Carlos Andrés Restrepo - César Leonardo Canales - Ana María Muñoz

Planeación y realización

Experimento

Los problemas algorítmicos que se presentan dentro de la ciencia de la computación tienen como uno de sus enfoques más importantes encontrar la solución más eficiente. Por esto, es posible encontrar dentro de uno de los problemas fundamentales, el de ordenamiento, distintas maneras de llegar al objetivo principal que es convertir unos valores recibidos como entradas a unas salidas completamente ordenadas, de la forma:

Entrada: una secuencia de n números $< a_1, a_2, a_3, ...a_n >$

Salida: una permutación $< a_{1'}, a_{2'}, a_{3'}, ...a_{n'} >$ de la secuencia de entrada tal que $a_{1'} \leq a_{2'} \leq a_{3'} \leq ... \leq a_{n'}$.

De esta manera, los programadores siempre han tenido que enfrentarse a decidir sobre una serie de factores que afectan directamente la búsqueda de la solución más rápida y efectiva para la situación que se quiera resolver. A partir de esta problemática, el principal procedimiento a tener en cuenta para escoger el algoritmo ideal que resolverá el problema es el análisis de complejidad temporal de esos algoritmos, "debido a que el comportamiento de un algoritmo puede ser diferente para cada una de las posibles entradas, se necesitan medios para resumir este comportamiento en fórmulas simples y fáciles de entender" (Salas y Rodríguez, 2013, p.24).

A partir de lo anterior, se destaca que es clave para llegar a encontrar la solución con mejor eficiencia, tener en cuenta los elementos que influyen para la ejecución y que el desarrollador debe analizar cuáles son las características de la situación problemática en la que se encuentra y los recursos con los que cuenta para desarrollarlo puesto que, como describen Gómez y Cervantes (2014), el tiempo de ejecución de un algoritmo depende de la cantidad de datos de entrada, de la forma en que se implementará el programa, también del procesador del computador y finalmente de la complejidad del algoritmo.

Finalmente, el problema principal a trabajar es encontrar la solución más eficiente y eficaz que resuelva un problema de ordenamiento teniendo distintos tamaños de entrada, diferente

capacidad de RAM y el estado de los números a estudiar que pueden ser: ascendente, descendente y aleatoriamente. De lo anterior, es importante realizar un análisis exhaustivo para los algoritmos de ordenamiento que se implementarán: HeapSort y MergeSort; puesto que, permitirá llegar a una decisión final sobre el algoritmo más efectivo y también, será posible concluir cómo los distintos factores escogidos para el experimento pueden afectar el tiempo de ejecución del programa. Para así, resaltar que la importancia de este problema radica en que es posible establecer límites y características para llevar a cabo la solución de un problema a partir de los recursos que se utilizarán y, de esta manera, definir cuál es la mejor estrategia a realizar, conociendo que se pueden obtener las salidas esperadas de distintos tiempos y a partir de estos resultados, escoger el más eficiente.

Unidad experimental

Para llevar a cabo el experimento se define la unidad experimental, la cual permite generar un valor que sea representativo en el proceso de experimentación. La unidad experimental para este caso son las entradas que recibirán los algoritmos de ordenamiento, pues estas entradas son como muestras que al usarse en dichos algoritmos generarán valores que serán importantes para el análisis del experimento.

Definición de factores

- **Factores controlables**
    - Algoritmo de Ordenamiento: para la realización del experimento se usarán dos algoritmos de ordenamiento bastantes conocidos: Heapsort y Merge Sort.
    - Tamaño del arreglo: Se usarán entradas con diferente tamaños, como por ejemplo arreglos con 10^1, 10^2, 10^3, etc.
    - Estado de los valores en el arreglo: Los valores de los arreglos que se usarán como entrada a los algoritmos tendrán diferentes estados (orden descendente, orden aleatorio, entre otros).

Carlos Andrés Restrepo - César Leonardo Canales - Ana María Muñoz

  - Tamaño memoria RAM: Se ejecutarán los algoritmos con las distintas entradas y valores de los datos en diferentes computadores que se diferencien por el tamaño de RAM que tenga cada uno, en este caso, serán 2 distintos: 8 GB y 16 GB.

