UNIVERSIDAD DEL ROSARIO
MATEMÁTICAS APLICADAS Y CIENCIAS DE LA COMPUTACIÓN
ALGORITMOS Y ESTRUCTURAS DE DATOS

PROYECTO FINAL

DOCUMENTO FINAL

ANÁLISIS DE ALGORITMOS DE ORDENAMIENTO, A PARTIR DE UNA CLASE DE VECTORES

MANUELA ACOSTA FAJARDO
MARÍA PAULA GAVIRIA

REPOSITORIO: manuacostaf18/Proyecto-Algoritmos

I. RESUMEN EJECUTIVO

I.I Descripción del problema identificado

Queremos demostrar, de manera gráfica, el funcionamiento de diez diferentes algoritmos de ordenamiento, trabajando con vectores de números enteros, de forma que se evidencie cuál de todos es más efectivo y más rápido en cada situación específica.

I.II Solución propuesta

Para determinar cuál de los algoritmos seleccionados es más eficiente, analizamos uno a uno, entendimos su funcionamiento, realizamos una serie de pruebas en las que se calculaba el tiempo de ejecución de los mismos y comparamos los resultados.

I.III Resumen del trabajo desarrollado

Inicialmente, seleccionamos los diez algoritmos de ordenamiento con los que trabajaríamos. Después de esto, creamos nuestra propia estructura de datos, una clase de vectores, con sus propios métodos y atributos, la cual usaríamos para probar los algoritmos. Nuestro siguiente paso a seguir fue adaptar los algoritmos escogidos a nuestra clase, de forma que funcionaran con vectores creados por medio de la misma. Además, analizamos el funcionamiento de los algoritmos, llegando a una breve explicación de cada uno de ellos. Posteriormente, hicimos uso de nuestro método ‘display’ dentro del código de los algoritmos, de forma que fuera visible la manera y el paso a paso en que éstos ordenan los vectores. Después de esto, ejecutamos cada uno de los algoritmos con vectores ordenados ascendentemente, descendentemente, y aleatorios, de diversos tamaños, calculando el tiempo en cada caso, para después promediar y graficar. Ya con las gráficas, siendo más visible los tiempos de ejecución de los algoritmos en cada caso, hicimos un análisis detallado de éstos, llegando a la conclusión de cuál de los

algoritmos de ordenamiento seleccionados es más eficiente en cada uno de los casos.

II. FUNCIONALIDAD Y DESCRIPCIÓN DE LA HERRAMIENTA

II.I Clase de vectores

Esta estructura de datos, una clase que genera vectores a partir de arreglos dinámicos, nos permite crear los vectores necesarios para realizar las pruebas de los algoritmos seleccionados. Métodos como ‘getsize’, ‘getfront’, ‘getmin’, ‘getmax’, ‘set’, entre otros, nos permiten obtener datos e información referente al vector creado, los cuales sirven como argumentos de algunos de los algoritmos. Además, el método ‘display’ nos permite evidenciar gráficamente cómo se encuentra organizado el vector, y los cambios que en él se hacen.

II.II Algoritmos de ordenamiento

Los algoritmos de ordenamiento, en nuestra opinión, son la herramienta fundamental del proyecto, ya que en base a ellos se centra todo nuestro análisis. Ellos fueron utilizados para ordenar vectores de diferentes datos y tamaños, obteniendo su tiempo de ejecución, entendiendo a fondo la forma en que trabajan, y analizando, así, cuál es más eficiente, en cuanto a tiempo, en situaciones específicas.

II.III Funciones adicionales

Además de las herramientas anteriormente mencionadas, hicimos uso de algunas funciones adicionales, como ‘swap’, o ‘gettime’, que nos permitieron hacer cambios de elementos dentro de los vectores, al momento de ejecutar los algoritmos, u obtener el tiempo de ejecución de los mismos, respectivamente. Gracias al uso de dichas funciones, fue posible analizar a fondo los algoritmos seleccionados.

