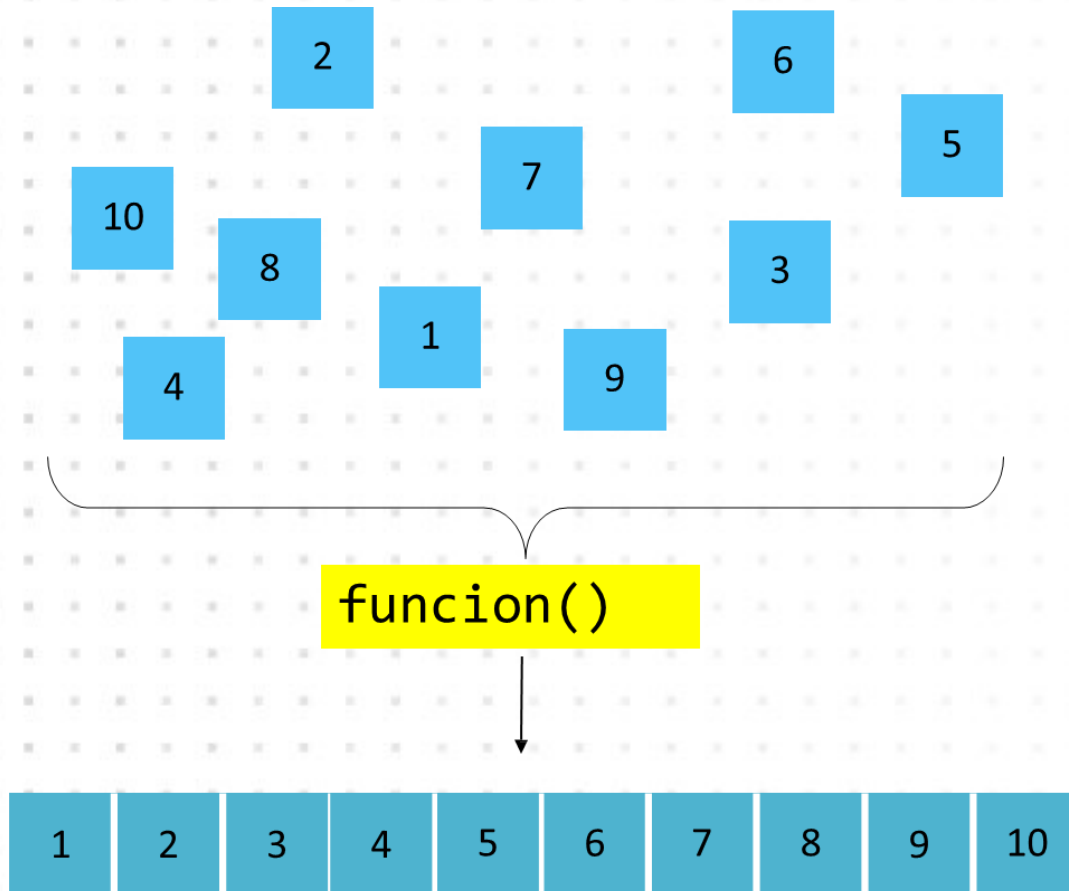
- Los días que tengamos clase debemos tener en cuenta el horario de inicio de clases.
- La participación de los estudiantes se dará acorde a su solicitud levantando la mano.

Inicio:

¿Qué expectativas tienen del curso?

El número de usuarios activos en redes sociales superó los 5,000 millones en 2023, lo que equivale al 62.3% de la población mundial. (AFP)

Utilidad:



Mira la imagen y responde

¿Cómo se puede llamar al procedimiento que realiza el método `funcion()`?

¿En que situaciones has podido observar este proceso?

¿Qué estructura se está utilizando?

Logro de la sesión:

“Al finalizar la sesión, el estudiante resolverá problemas propuestos utilizando algoritmos de ordenamiento básicos, analizando la complejidad algorítmica mediante la notación Big O.”



Temario:

- Prueba de entrada.
- Introducción a los Algoritmos:
- Introducción a las estructuras de datos. Tipos.

Introducción a los algoritmos

¿Qué es un **Algoritmo**?

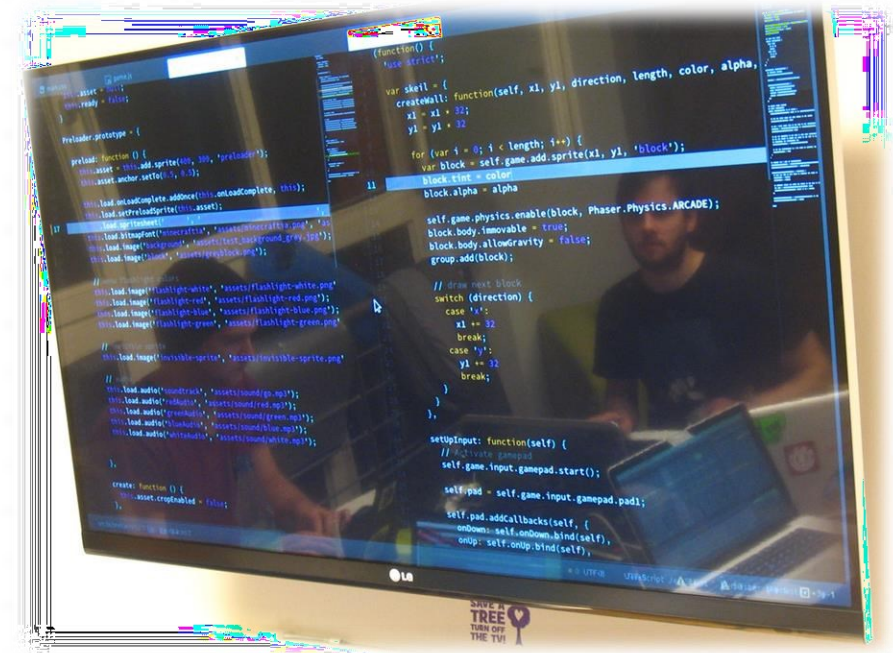
Según (*Joyanes, 2008*), un algoritmo es un **método para resolver un problema** mediante 3 pasos: diseño, codificación y ejecución



Introducción a los algoritmos

¿Qué es un Algoritmo?

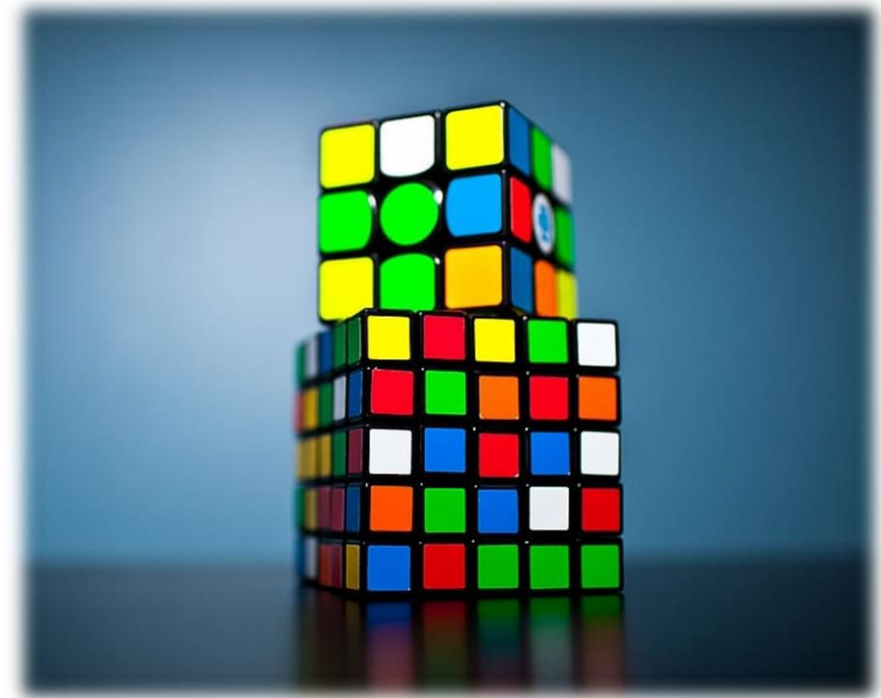
Según (*Sedgewick & Wayne, 2011*), un algoritmo es un **método finito, determinista y eficaz** que resuelve un problema, y que es adecuado para ser un programa de computadora



Introducción a los algoritmos

¿Qué es un **Algoritmo**?

