



UNIVERSIDAD TECNOLÓGICA DE PANAMÁ CENTRO REGIONAL DE  
CHIRIQUÍ  
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES



**CARRERA:**

Gestión y Desarrollo de Software

**ACTIVIDAD No. 16**

**LABORATORIO No. 9**

**“Laboratorio 9”**

**ASIGNATURA:** Estructura de Datos II

**DOCENTE:**

Profa. Nunehar Mondul

**ESTUDIANTE/s:**

Jorge Jiménez (4-826-874)

-

-

*I SEMESTRE 2025*

**FECHA:**

06/24/2025

## Desarrollo

1. "Ordenar la siguiente lista por el método de inserción, burbuja y de árbol binario."

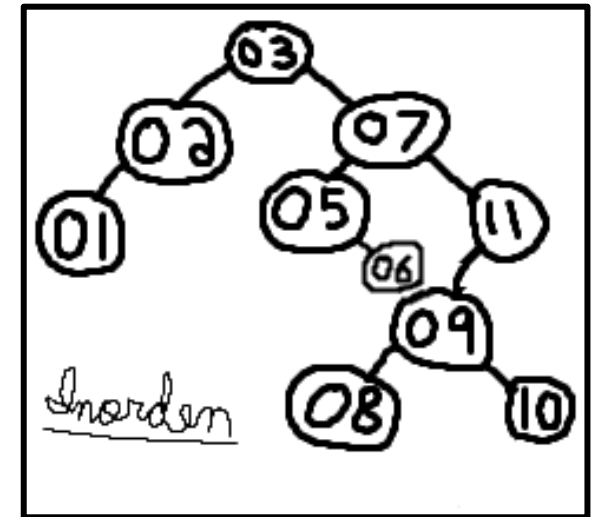
Inserción									
03	07	11	02	09	01	08	05	10	06
03	07	11	02	09	01	08	05	10	06
03	07	11	02	09	01	08	05	10	06
03	07	11	02	09	01	08	05	10	06
02	03	07	11	09	01	08	05	10	06
02	03	07	09	11	01	08	05	10	06
01	02	03	07	09	11	08	05	10	06
01	02	03	07	08	09	11	05	10	06
01	02	03	05	07	08	09	11	10	06
01	02	03	05	07	08	09	10	11	06
01	02	03	05	06	07	08	09	10	11

### Burbuja:

03-07-11-02-09-01-08-05-10-06  
 03-07-02-11-09-01-08-05-10-06  
 03-07-02-09-11-01-08-05-10-06  
 03-07-02-09-01-11-08-05-10-06  
 03-07-02-09-01-08-11-05-10-06  
 03-07-02-09-01-08-05-11-10-06  
 03-07-02-09-01-08-05-10-11-06  
 03-07-02-09-01-08-05-10-06-11  
 03-02-07-01-09-08-05-10-06-11  
 03-02-07-01-08-09-05-10-06-11  
 03-02-07-01-08-05-09-10-06-11  
 03-02-07-01-08-05-09-06-10-11  
 02-03-01-07-05-08-09-06-10-11  
 02-03-01-07-05-08-06-09-10-11  
 02-01-03-07-05-06-08-09-10-11  
 01-02-03-05-07-06-08-09-10-11  
 01-02-03-05-06-07-08-09-10-11

### Árbol Binario:

Recorrido in-order (izquierda, raíz, derecha):  
 01, 02, 03, 05, 06, 07, 08, 09, 10, 11



2. "¿Cuál fue la diferencia en los tiempos de ordenamiento entre un método y otro?"

- Inserción: Rápido para listas pequeñas o casi ordenadas.
- Burbuja: Más lento; realiza muchas comparaciones innecesarias.
- Árbol binario: Más eficiente en promedio ( $O(n \log n)$ ), especialmente con estructuras balanceadas.

### 3. “¿Por qué es necesario ordenar antes de aplicar el método binario? ¿Como sería el código en java?”

-Bueno... si recuerdo bien, no es necesario ordenar antes de usar un árbol binario, ya que el recorrido in-order del árbol binario ordena automáticamente.

Código en java:

```
OrdenamientoComparado.java x ArbolBinario.java x
Source History
1 package binario;
2 class Nodo {
3     int valor;
4     Nodo izquierdo, derecho;
5
6     Nodo(int valor) {
7         this.valor = valor;
8         izquierdo = derecho = null;
9     }
10 }
11
12 public class ArbolBinario { // es lo mismo que hicimos hace unas clases
13     Nodo raiz;
14
15     void insertar(int valor) {
16         raiz = insertarRec(raiz, valor);
17     }
18
19     Nodo insertarRec(Nodo raiz, int valor) {
20         if (raiz == null) return new Nodo(valor);
21         if (valor < raiz.valor) raiz.izquierdo = insertarRec(raiz.izquierdo, valor);
22         else raiz.derecho = insertarRec(raiz.derecho, valor);
23         return raiz;
24     }
25
26     void inOrden(Nodo nodo) {
27         if (nodo != null) {
```

```
28             inOrden(nodo.izquierdo);
29             System.out.print(nodo.valor + " ");
30             inOrden(nodo.derecho);
31         }
32     }
33
34     public static void main(String[] args) {
35         ArbolBinario arbol = new ArbolBinario();
36         int[] datos = {3, 7, 11, 2, 9, 1, 8, 5, 10, 6}; // Le puse el que
37         for (int n : datos) arbol.insertar(n); // estaba en el documento
38         arbol.inOrden(arbol.raiz); // Imprime lista ordenada
39     }
40 }
```

```
Output - binario (run) x
run:
1 2 3 5 6 7 8 9 10 11 BUILD SUCCESSFUL (total time: 0 seconds)
```