III. ALGORITMOS Y ESTRUCTURAS DE DATOS

III.I Clase de vector - Estructura de datos

Como se mencionó anteriormente, creamos nuestra propia estructura de datos, una clase de vectores, por medio de arreglos, para obtener nuestros propios vectores, con cualidades necesarias para probar los algoritmos seleccionados.

III.II Algoritmos de ordenamiento

Para el estudio de los algoritmos de ordenamiento, seleccionamos los siguientes diez algoritmos. La explicación detallada y el código de cada uno de ellos se encuentra en las diapositivas.

1. Heap Sort
2. Counting Sort
3. Selection Sort
4. Bubble Sort
5. Insertion Sort
6. Quick Sort
7. Merge Sort
8. Shell Sort
9. Gnome Sort
10. Cocktail Sort

IV. ANÁLISIS DE ALGORITMOS

La eficiencia de los algoritmos, a la cual hacemos referencia en los siguientes párrafos, es tomada en cuanto al tiempo que éstos toman ordenando un vector. Es decir, el algoritmo más eficiente es aquel que toma menos tiempo en ordenarlo, y el menos eficiente, el que más tiempo toma.

IV.I Vectores desordenados

Haciendo un análisis detallado de los resultados de ejecución de todos los algoritmos al ser implementados con vectores desordenados, observamos que se presentan tres escenarios: (1) algunos algoritmos disminuyen su eficiencia a medida que aumenta el tamaño de los vectores, (2) otros aumentan su eficiencia a medida que aumenta el tamaño de los vectores, y (3) unos pocos se mantienen constantes. En el primer caso se puede presenciar este cambio en los algoritmos Gnome, Cocktail, Selection, Bubble e Insertion; de los cuales Bubble y Cocktail se posicionan con la menor eficiencia a medida que aumenta el tamaño. En el segundo caso se puede presenciar el cambio en los algoritmos Counting, Heap, Shell y Merge; de los cuales los que más representan este cambio son el Counting y Heap, donde el último logra estar como el tercer mejor algoritmo. Por último, en el último caso se encuentra Quick, se mantiene en primer y segundo lugar, mostrando independencia del tamaño para tener una buena eficiencia.

	VECTORES DESORDENADOS				
	VEC 5	**VEC 10**	**VEC 50**	**VEC 100**	**VEC 1000**
1	Gnome	Insertion	Shell	Quick	Quick
2	Quick	Quick	Quick	Shell	Heap
3	Cocktail	Shell	Heap	Heap	Shell
4	Shell	Gnome	Insertion	Counting	Counting
5	Selection	Selection	Gnome	Merge	Merge
6	Bubble	Heap	Selection	Insertion	Gnome
7	Heap	Cocktail	Cocktail	Gnome	Selection
8	Insertion	Bubble	Merge	Selection	Insertion
9	Merge	Merge	Counting	Cocktail	Cocktail
10	Counting	Counting	Bubble	Bubble	Bubble

IV.II Vectores invertidos

Teniendo en cuenta los mismos posibles escenarios en los vectores desordenados, en el análisis de los invertidos (de mayor a menor) llegamos a unos resultados similares. En el primer caso, en donde la eficiencia disminuye al aumentar el tamaño del vector, los algoritmos que presentan esta caracteristica son Insertion, Cocktail, Quick y Bubble. En el segundo caso, donde la eficiencia aumenta al aumentar el tamaño de los vectores, se encuentran los algoritmos Heap, Merge y Counting; y en los tres es muy notorio su poca eficiencia en vectores de tamaño pequeño pero luego son muy eficientes en los vectores de mayor tamaño. En el tercer caso, se pueden encontrar los algoritmos Shell y Selection, en donde en especial el primero muestra una constante eficiencia, siendo este el mejor algoritmo de ordenamiento para vectores cuyos datos están en orden descendente. Por otro lado está el algoritmo Gnome, el cual no presenta alguna característica concreta de estos tres posibles casos ya que siempre está cambiando su comportamiento.