Según (*Vinu V Das, 2006*) un algoritmo es una secuencia finita **paso a paso** de instrucciones para **resolver un problema bien definido**.



Introducción a los algoritmos

¿Cuáles son sus **características?**



Introducción a los algoritmos

¿Cuál es su **estructura**?



Introducción a los algoritmos

¿Qué son los Datos?

- Son hechos «**crudos**», que no han sido procesados
- Son hechos **individuales**
- Por si mismos, sin contexto no tienen mayor significado
- Son **piezas** de información, no información
- La calidad de datos es clave para obtener información
- Pueden extraerse de la información



Introducción a los algoritmos

¿Qué es una **Variable**?

- ✓ Es un elemento que permite **almacenar un valor** de forma temporal
- ✓ El valor que guarda, **puede cambiar** mientras el programa se encuentra ejecutándose
- ✓ Un programa puede tener muchas variables, pero cada una se **identifica** de manera única mediante su **nombre**

edad

19

Introducción a los algoritmos

¿Cómo es **nomenclatura** de variables en Java?

- ✓ Debe comenzar con letras (A-Z, a-z), underscore (_) o signo dólar (\$)
- ✓ Es case sensitive (sensible a mayúsculas)
- ✓ Puede contener números después del primer carácter
- ✓ No puede empezar con números o símbolos matemáticos
- ✓ No puede ser un keyword
(<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/keywords.html>)

CORRECTO

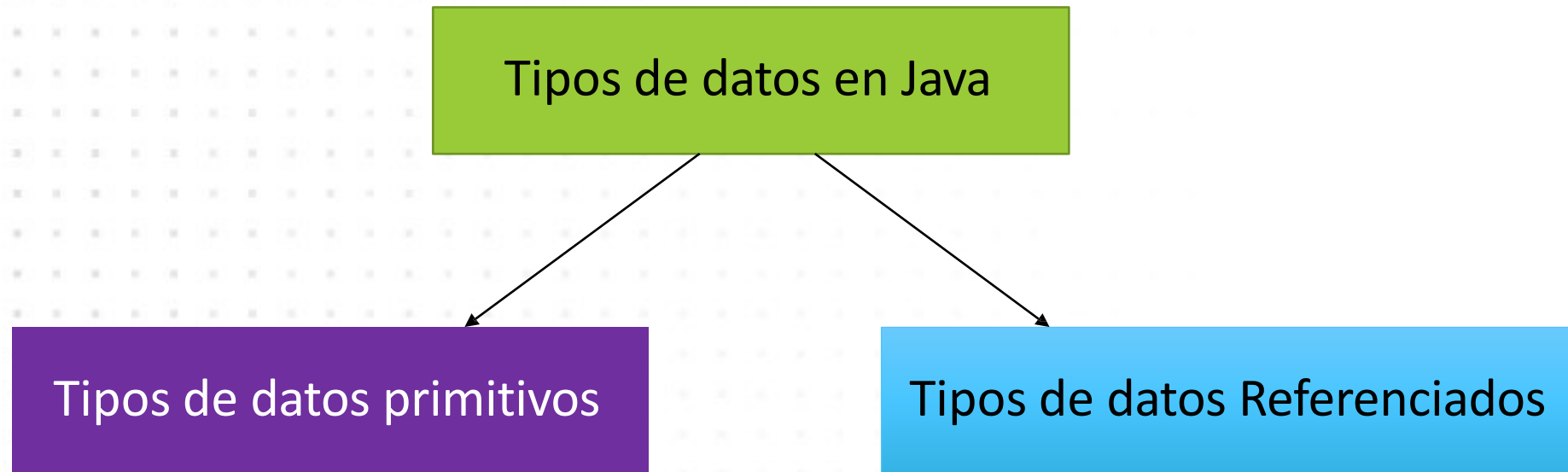
- ✓ edad
- ✓ horas_extra
- ✓ facebook
- ✓ codigo_alumno
- ✓ precio_final
- ✓ notal
- ✓ tiene_sobrepeso
- ✓ item
- ✓ descripcion
- ✓ \$resultado

INCORRECTO

- × 4edad
- × horas extra
- × facebook.com
- × codigo_alumno
- × &precio_final
- × 1raNota
- × class
- × private
- × #descripción
- × 123

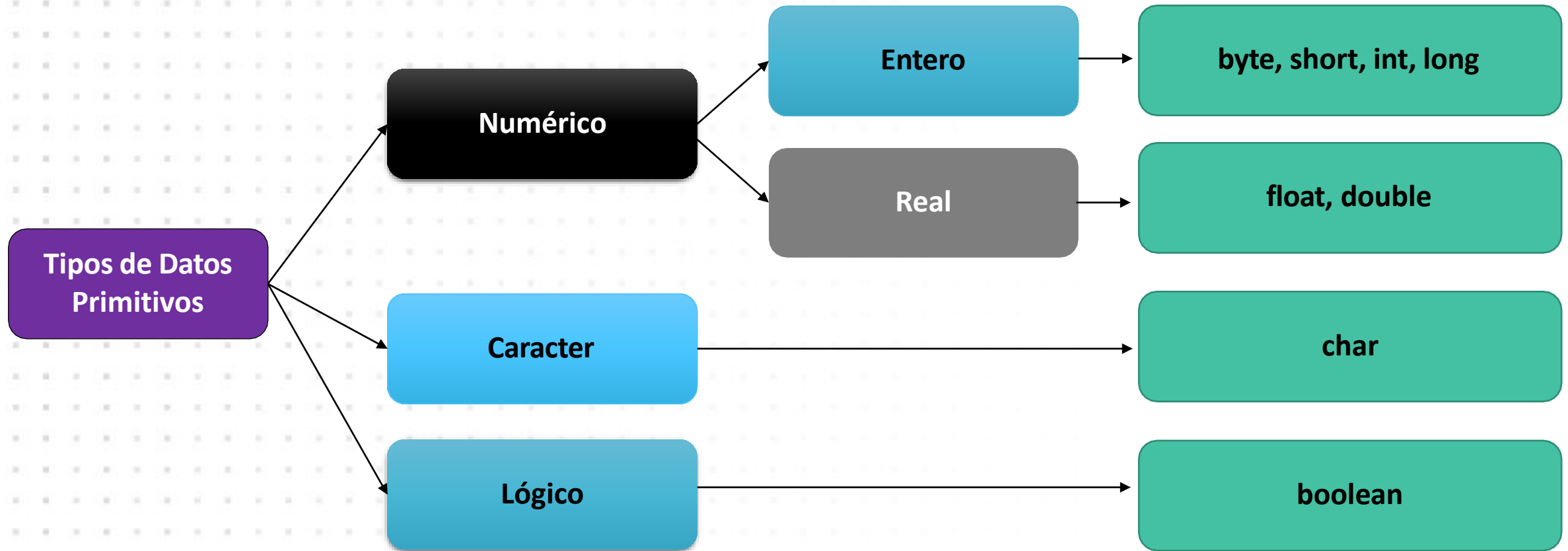
Introducción a los algoritmos

¿Qué Tipos de datos existen Java?



