RESULTADO DE ALGORITMOS DE BUSQUEDA Y ORDENAMIENTOS

Reporte de resultados después de pruebas con algoritmos de búsqueda y ordenamientos

Elaborado por: Diego Chevez

Repositorio: Búsqueda y Ordenamientos en Java

Índice

1. Introducción

En este proyecto se implementaron algoritmos de búsqueda (secuencial y binaria) y de ordenamiento (Bubble Sort, Insertion Sort y Selection Sort) utilizando Java, con el objetivo de comprender su funcionamiento, analizar sus tiempos de ejecución y comparar sus complejidades en distintos escenarios.

Se utilizó un dataset aleatorio de entre 5 y 20 elementos por ejecución, permitiendo pruebas variadas y realistas para el análisis de tiempos.

2. Pseudocódigo de algoritmos

En el siguiente apartado, se detalla de manera concisa la lógica detrás de cada algoritmo.

Algoritmo	Pseudocódigo resumido en español
Búsqueda Secuencial	- Recorrer cada elemento de la lista. - Si el elemento es igual al valor buscado, retornar índice. - Si no se encuentra, retornar "No encontrado".
Búsqueda Binaria	- Ordenar la lista. - Establecer bajo = 0, alto = longitud - 1. - Mientras bajo ≤ alto: → medio = (bajo + alto) / 2. → Si lista[medio] == valor: retornar índice. → Si lista[medio] < valor: bajo = medio + 1. → Si lista[medio] > valor: alto = medio - 1. - Si no se encuentra, retornar "No encontrado".
Bubble Sort	- Para i de 0 a n-1: → Para j de 0 a n-i-1: → Si lista[j] > lista[j+1]: intercambiar lista[j] con lista[j+1].
Insertion Sort	- Para i de 1 a n-1: → key = lista[i]. → j = i - 1. → Mientras j ≥ 0 y lista[j] > key: → lista[j+1] = lista[j]. → j = j - 1. → lista[j+1] = key.
Selection Sort	- Para i de 0 a n-1: → min_idx = i. → Para j de i+1 a n: → Si lista[j] < lista[min_idx]: min_idx = j. → Intercambiar lista[i] con lista[min_idx].

By: Diego Chevez

3. Pruebas realizadas

- Se generó automáticamente un dataset aleatorio (5 a 20 elementos, valores 1-100) en cada ejecución.
- Se realizaron búsquedas secuenciales y binarias, seleccionando distintos valores (algunos existentes y otros no) para comprobar tiempos y resultados.
- Se aplicaron los tres algoritmos de ordenamiento, verificando el estado del dataset antes y después de ordenar.
- Se utilizó System.nanoTime() para capturar el tiempo de ejecución de cada operación.
- Se revisó el historial de operaciones para comparar los tiempos y complejidades.

Ejemplos

```
========= SEQUENTIAL SEARCH =========
Available values to search: [87, 81, 87, 8, 60, 78, 41, 8, 6, 13, 9, 68, 84, 67, 86] (Length: 15)
Enter the number to search: 13
Found at index 9 | Execution time (ns): 1200
Complexity: Best O(1), Worst O(n), Avg O(n)
```

```
========= BINARY SEARCH =========
Available values to search (sorted): [6, 8, 8, 9, 13, 41, 60, 67, 68, 78, 81, 84, 86, 87, 87] (Length: 15)
Enter the number to search: 13
Found at index 4 | Execution time (ns): 1200
Complexity: Best O(1), Worst O(log n), Avg O(log n)
```

```
========= BUBBLE SORT =========
List before sorting: [87, 81, 87, 8, 60, 78, 41, 8, 6, 13, 9, 68, 84, 67, 86] (Length: 15)
List after sorting:  [6, 8, 8, 9, 13, 41, 60, 67, 68, 78, 81, 84, 86, 87, 87] (Length: 15)
Bubble Sort completed | Execution time (ns): 7000
Complexity: Best O(n), Worst O(n^2), Avg O(n^2)
```

```
========= INSERTION SORT =========
List before sorting: [87, 81, 87, 8, 60, 78, 41, 8, 6, 13, 9, 68, 84, 67, 86] (Length: 15)
List after sorting:  [6, 8, 8, 9, 13, 41, 60, 67, 68, 78, 81, 84, 86, 87, 87] (Length: 15)
Insertion Sort completed | Execution time (ns): 3400
Complexity: Best O(n), Worst O(n^2), Avg O(n^2)
```

```
========= SELECTION SORT =========
List before sorting: [87, 81, 87, 8, 60, 78, 41, 8, 6, 13, 9, 68, 84, 67, 86] (Length: 15)
List after sorting:  [6, 8, 8, 9, 13, 41, 60, 67, 68, 78, 81, 84, 86, 87, 87] (Length: 15)
Selection Sort completed | Execution time (ns): 11200
Complexity: Best O(n^2), Worst O(n^2), Avg O(n^2)
```

```
========= RECORD =========
Operation           Target     Time (ns)       Result                  Complexity
Sequential Search   13         1200            Found at index 9        Best O(1), Worst O(n), Avg O(n)
Binary Search       13         1200            Found at index 4        Best O(1), Worst O(log n), Avg O(log n)
Bubble Sort         -          7000            Sorted                  Best O(n), Worst O(n^2), Avg O(n^2)
Insertion Sort      -          3400            Sorted                  Best O(n), Worst O(n^2), Avg O(n^2)
Selection Sort      -          11200           Sorted                  Best O(n^2), Worst O(n^2), Avg O(n^2)
```


By: Diego Chevez

4. Resultados de tiempos y complejidades

Los siguientes resultados reflejan las operaciones realizadas durante las pruebas:

Operación	Elemento	Tiempo (ns)	Resultado	Complejidad
Sequential Search	13	1200	Found at index 9	Best O(1), Worst O(n), Avg O(n)
Binary Search	13	1200	Found at index 4	Best O(1), Worst O(log n), Avg O(log n)
Bubble Sort	-	7000	Sorted	Best O(n), Worst O(n^2), Avg O(n^2)
Insertion Sort	-	3400	Sorted	Best O(n), Worst O(n^2), Avg O(n^2)
Selection Sort	-	11200	Sorted	Best O(n^2), Worst O(n^2), Avg O(n^2)

5. Conclusiones

- Comprensión de algoritmos: La práctica permitió comprender el funcionamiento interno de algoritmos de búsqueda y ordenamiento, visualizando paso a paso cómo recorren y reorganizan los datos.
- Análisis de tiempos: Los tiempos de ejecución variaron según la operación y la cantidad de elementos, demostrando la diferencia entre complejidades lineales, logarítmicas y cuadráticas.
- Buenas prácticas en Java: Se reforzó la aplicación de POO, principios SOLID y estructuras modulares, facilitando la mantenibilidad y escalabilidad del proyecto.
- Aprendizaje práctico: Los algoritmos de ordenamiento mostraron diferencias en tiempos dependiendo del estado inicial del dataset (mejor caso, peor caso y caso promedio), confirmando la teoría estudiada.

By: Diego Chevez