	VECTORES INVERTIDOS				
	VEC 5	**VEC 10**	**VEC 50**	**VEC 100**	**VEC 1000**
1	Shell	Shell	Shell	Shell	Shell
2	Insertion	Quick	Heap	Heap	Heap
3	Cocktail	Selection	Selection	Selection	Merge
4	Selection	Insertion	Quick	Merge	Counting
5	Quick	Gnome	Insertion	Quick	Gnome
6	Gnome	Heap	Merge	Counting	Quick
7	Heap	Cocktail	Gnome	Insertion	Cocktail
8	Bubble	Bubble	Cocktail	Gnome	Selection
9	Merge	Merge	Counting	Cocktail	Bubble
10	Counting	Counting	Bubble	Bubble	Insertion

IV.III Vectores ordenados

En el caso de los vectores con elementos ordenados en orden ascendente, a pesar de que parecería poco útil utilizar los algoritmos para ordenarlos de nuevo, queríamos analizar cuánto tiempo tardaban éstos en recorrer el vector, y analizar que efectivamente estaban ordenados.

En este caso obtuvimos resultados interesantes, ya que los algoritmos se dividen en dos grupos: Gnome, Insertion, Cocktail, Bubble y Shell siempre están en las primeras cinco posiciones y Selection, Quick, Heap, Merge y Counting en las otras cinco. Además, dentro de estos dos grupos se presentan los casos que hemos

detectado en los otros dos análisis. En el primer grupo, Gnome y Shell se mantienen constantes, donde el primero siempre ocupa el primer lugar, evidenciando que es el mejor algoritmos para verificar que el vector efectivamente está ordenado. Por otro lado Cocktail aumenta su eficiencia, e Insertion y Bubble se van intercalando. En el segundo grupo, Heap, Merge y Counting aumentan su eficiencia a medida que aumenta el tamaño, pero al estar en el segundo grupo no llegan a ser tan eficientes, como algunos que están en el primer grupo. Contrario a esto, quedan Selection y Quick disminuyen su eficiencia, evidenciando que definitivamente estos no son los algoritmos ideales para verificar que el vector esté en orden.

	VECTORES ORDENADOS				
	VEC 5	**VEC 10**	**VEC 50**	**VEC 100**	**VEC 1000**
1	Gnome	Gnome	Gnome	Gnome	Gnome
2	Insertion	Cocktail	Cocktail	Cocktail	Cocktail
3	Cocktail	Insertion	Bubble	Insertion	Bubble
4	Bubble	Bubble	Insertion	Bubble	Insertion
5	Shell	Shell	Shell	Shell	Shell
6	Selection	Selection	Heap	Heap	Heap
7	Quick	Heap	Selection	Merge	Merge
8	Heap	Quick	Merge	Selection	Counting
9	Merge	Merge	Quick	Counting	Selection
10	Counting	Counting	Counting	Quick	Quick

Nota: En el repositorio hay una carpeta llamada “Gráficas”, en donde se encuentran 15 distintas gráficas que evidencia a mayor detalle la eficiencia de los algoritmos dependiendo del tamaño y el tipo de vector que se está ordenando.

V. CONCLUSIONES

Teniendo en cuenta los anteriores análisis y observaciones, podemos concluir nuestro análisis de los algoritmos de ordenamiento a partir de vectores de números enteros con las siguientes afirmaciones:

- No hay algoritmo de ordenamiento que se destaque más que otro, la eficiencia de cada uno de ellos depende de la funcionalidad que se está buscando en cada caso.
- Teniendo en cuenta nuestros resultados, podemos clasificar como los algoritmos más eficientes, en cada caso, a los siguientes:
    - Vectores desordenados: (1) Quick Sort, (2) HeapSort, y (3) Shell Sort.
    - Vectores invertidos: (1) Shell Sort, (2) Heap Sort, (3) Selection Sort.

- Vectores ordenados: (1) Gnome Sort, (2) Cocktail Sort, (3) Insertion Sort.

VI. REFERENCIAS

VI.I Weiss, Mark A. (2013). Data Structures and Algorithm Analysis in C++. Prentice Hall.
VI.II GeeksforGeeks | A computer science portal for geeks. (s.f.). Recuperado 25 noviembre, 2019, de https://www.geeksforgeeks.org/