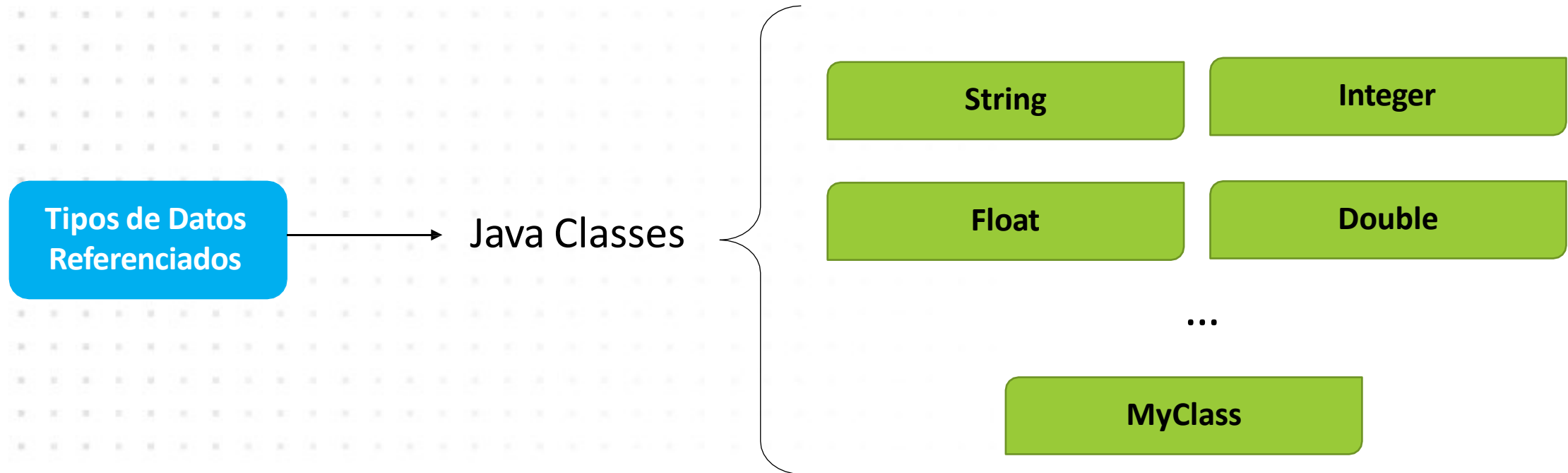
Introducción a los algoritmos

¿Cuáles son los **Tipos de datos primitivos** en Java?



Introducción a los algoritmos

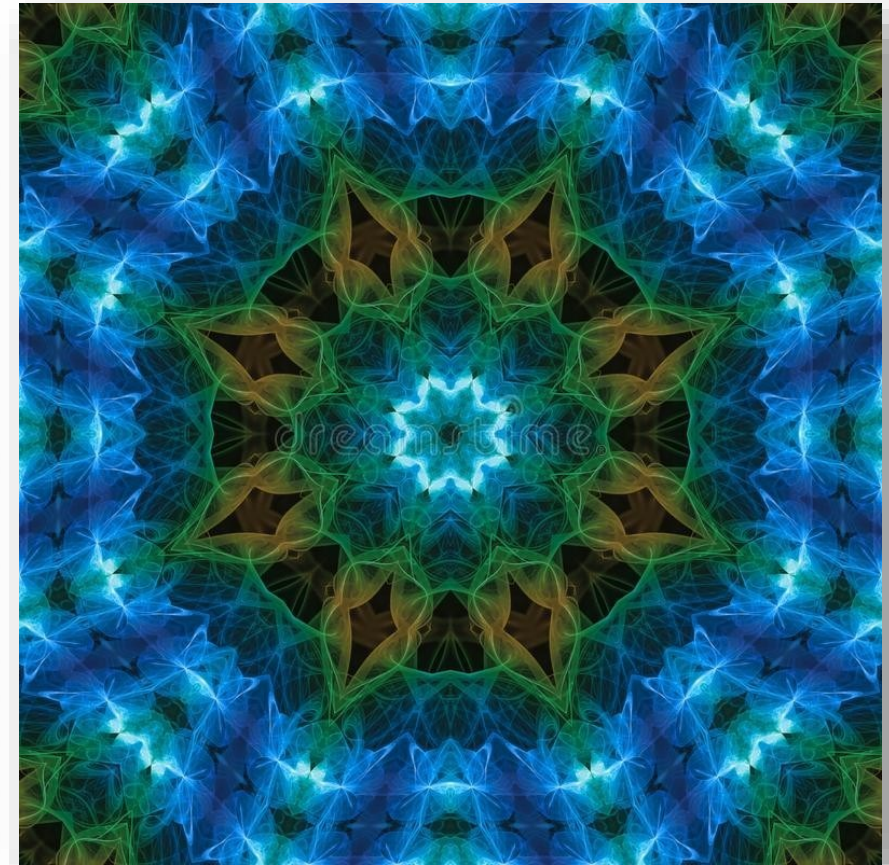
¿Cuáles son los **Tipos de datos referenciados** en Java?



Introducción a los algoritmos

Complejidad algorítmica

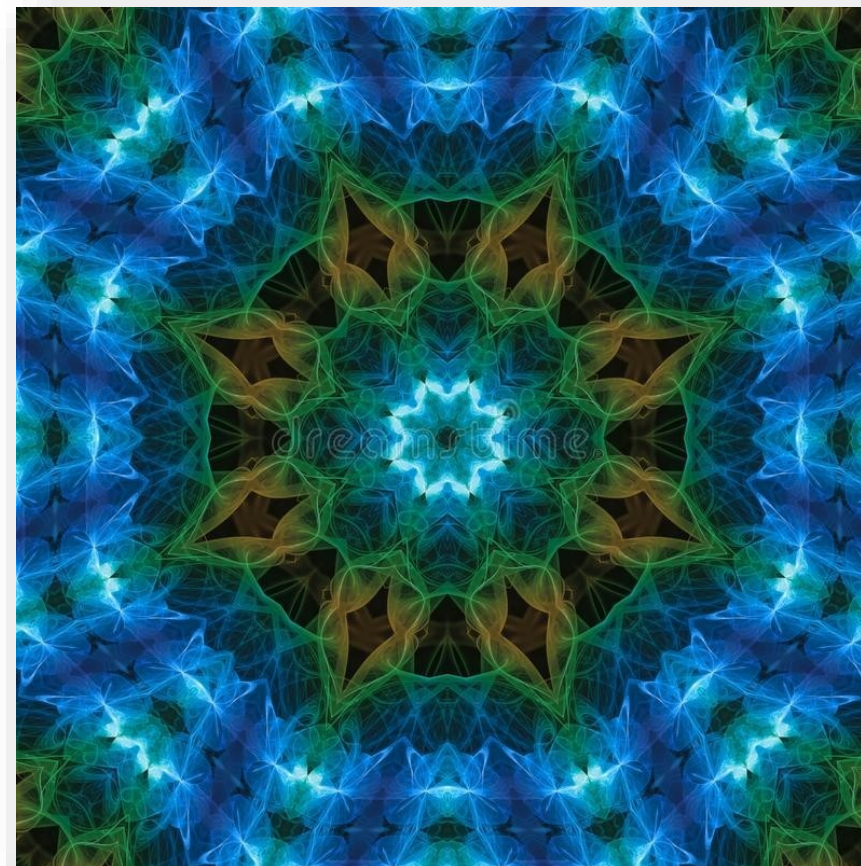
- Representa la **cantidad de recursos que consume** un algoritmo al resolver un problema
- Los criterios empleados para evaluar la complejidad algorítmica **no son absolutas sino relativas** al tamaño del problema
- Un algoritmo puede ser analizado por 2 factores:
 - Complejidad temporal
 - Complejidad espacial



Introducción a los algoritmos

¿Qué es la Complejidad espacial?

