**Taller № 3 - Grupal - Complejidad Algorítmica**

**Federico Vargas Rozo**

**Tabla 1 - Algoritmo de Selección - Lenguaje C**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tamaño de la entrada** | **#Comparaciones** | **#Intercambios** | **Estable / Inestable** | **Inserción (SI/NO)** |
| n = 1.000 | *499500* | *989* | *Inestable* | *No* |
| n = 1.500 | *1124250* | *1493* | *Inestable* | *No* |
| n = 2.000 | *1999000* | *1988* | *Inestable* | *No* |
| n = 2.500 | *3123750* | *2486* | *Inestable* | *No* |
| n = 3.000 | *4498500* | *2986* | *Inestable* | *No* |

**Tabla 2 - Complejidad Temporal - Algoritmo de Selección**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Tiempo de Ejecución (ns)** | | | |
| **Tamaño de la entrada** | **Java** | **Python** | **Lenguaje C** | **Go** |
| n = 1.000 | *2020400* | *21579100* | *991300* | *514300* |
| n = 1.500 | *3539900* | *50190000* | *2241800* | *1061400* |
| n = 2.000 | *5381600* | *91666600* | *3735800* | *1565400* |
| n = 2.500 | *6997200* | *159316600* | *5667400* | *2848800* |
| n = 3.000 | *7526200* | *224421000* | *8119300* | *4302100* |

**Gráficas – Complejidad Temporal – Algoritmo de Selección**

**Gráfico, Gráfico de líneas

Descripción generada automáticamente**

**Gráfico, Gráfico de líneas

Descripción generada automáticamente**

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

**Análisis – Algoritmo de Selección**

**Mejor Caso:** El mejor caso ocurre cuando la entrada (el arreglo) ya está ordenado. En este caso, el algoritmo realiza la mínima cantidad de comparaciones e intercambios. La complejidad temporal del Algoritmo de Selección es **O(n^2)**.

**Peor Caso:** El peor caso ocurre cuando la entrada (el arreglo) está ordenado al revés (sea ascendente o descendientemente, dependiendo si el algoritmo ordena asc. o desc.). En este caso, el algoritmo realiza la máxima cantidad de comparaciones e intercambios. La complejidad temporal del Algoritmo de Selección es **O(n^2)**.

**Complejidad Temporal:** Para todos los casos, la complejidad temporal del Algoritmo de Selección es **O(n^2)**. Esto es porque el algoritmo realiza un bucle anidado que compara cada elemento con todos los demás elementos, resultando en un número cuadrático de comparaciones e intercambios.

**Complejidad Espacial:** El Algoritmo de Selección tiene una complejidad espacial de **O(1)**, pues utiliza una cantidad constante adicional de memoria para ordenar el arreglo. Esto es porque el Algoritmo de Selección es un algoritmo de “intercambio en el lugar”, lo que significa que el algoritmo ordena intercambiando los elementos de lugar, sin utilizar más espacio en otras estructuras de datos como arreglos o listas. Esto lo hace un algoritmo eficiente respecto al uso de memoria, especialmente en arreglos grandes que pueden no caber en la memoria.

**Salida – Algoritmo de Selección (Entrada n = 3000)**

Texto

Descripción generada automáticamente

*Se evidencia salida para un arreglo de tamaño 3000, con el Lenguaje C, en el IDE JetBrains CLion.*

**Conclusiones – Algoritmo de Selección**

El Algoritmo de Selección es un algoritmo de ordenación simple que funciona buscando repetidamente el elemento mínimo de la parte no ordenada del arreglo y colocándolo al principio. Es un algoritmo de ordenación en el lugar, lo que significa que no requiere ningún espacio de almacenamiento adicional proporcional al tamaño de la matriz de entrada.

El Algoritmo de Selección es un algoritmo de ordenación inestable, lo que significa que el orden relativo de los elementos iguales puede cambiar durante el proceso de ordenación. Esto puede ser una desventaja en algunas aplicaciones, donde el orden original de los elementos iguales es importante.

La complejidad temporal del Algoritmo de Selección es **O(n^2)** en todos los casos, donde n es el tamaño del arreglo de entrada. Esto se debe a que el algoritmo realiza un bucle anidado que compara cada elemento con todos los demás elementos, lo que da como resultado una cantidad cuadrática de comparaciones e intercambios. Esto hace que el algoritmo de ordenación por selección sea ineficiente para arreglos grandes, especialmente en comparación con algoritmos de ordenación más avanzados como *Quicksort* o *Mergesort*.

**Bibliografía y Referencias**

1. Knuth Donald. *The Art of Computer Programming, vol 3: Sorting and Searching, 3rd ed.* (1997). ISBN 0-201-89685-0. Pg. 138-141, Sección 5.2.3: Sorting by Selection.
2. Sedgewick Robert. *Algorithms in C++, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching: Fundamentals. 2nd ed.* (1998). ISBN 0-201-35088-2. Pg. 273-274.
3. Shiksha Online. *Time and Space Complexity of Sorting Algorithms.* (Jul 2024). [En Linea]. Disponible: <https://shorturl.at/9VTEU>
4. GeeksForGeeks. *Time and Space Complexity analysis of Selection Sort.* (May 2024). [En Linea]. Disponible: <https://shorturl.at/8P6vg>
5. Medium. *Time & Space Complexity in Sorting Algorithms / Software Interview.* (Feb 2024). [En Linea]. Disponible: <https://shorturl.at/FUnOE>