- **Factores no controlables**
  - Cantidad de procesos que se lleven a cabo dentro de cada uno de los computadores en los cuales se ejecutarán los algoritmos.
  - La generación aleatoria de los números que componen al array que se crea para realizar las pruebas pues el algoritmo de ordenamiento utilizado puede tomar diferentes tiempos para ordenar dos secuencias de entradas del mismo tamaño, dependiendo de cuan parcialmente ordenados se encuentren.
- **Factores estudiados**
  - Algoritmos de ordenamiento
  - Tamaño del arreglo
  - Estado de los valores en el arreglo
  - Tamaño memoria RAM

Análisis

➔ **Análisis complejidad temporal algoritmo Merge Sort**

Asumiendo que se van a ordenar un total de n elementos en total, tenemos los siguientes tres pasos:

1. El paso de dividir toma tiempo constante ya que lo único que se hace en este es calcular el punto medio de los índices l y r, indicamos tiempo constante con O(1).
2. En el paso de conquistar ordenamos recursivamente los dos subarreglos de aproximadamente n/2 elementos cada uno, esto toma una cantidad de tiempo pero la tomaremos en cuenta en el momento que consideremos los subproblemas.
3. El paso de combinar combina un total de n elementos, haciendo que tenga una complejidad lineal, esto es, O(n).

Carlos Andrés Restrepo - César Leonardo Canales - Ana María Muñoz

Utilizando esta información, si el tiempo de ejecución del algoritmo en una lista de n elementos lo denotamos como T(n), entonces la recurrencia

$$T(n) = 2T(n/2) + n$$

Se sigue inmediatamente después de las definiciones dadas anteriormente, esta recurrencia se puede resolver directamente utilizando el teorema maestro, su solución es O(nlogn).

➔ **Análisis complejidad espacial algoritmo Merge Sort**

La complejidad espacial del Merge Sort siempre será O(n) incluyendo los arreglos, si se dibuja el árbol de espacio, pareciera que la complejidad es O(nlogn). Sin embargo, dado que el código es uno del tipo "Depth first code", siempre se expandirá en solo una rama del árbol, por tanto, el espacio total utilizado siempre tendrá como cota superior O(3n) = O(n).

➔ **Análisis complejidad temporal algoritmo HeapSort**

Utilizando el hecho que el algoritmo utiliza una estructura de datos como auxiliar en el ordenamiento de los datos, el cálculo de la complejidad temporal se facilita bastante. Primeramente, la construcción de un heap de n elementos toma O(nlogn) ya que este considera cada uno de los elementos y mediante la función de Heapify(ordenamiento de los elementos del heap de manera recursiva) hace que el elemento se posicione en la posición que le corresponde. Esto es hace un recorrido de los n elementos y en cada iteración de ese recorrido usa Heapify el cual tiene una complejidad de O(logn), por tanto la complejidad total de construir un heap es O(nlogn).

Ahora bien, ya teniendo construido el heap, el HeapSort consiste en aprovechar el hecho de que los elementos ya están ordenados en el árbol para iterar n veces y en cada iteración extraer la raíz(el menor elemento actual en el heap) y hacer un heapify para colocar los elementos, sin la raíz, nuevamente en su lugar. Lo dicho anteriormente, implica que el algoritmo tenga una complejidad de O(nlogn) ya que itera n veces extrayendo la raíz y luego haciendo un heapify en la misma iteración.

Carlos Andrés Restrepo - César Leonardo Canales - Ana María Muñoz

➔ Análisis complejidad espacial algoritmo HeapSort

Dado que el algoritmo utiliza el heap o montículo para su funcionamiento, y esta estructura de datos guarda n elementos en un árbol sin utilizar memoria adicional, la complejidad espacial del algoritmo es O(n).

➔ Análisis ANOVA SPSS

Durante el proceso de análisis, se realiza el ANOVA por medio de la plataforma SPSS, teniendo en cuenta que el factor a evaluar en este caso fue la cantidad de memoria RAM. A partir de la tabla, es posible descubrir el estadístico F con su nivel de significancia, lo que describe si la hipótesis en la cual se fundamenta ANOVA, la cual es la igualdad de medias entre los datos recolectados, se rechaza o se acepta. Siendo así, se evalúa el número decimal obtenido en la columna de significación (sig.), sabiendo que si se obtuvo un resultado menor o igual que 0.05 se rechaza la hipótesis de igualdad de medias.

ANOVA

tiempo_ejecución

	Suma de cuadrados	gl	Media cuadrática	F	Sig.
Entre grupos	792,067	1	792,067	,544	,464
Dentro de grupos	84484,267	58	1456,625		
Total	85276,333	59			

De esta manera, es importante precisar que el resultado obtenido del nivel de significancia teniendo como variable dependiente al tiempo de ejecución y como factor principal la cantidad de memoria RAM es de 0.464, se infiere que, a pesar de que este resultado lleva a aclarar que no existen diferencias significativas entre los grupos de datos debido a que supera el 0.05, también se puede interpretar que, al ser la F significativa, por lo menos, dos niveles del factor producen distintos efectos al dependiente.