- Mide la memoria que necesita el algoritmo para ejecutar sus instrucciones por completo con una entrada de tamaño N
- La memoria consumida es temporal, y se utiliza mientras dure la ejecución del algoritmo
- También depende del tamaño del input





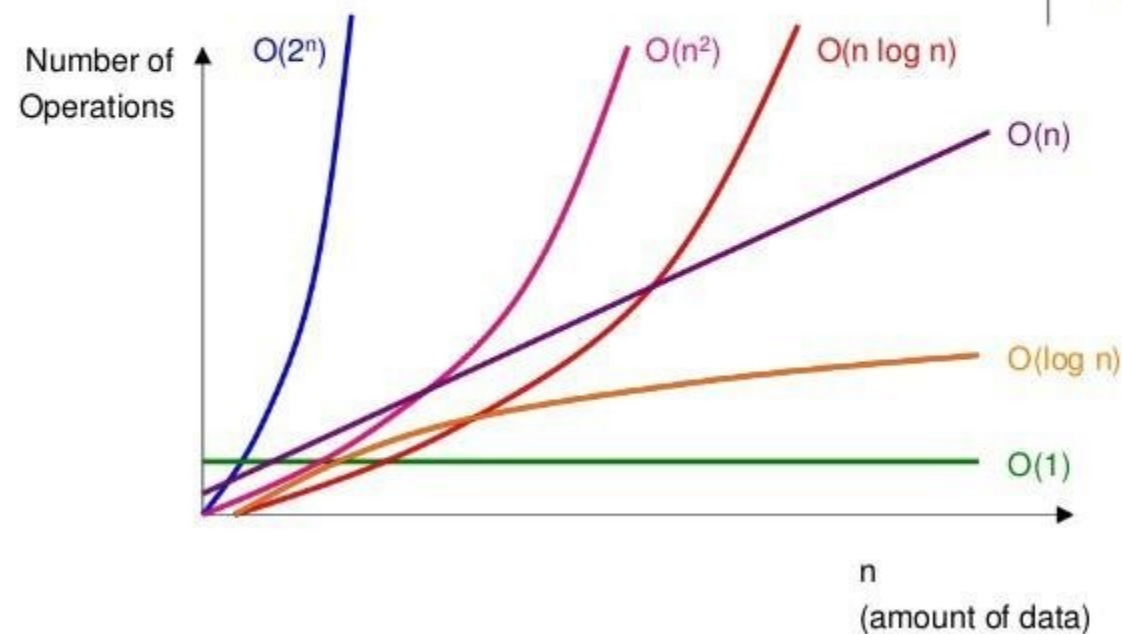
Notación Big-O

Introducción a los algoritmos

¿Qué es la notación Big O?

- Es una notación matemática muy popular para analizar la complejidad de un algoritmo.
- La letra O significa “orden de aproximación”
- Esta notación caracteriza funciones matemáticas según el radio de crecimiento

Comparing Big O Functions



(C) 2010 Thomas J Cortina, Carnegie Mellon University

Introducción a los algoritmos

¿Qué es la notación Big O?

- Entre las funciones más usadas en algoritmos tenemos:

Constante: $O(1)$

Logarítmico: $O(\log n)$

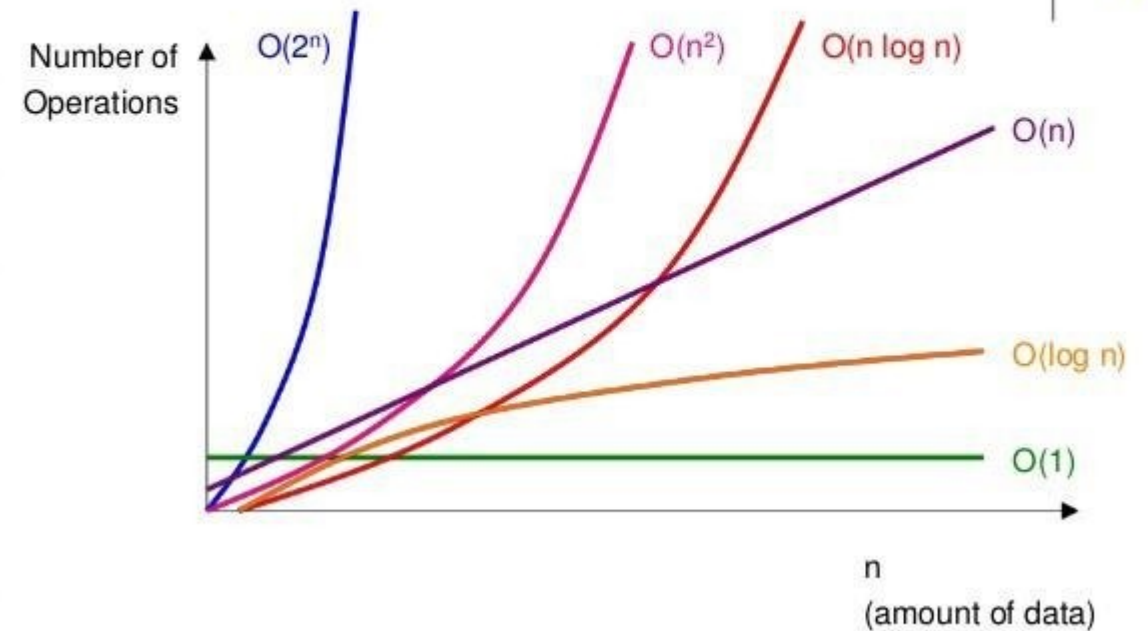
Lineal: $O(n)$

Loglineal: $O(n \log n)$

Cuadrática: $O(n^2)$

Exponencial: $O(2^n)$

Comparing Big O Functions



(C) 2010 Thomas J Cortina, Carnegie Mellon University

Introducción a los algoritmos

Ejemplo de función **Big O Constante**

- En el ejemplo, se utiliza un algoritmo simple para mostrar un valor de texto
- Se evaluará el método llamado “constante”
- En la línea 9, el tiempo de acceso al array “ciudad” siempre será el mismo al margen del índice que se utilice (0, 1 o 2)
- Se concluye que el algoritmo del método tiene una complejidad constante

```
1 package com.empresa;
2
3 ▶ public class Big0 {
4
5     public static void constante(){
6         String[] ciudad = {"Lima",
7                             "Piura",
8                             "Chiclayo"};
9         System.out.println(ciudad[2]);
10    }
11
12 ▶ public static void main(String[] args) {
13     constante();
14 }
15 }
```

Introducción a los algoritmos

Ejemplo de función **Big O Lineal**

- En el ejemplo, se utiliza un algoritmo simple para mostrar todos los datos del array y se evaluará el método llamado “lineal”
- En las líneas 9-11, el tiempo que toma en mostrar todos los datos, va a depender del tamaño del array de manera proporcional
- Se concluye que el algoritmo del método tiene una complejidad lineal

```
1 package com.empresa;
2
3 public class BigO {
4
5     public static void lineal(){
6         String[] ciudad = {"Lima",
7                             "Piura",
8                             "Chiclayo"};
9         for (int i = 0; i < ciudad.length; i++) {
10             System.out.println(ciudad[i]);
11         }
12     }
13
14     public static void main(String[] args) {
15         lineal();
16     }
17 }
```

Introducción a los algoritmos

Ejemplo de función Big O Cuadrática

- En el ejemplo, se utiliza un algoritmo simple para mostrar todos los datos del array N veces y se evaluará el método llamado “cuadratico”
- En las líneas 10-14, el tiempo que toma en mostrar todos los datos N veces, va a depender del tamaño del array de manera proporcional ($N*N$)
- Se concluye que el algoritmo del método tiene una complejidad cuadrática

```
1 package com.empresa;
2
3 public class BigO {
4
5     public static void cuadratica(){
6         String[] ciudad = {"Lima",
7                             "Piura",
8                             "Chiclayo"};
9         int n = ciudad.length;
10        for (int i = 0; i < n; i++) {
11            for (int j = 0; j < n; j++) {
12                System.out.println(ciudad[i]);
13            }
14        }
15    }
16
17    public static void main(String[] args) {
18        cuadratica();
19    }
```

Introducción a los algoritmos

Ejemplo de función Big O Logarítmica

- En el ejemplo, se utiliza un algoritmo simple para mostrar los valores 1024,512,256,128,64,32,16,8,4,2,1 y se evaluará el método llamado “cuadratico”
- En las líneas 7-9, el tiempo que toma en mostrar todos los datos crece al inicio, pero se estabiliza conforme aumenta el tamaño de num
- Se concluye que el algoritmo del método tiene una complejidad logarítmica

```
1 package com.empresa;
2
3 public class BigO {
4
5     public static void logaritmica(){
6         int num = 1024;
7         while(num >=1){
8             System.out.println(num);
9             num = num /2;
10        }
11    }
12
13    public static void main(String[] args) {
14        logaritmica();
15    }
16 }
```

Introducción a los algoritmos



Métodos de Ordenamiento

Métodos de ordenamiento

¿Qué es Swap?