### 4. “¿Por qué son más eficientes los array que las listas para implementación de algoritmos de ordenamiento?”

- Los arrays permiten acceso directo a cualquier elemento con índice, lo que hace más rápidos los algoritmos de ordenamiento.
- Las listas enlazadas requieren recorrer nodos para acceder a un valor, lo que incrementa el tiempo.

## 5. "Cambiar el siguiente código, por otro algoritmo de ordenamiento y documentar."

```
OrdenamientoComparado.java x
Source History
1 package ordenamiento;
2
3 public class OrdenamientoComparado {
4     // Nota: en vez de cambiarlo por el ordenamiento nuevo, decidi dejar el bubble
5     // y añadir el nuevo... solo para así tambien tener el bubble para comparar
6
7     // Método de Burbuja
8     public static void bubbleSort(int[] array) {
9         int n = array.length;
10        boolean swapped;
11
12        for (int i = 0; i < n - 1; i++) {
13            swapped = false;
14
15            for (int j = 0; j < n - i - 1; j++) {
16                if (array[j] > array[j + 1]) {
17                    int temp = array[j];
18                    array[j] = array[j + 1];
19                    array[j + 1] = temp;
20                    swapped = true;
21                }
22            }
23
24            if (!swapped) {
25                break;
26            }
27        }
28    }
29
30    // Método de Inserción
31    public static void insertionSort(int[] array) {
32        int n = array.length;
33
34        for (int i = 1; i < n; i++) {
35            int key = array[i];
36            int j = i - 1;
37
38            // Mover los elementos mayores que key a una posición adelante
39            while (j >= 0 && array[j] > key) {
40                array[j + 1] = array[j];
41                j = j - 1;
42            }
43            array[j + 1] = key;
44        }
45    }
46
47    // Método para imprimir cualquier arreglo
48    public static void imprimirArreglo(String titulo, int[] array) {
49        System.out.println(titulo);
50        for (int num : array) {
51            System.out.print(num + " ");
52        }
53        System.out.println("\n");
54    }
55
56    public static void main(String[] args) {
57        int[] original = {64, 34, 25, 12, 22, 11, 90};
58
59        // Copias del arreglo original para cada algoritmo
60        int[] copiaBurbuja = original.clone();
61        int[] copiaInsercion = original.clone();
62
63        imprimirArreglo("Arreglo original:", original);
64
65        // Ordenar con Burbuja
66        bubbleSort(copiaBurbuja);
67        imprimirArreglo("Arreglo ordenado con Bubble Sort:", copiaBurbuja);
68
69        // Ordenar con Inserción
70        insertionSort(copiaInsercion);
71        imprimirArreglo("Arreglo ordenado con Insertion Sort:", copiaInsercion);
72    }
73 }
```

```
28 }
29
30 // Método de Inserción
31 public static void insertionSort(int[] array) {
32     int n = array.length;
33
34     for (int i = 1; i < n; i++) {
35         int key = array[i];
36         int j = i - 1;
37
38         // Mover los elementos mayores que key a una posición adelante
39         while (j >= 0 && array[j] > key) {
40             array[j + 1] = array[j];
41             j = j - 1;
42         }
43         array[j + 1] = key;
44     }
45 }
46
47 // Método para imprimir cualquier arreglo
48 public static void imprimirArreglo(String titulo, int[] array) {
49     System.out.println(titulo);
50     for (int num : array) {
51         System.out.print(num + " ");
52     }
53     System.out.println("\n");
54 }
```

```
55
56 public static void main(String[] args) {
57     int[] original = {64, 34, 25, 12, 22, 11, 90};
58
59     // Copias del arreglo original para cada algoritmo
60     int[] copiaBurbuja = original.clone();
61     int[] copiaInsercion = original.clone();
62
63     imprimirArreglo("Arreglo original:", original);
64
65     // Ordenar con Burbuja
66     bubbleSort(copiaBurbuja);
67     imprimirArreglo("Arreglo ordenado con Bubble Sort:", copiaBurbuja);
68
69     // Ordenar con Inserción
70     insertionSort(copiaInsercion);
71     imprimirArreglo("Arreglo ordenado con Insertion Sort:", copiaInsercion);
72 }
73 }
```

```
Output - ordenamiento (run) x
run:
Arreglo original:
64 34 25 12 22 11 90

Arreglo ordenado con Bubble Sort:
11 12 22 25 34 64 90

Arreglo ordenado con Insertion Sort:
11 12 22 25 34 64 90
```