De lo anterior, se interpreta que los datos recolectados siempre van a presentar resultados similares cuando se realiza un experimento de este tipo, como al realizar la prueba del algoritmo con tamaños de entradas pequeños como 10 o 100 datos numéricos. Es por esto que, al realizar el

análisis de varianza no se destaca mucho la diferencia que se presenta por el cambio de tamaño de memoria RAM en el que se ejecuta. Sin embargo, se encontrarán datos que permiten observar la diferencia de tiempo de ejecución siendo visible principalmente cuando el tamaño alcanza miles o cientos de miles:

HeapSort	100000	Orden aleatorio	8 GB	127
HeapSort	100000	Orden ascendente	8 GB	154
HeapSort	100000	Orden descendente	8 GB	172

En este caso, se analiza el algoritmo HeapSort ejecutado por medio de un computador con tamaño de memoria RAM de 8GB, teniendo unos tiempos de ejecución que difieren visiblemente de los obtenidos en el computador de RAM 16GB:

HeapSort	100000	Orden aleatorio	16 GB	95
HeapSort	100000	Orden ascendente	16 GB	90
HeapSort	100000	Orden descendente	16 GB	87

Así mismo, se presentan los resultados obtenidos con el algoritmo MergeSort cuando el tamaño de entrada es significativo como en el anterior caso, comparando cuando se ejecuta con el factor 8GB:

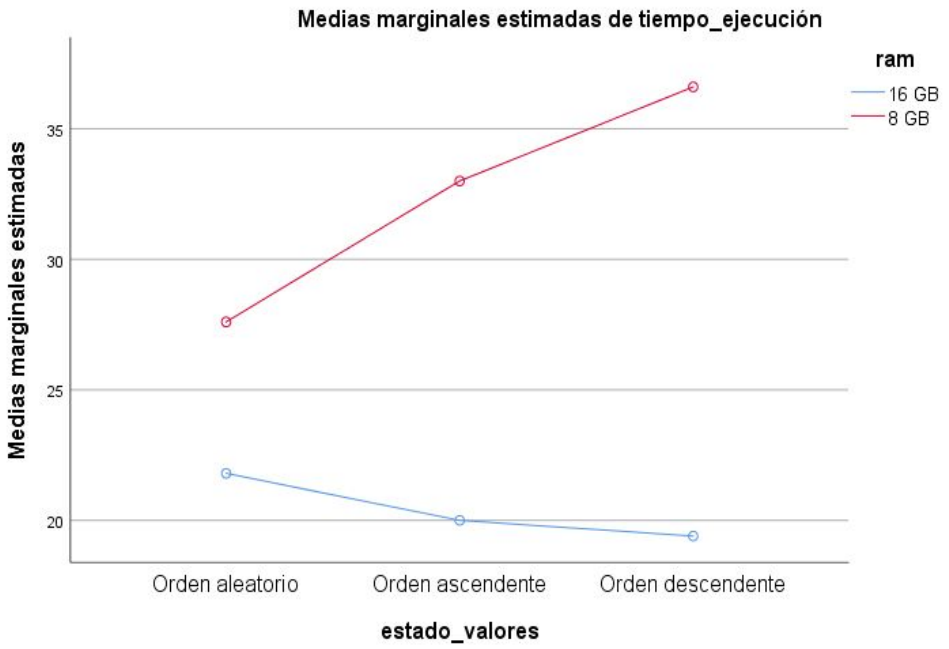
Merge sort	100000	Orden aleatorio	8 GB	66
Merge sort	100000	Orden ascendente	8 GB	33
Merge sort	100000	Orden descendente	8 GB	32

Y, teniendo el comportamiento que presenta al ejecutarse en el computador de tamaño de memoria RAM de 16 GB:

Merge sort	100000	Orden aleatorio	16 GB	41
Merge sort	100000	Orden ascendente	16 GB	27
Merge sort	100000	Orden descendente	16 GB	25

Finalmente, se comparan los comportamientos de tiempo de ejecución en un gráfico que confirma la anterior apreciación sobre los cambios que se presentan y la forma en que influye el tamaño de la memoria RAM durante este experimento.