Intercambio

SWAP



Es un procedimiento que permite intercambiar valores u objetos



Métodos de ordenamiento

¿Qué es Swap?

Intercambio

SWAP



Es un procedimiento que permite
intercambiar valores u objetos

nombre1

Juan

nombre2

Rosa

swap

nombre1

Rosa

nombre2

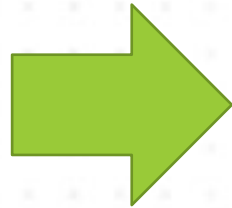
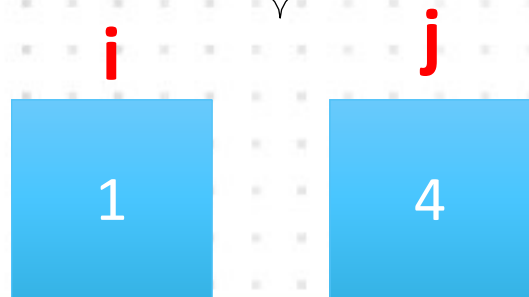
Juan

Métodos de ordenamiento

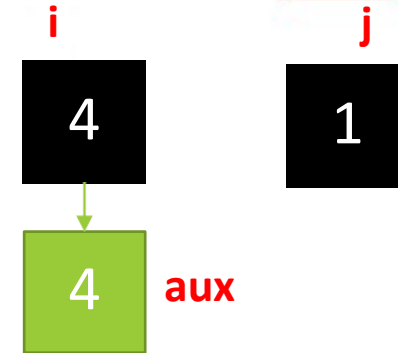
¿Qué es Swap?



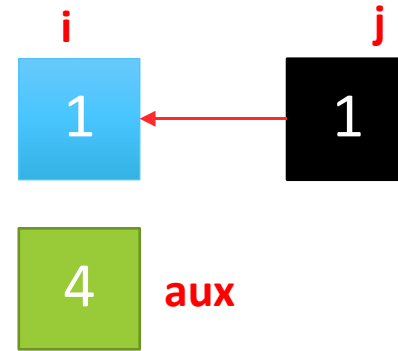
Es el proceso de intercambiar 2 variables independientes o 2 elementos de un array



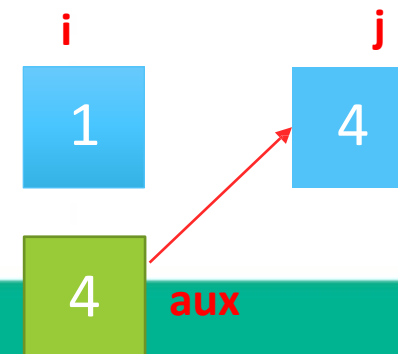
Paso 1:



Paso 2:

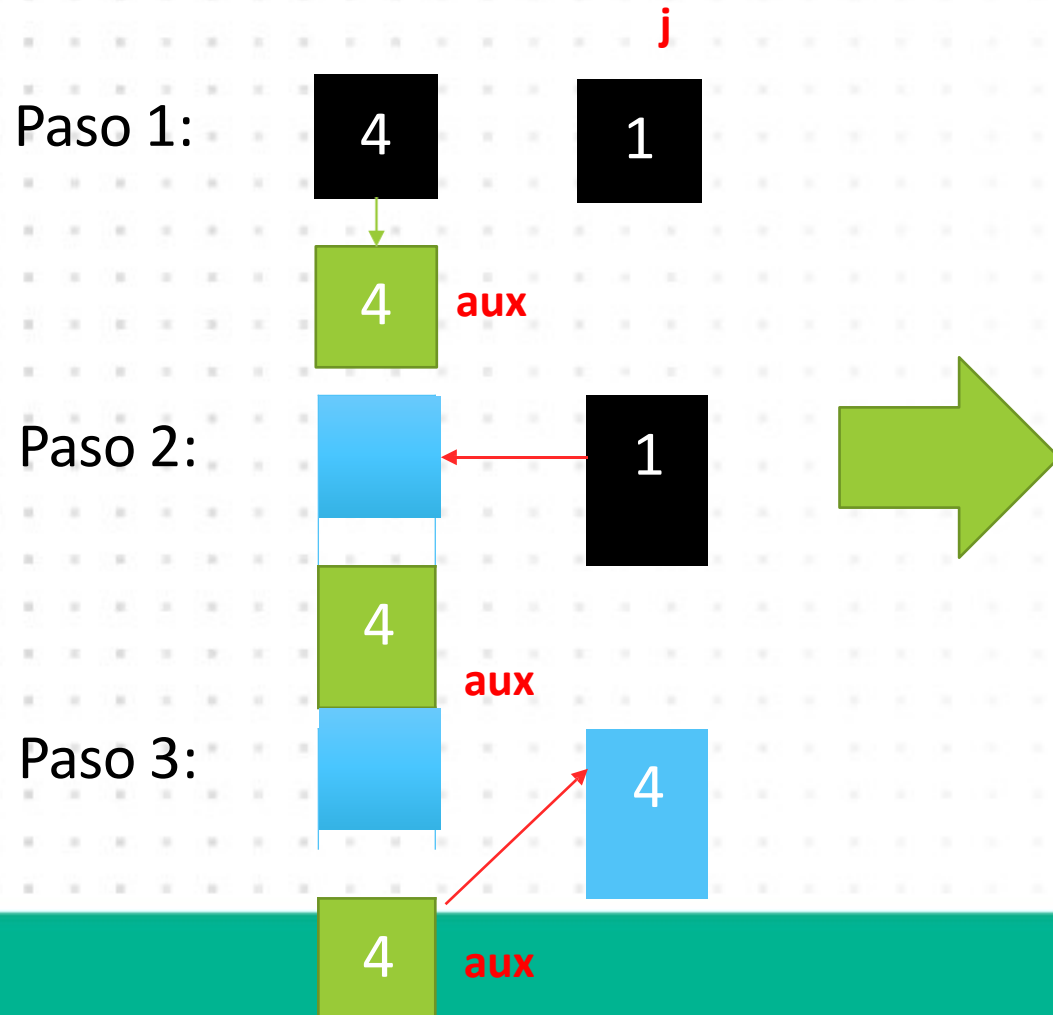


Paso 3:



Métodos de ordenamiento

Array Swap en Java



```
public static void swap(int[] arreglo,  
                        int i, int j) {  
    int aux = arreglo[i];    // Paso 1  
    arreglo[i] = arreglo[j]; // Paso 2  
    arreglo[j] = aux;        // Paso 3  
}
```

Métodos de ordenamiento

Algoritmo de Ordenamiento

- Es un algoritmo encargado de colocar elementos en un orden particular
- Por lo general el orden es numérico
- Sin embargo también se puede realizar un orden alfanumérico
- Existen muchos algoritmos de ordenamiento



Métodos de ordenamiento

Algoritmos de Ordenamiento Básicos

Bubble Sort

Selection Sort

Insertion Sort

Heap Sort

Link: <https://www.toptal.com/developers/sorting-algorithms>

Métodos de ordenamiento



Bubble Sort (Burbuja)

Métodos de ordenamiento

Bubble Sort

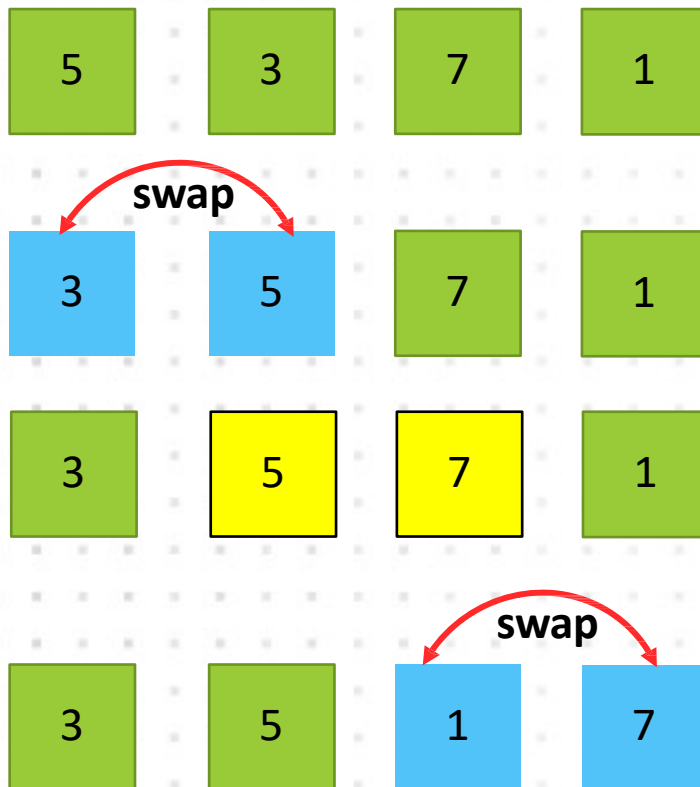
- Es el algoritmo de ordenamiento más simple
- Utiliza 2 bucles for con los que repite una comparación de elementos adyacentes
- Intercambia los valores (**swap**) si están desordenados, caso contrario no los intercambia
- Es un algoritmo de bajo rendimiento



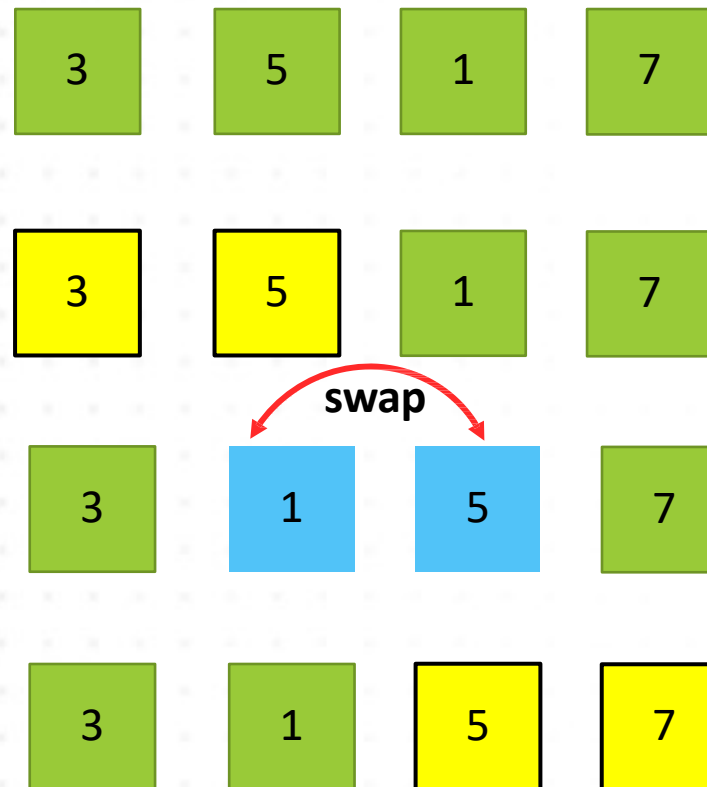
Métodos de ordenamiento

Bubble Sort

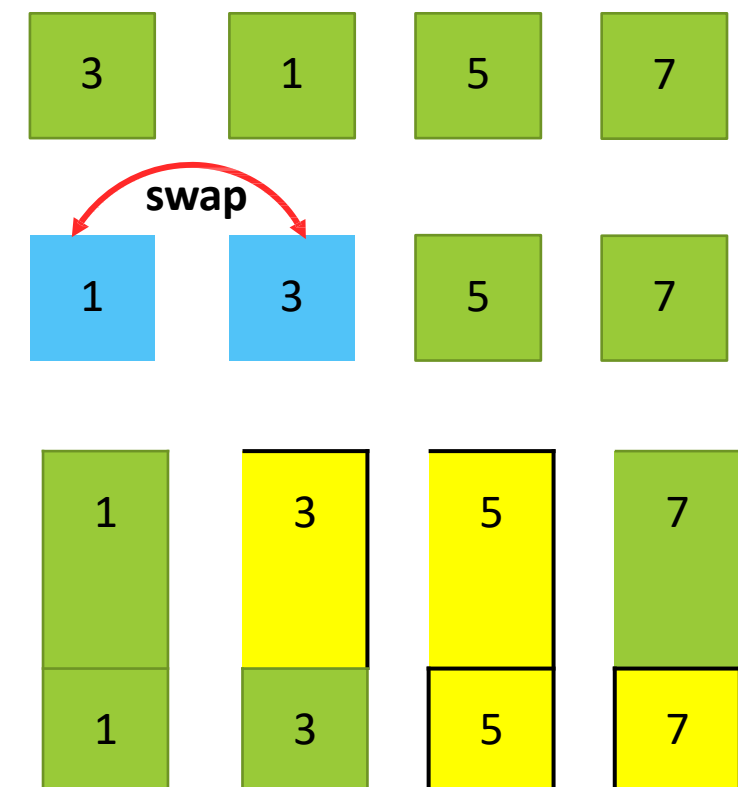
Iteración 1



Iteración 2



Iteración 3



Métodos de ordenamiento

Bubble Sort en Java

Pseudocódigo

```
Para i = 0 hasta i < longitud(arreglo)
    Para j = 0 hasta j < longitud(arreglo)-1
        Si (arreglo[j] > arreglo[j+1])
            swap(arreglo, j, j+1)
        FinSi
    FinPara
FinPara
```

```
public static int[] sort(int[] arreglo){
    for (int i = 0; i < arreglo.length; i++) {
        for (int j = 0; j < arreglo.length-1; j++) {
            if (arreglo[j] > arreglo[j+1]){
                swap(arreglo, j, j+1);
            }
        }
    }
    return arreglo;
}
```



Selection Sort (Selección)