Carlos Andrés Restrepo - César Leonardo Canales - Ana María Muñoz



Interpretación

Inicialmente, se tenía planeado como objetivo encontrar cuáles eran los recursos más adecuados para realizar la solución más eficiente dentro del contexto de la problemática. Así mismo, teniendo en cuenta que el tamaño de memoria RAM juega un papel vital para los resultados del experimento, se buscó confirmar la magnitud de influencia que representaría dentro de los distintos procesos que se realizaron con los cambios en las entradas que recibía: el tamaño del arreglo varió y el orden en el que se encontraban descritos los datos.

Igualmente, una de las conjeturas iniciales era esperar unas soluciones más eficientes cuando el RAM tuviera un mayor tamaño, lo cual a partir de los datos presentados en la anterior fase, fue posible de confirmar puesto que, de acuerdo a las medias que se obtuvieron para los tiempos de ejecución y los datos recolectados directamente del programa, permitieron apreciar y concluir que, efectivamente, las salidas se recolectaron en un tiempo menor con el computador de tamaño de memoria RAM más grande: el de 16 GB.

Del mismo modo, se interpretó que, el otro factor importante que influyó en la obtención de las soluciones fue el algoritmo que se utilizó para resolver la problemática de ordenamiento. Siendo

el algoritmo MergeSort el que permitió obtener los resultados de manera más eficiente pues tuvo los tiempos de ejecución más bajos.

A partir de lo anterior, y conociendo los datos obtenidos a partir de todas las combinaciones realizadas durante el experimento, el tratamiento ganador sería:

Numero de tratamiento	Algoritmo de ordenamiento	Tamaño del arreglo	Estado de los valores en el arreglo	Cantidad de memoria RAM	Tiempo en ms
18	Merge sort	10	Orden descendente	16 GB	1

Debido a que, cuenta con el tiempo de ejecución menor en comparación con las demás condiciones que se utilizaron y, se tiene en cuenta que los demás resultados obtenidos en el tiempo, el algoritmo que mejor comportamiento tuvo fue el MergeSort. Así como los mejores resultados fueron brindados por la memoria RAM de 16 GB.

No obstante, se destaca que el principal hallazgo al realizar este experimento es que entre los dos algoritmos estudiados, MergeSort y HeapSort, se tiene que el más eficiente es el MergeSort pues siempre presentó tiempos menores de ejecución cuando los tamaños de las entradas aumentaron a miles. Igualmente, fue posible descubrir que para tamaños de entradas pequeños, que van entre 10 y 100 elementos, los factores estudiados no afectaron mucho el tiempo de ejecución ya que el tiempo de ejecución para esos casos fue de 1 milisegundo o menos.

Control y conclusiones

- ➔ Se recomienda para una próxima implementación de alguno de estos algoritmos dentro de un contexto problemático similar, tener en cuenta que se obtienen resultados más eficientes al implementar el algoritmo MergeSort.
- ➔ Del mismo modo, se destaca que los factores y recursos que intervienen para el desarrollo de la solución debe involucrar características que permitan tener una medida de tiempo menor, que pueden ser ejecutar el programa en un computador con cantidad de memoria RAM, si es posible, de 16 GB.

- ➔ Finalmente, se reconoce que los tamaños de entrada afectan directamente al tiempo que toma el programa para presentar las salidas esperadas.

Referencias

Salas, R. E. y Rodríguez, J. E. (2013) *Análisis de complejidad algorítmica* [Artículo para Maestría, Universidad Nacional de Colombia]. Recuperado de https://www.academia.edu/28858762/An%C3%A1lisis_de_complejidad_algor%C3%ADtmica

Gómez, M. C. y Cervantes, J. (2014). *Introducción al Análisis y al Diseño de Algoritmos.* Recuperado de http://www.cua.uam.mx/pdfs/conoce/libroselec/Notas_Analisis_AlgoritmosVF.pdf

Informe de tiempo

Leaderboard
https://wakatime.com/leaders/sec/Experiment%2BDesign

César Canales Rivera
https://wakatime.com/@Sleeptight/projects/adxqeurlxp?start=2020-02-11&end=2020-02-17

Carlos Restrepo Marín
https://wakatime.com/@Carlosches/projects/nwsuvcjjgx?start=2020-02-11&end=2020-02-17

Ana María Muñoz Valencia
https://wakatime.com/@553e1bd1-411c-4071-be62-b09836413771/projects/uxqkwnsxnx?start=2020-02-11&end=2020-02-17