Métodos de ordenamiento

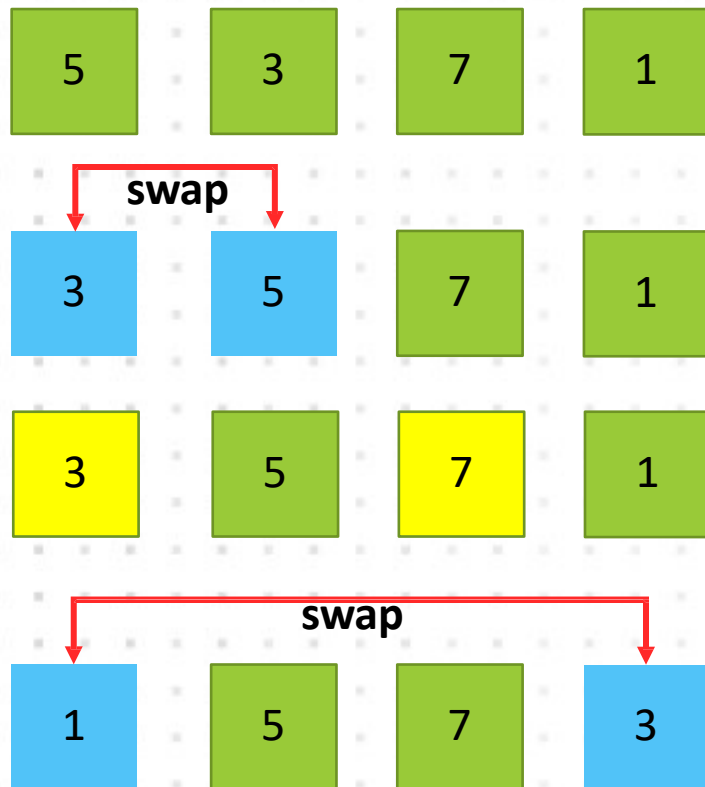
Selection Sort

- Es un algoritmo similar al bubble sort en simplicidad
- Utiliza 2 bucle for en búsqueda del índice con el valor a intercambiar
- Intercambia los valores swap de manera mucho más eficiente que bubble sort
- Es un algoritmo de bajo rendimiento

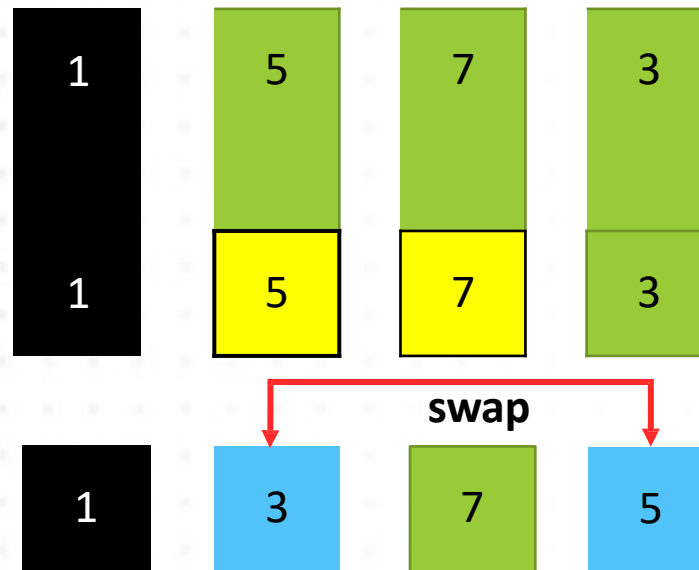


Métodos de ordenamiento

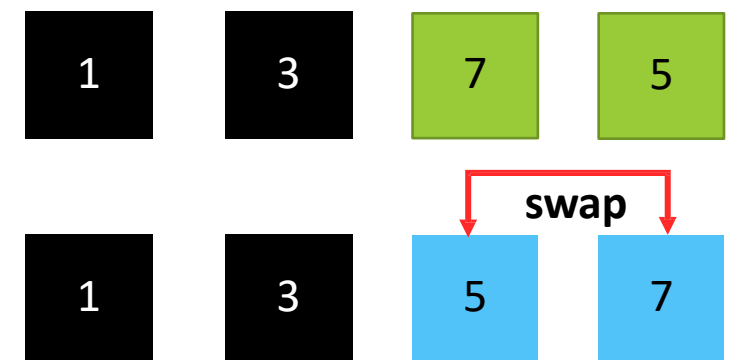
Selection Sort



Iteración 1



Iteración 2



Iteración 3

Métodos de ordenamiento

Selection Sort

1. Buscar el elemento mínimo (**min**) en el arreglo
2. Swap entre **min** y el primer elemento
3. Ordenar el resto del arreglo después del primer elemento de forma recursiva

BuscarIndiceElementoMin

- Considere el arreglo como una sub-arreglo que va desde la **posición inicio** hasta el resto del arreglo
- Buscar el índice de **min** en el resto del arreglo
- Comparar **min** con el valor de la **posición inicio** y actualizar min si es necesario



Métodos de ordenamiento

Selection Sort en Java

```
public static int buscarIdxElementoMin(int[] arreglo,
                                      int inicio){
    int idxMin = inicio;
    for (int i = inicio; i < arreglo.length; i++) {
        if (arreglo[i] < arreglo[idxMin]) {
            idxMin = i;
        }
    }
    return idxMin;
}
```

```
public static int[] sort(int[] arreglo, int inicio){
    if (inicio >= arreglo.length) return arreglo;
    int min = buscarIdxElementoMin(arreglo, inicio);
    if (min != inicio) swap(arreglo, inicio, min);
    return sort(arreglo, inicio+1);
}
```

Métodos de ordenamiento

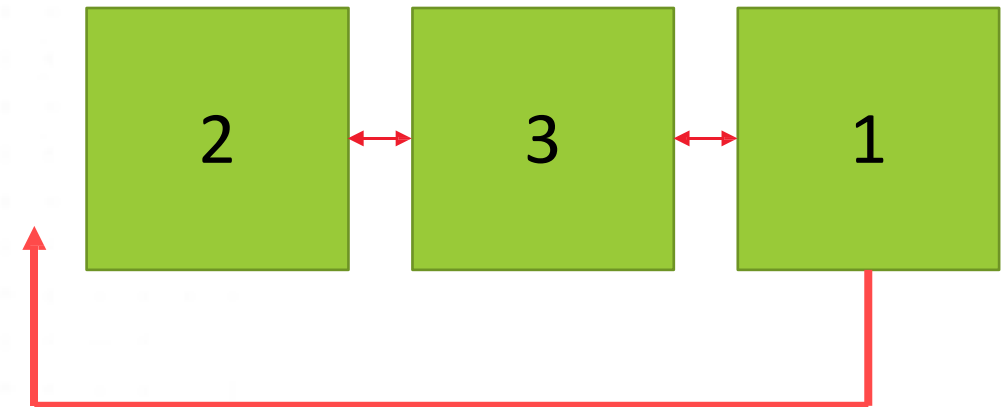


Insertion Sort (Inserción)

Métodos de ordenamiento

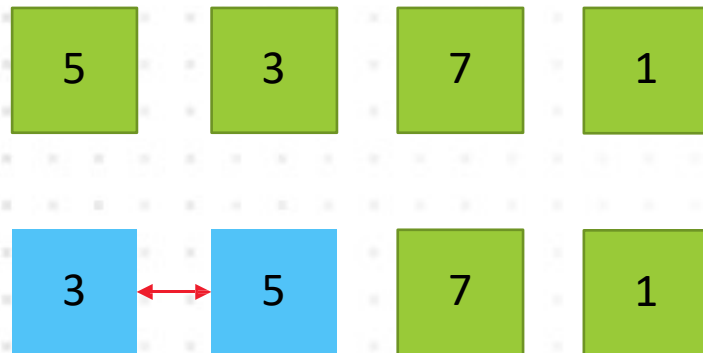
Insertion Sort

- Es un algoritmo similar al bubble sort en simplicidad
- Utiliza 2 bucles while descendentes para realizar la búsqueda
- Intercambia los valores swap de manera extensiva
- Es un algoritmo eficiente con conjuntos de datos pequeños

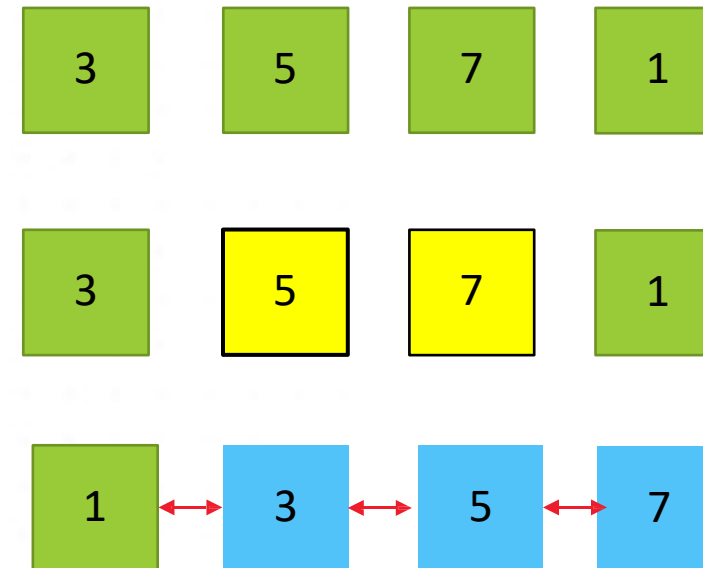


Métodos de ordenamiento

Insertion Sort



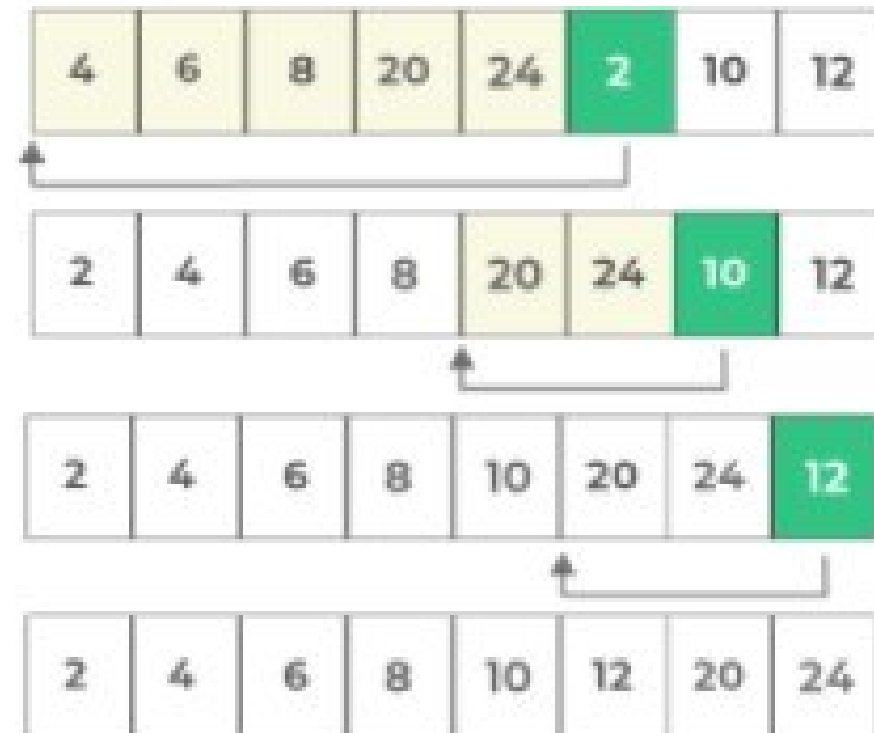
Iteración 1



Iteración 2

Métodos de ordenamiento

Insertion Sort

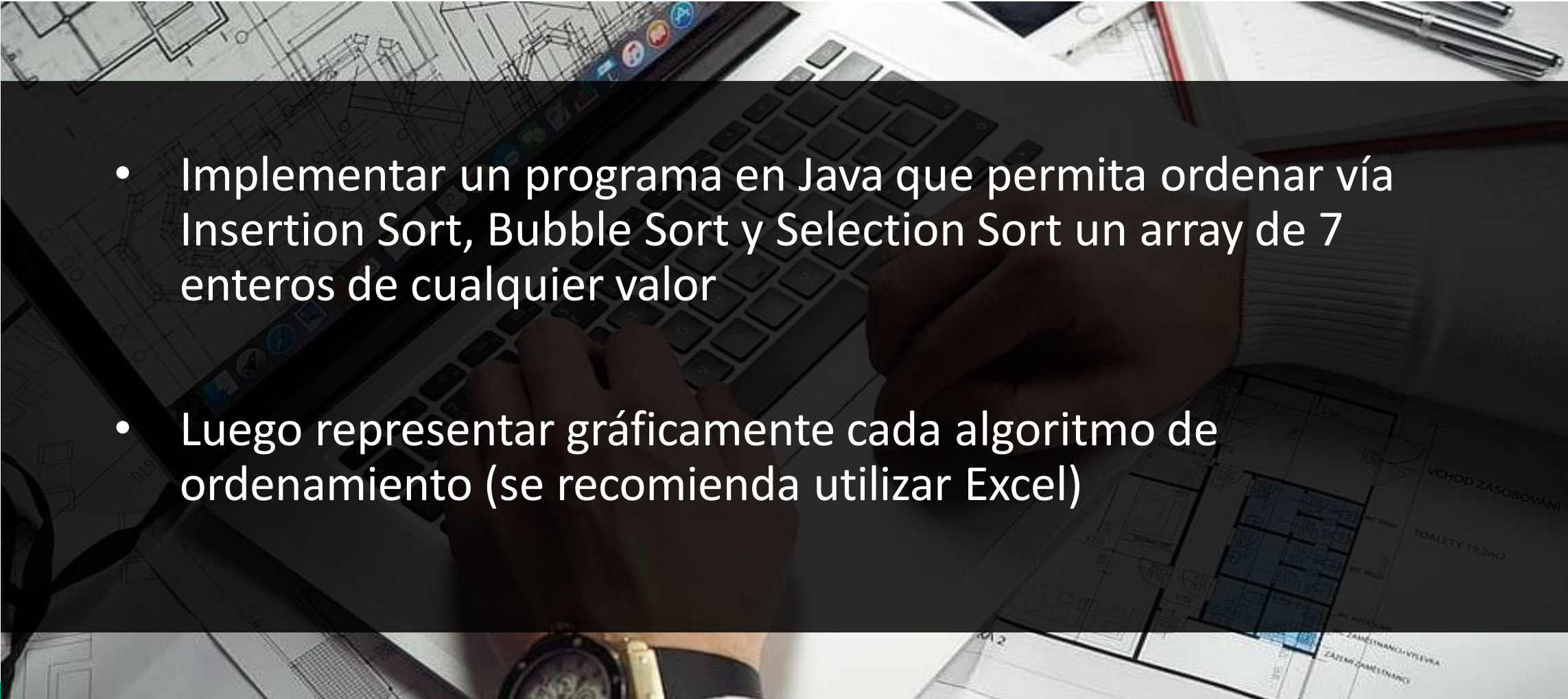


Métodos de ordenamiento

Insertion Sort en Java

```
public static int[] sort(int[] arreglo){  
    int i=1;  
    while (i < arreglo.length){  
        int j=i;  
        while (j > 0 && arreglo[j-1] > arreglo[j]){  
            swap(arreglo, j, j-1);  
            j -= 1;  
        }  
        i +=1;  
    }  
    return arreglo;  
}
```

Practiquemos

- 
- Implementar un programa en Java que permita ordenar vía Insertion Sort, Bubble Sort y Selection Sort un array de 7 enteros de cualquier valor
 - Luego representar gráficamente cada algoritmo de ordenamiento (se recomienda utilizar Excel)

Tarea

- Dado un arreglo de 20 elementos numéricos, representar gráficamente el proceso swap de todos los métodos de ordenamiento vistos en clase.

Consultas



¿Que hemos aprendido hoy?



- ¿Para que sirve notación Big O?
- ¿Qué es un algoritmo de ordenamiento?
- ¿Cuáles son las ventajas del método insertion sort, bubble sort y selection sort?

Bibliografía

- Tanenbaum & Van Steen (2008). Algoritmos y Estructuras de Datos - Principios y Paradigmas, 2da Edición. Pearson Education
- Khalid A. Mughal & Rolf W. Rasmussen (2017). A Programmer's guide to Java SE 8 Oracle Certified Associate



**Universidad
Tecnológica
del Perú**