Analisis de algoritmos de ordenamiento

Bejar Merma Ángel Andrés
Universidad Nacional de San Agustin
andresbjar97@gmail.com
Ciudad de Arequipa

Resumen

El trabajo consiste en un análisis del rendimiento de un conjunto de algoritmos en función de su tiempo de ejecución con diferentes conjuntos de datos tanto para c++ como para Python. Se prueban, comparan y concluye qué algoritmo es mejor para conjuntos de datos pequeños, promedio, veremos el peor caso, caso promedio y mejor.

1. Introducción

La ordenación es una de las cuestiones importantes en computación el problema de ordenación es importante porque de eso depende otros algoritmos los que los hacen mas o menos eficientes uno de de estos algoritmos es el algoritmo de búsqueda .

El concepto clave es algoritmo ,un algoritmo se puede definir como una secuencia de pasos bien definidos para resolver un problemas . El algoritmo toma una entrada y proporciona una salida.[2] Se han propuesto varios algoritmos de ordenación con diferentes restricciones, p. Ej. número de iteraciones (bucle interno, bucle externo), complejidad y problema de consumo de CPU.

2. Marco Teórico

Los algoritmos de ordenamiento se dividen en dos categorías bien diferenciadas:el Bubble Sort, Insertion Sort ,y Selection Sort en la categoría de O(n^2), mientras que Quick Sort y Merge Sort O(n log n).La descripción de cada uno de ellos a continuación.

a) **Bubble sort:** El algoritmo de ordenacion básico es la clasificación de burbujas. Compara dos elementos adyacentes y realiza una operación de intercambio si se encuentra un pedido incorrecto con pasos repetidos. Esto también se denomina algoritmo de ordenación por comparación.Una de sus ventajas es sus facil implementacion

b) **Heap sort:** La ordenacion de montón es una técnica de clasificación basada en comparaciones basada en la estructura de datos binary heap. Es similar a ordenamiento por selección donde primero encontramos el elemento mínimo y colocamos el elemento mínimo al principio. Repetimos el mismo proceso para el resto de elementos[6].

c) **Insertion sort**
El ordenamiento por inserción (insertion sort en inglés) es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria. Requiere O(n^2) operaciones para ordenar una lista de n elementos.

Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento k+1 y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se inserta el elemento k + 1 debiendo desplazarse los demás elementos[7].

d) **Shell sort**

ShellSort es principalmente una variación de Insertion Sort. En la ordenación por inserción, movemos los elementos solo una posición adelante. Cuando un elemento debe moverse mucho

más adelante, se involucran muchos movimientos. La idea de shellSort es permitir el intercambio de elementos lejanos. En shellSort, hacemos que la matriz esté ordenada por h para un valor grande de h. Seguimos reduciendo el valor de h hasta que se convierte en 1. Se dice que una matriz está ordenada por h si todas las sublistas de cada elemento h están ordenadas[5].

e) **Merge sort**

Este es un algoritmo de divide y conquista, con la ventaja de fusionar listas con nuevas listas ordenadas. La complejidad del peor caso de la ordenación por fusión es O (nlog n), ya que podría usarse para conjuntos de datos grandes y peores. La ordenación por fusión utiliza los siguientes tres pasos [8]

1) **Divide:**Si el tamaño de la matriz es mayor que 1, divídalo en dos subarreglos iguales de la mitad del tamaño.
2) **Conquista:** ordenar ambas subarreglos por recursividad
3) **Fusiona:** Combine ambas arreglos ordenadas en uno de tamaño original. Esto le dará un arreglo ordenada completo.

La ordenación por fusión es más adecuada para casos grandes y peores, pero usa más memoria en comparación con otros algoritmos de división y conquista. El algoritmo de clasificación de fusión se describe a continuación

f) **Quick sort**

3.1 Clasificación de burbujas: el algoritmo de clasificación básico es la clasificación de burbujas. Compara dos elementos adyacentes y realiza una operación de intercambio si se encuentra un pedido incorrecto con pasos repetidos. Esto también se denomina algoritmo de clasificación por comparación [7]. El tipo de burbuja original lo hace

3. Metodología

Lo que se hizo fue probar los algoritmos de ordenamiento en Python utilizando Google colab y el entorno de ejecución GPU. La maquina que nos asigno google fue una Tesla k80 se intento reiniciar el entono para obtener una tesla t 40 , pero fue inútil.

Para la pruebas en c++ La computadora que utilizo es una intel i5 de quinta generación con 2 cores ,8 gb de memoria ram

Los algoritmos escogidos son los siguientes:

- Bubble sort
- Heap sort
- Insertion sort
- Selection sort
- Shell sort
- Merge sort
- Quick sort

como datos de entrada usaremos libreoffice para graficar los resultados

4. Resultados

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.74       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   40C    P0    58W / 149W |    121MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
```

Figura 1:

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  2
Core(s) per socket:  1
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               63
Model name:          Intel(R) Xeon(R) CPU @ 2.30GHz
Stepping:            0
CPU MHz:             2299.998
BogoMIPS:            4599.99
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            46080K
NUMA node0 CPU(s):   0,1
Flags:               fpu vme de pse tsc msr pae mce cx8
```

Figura 2:

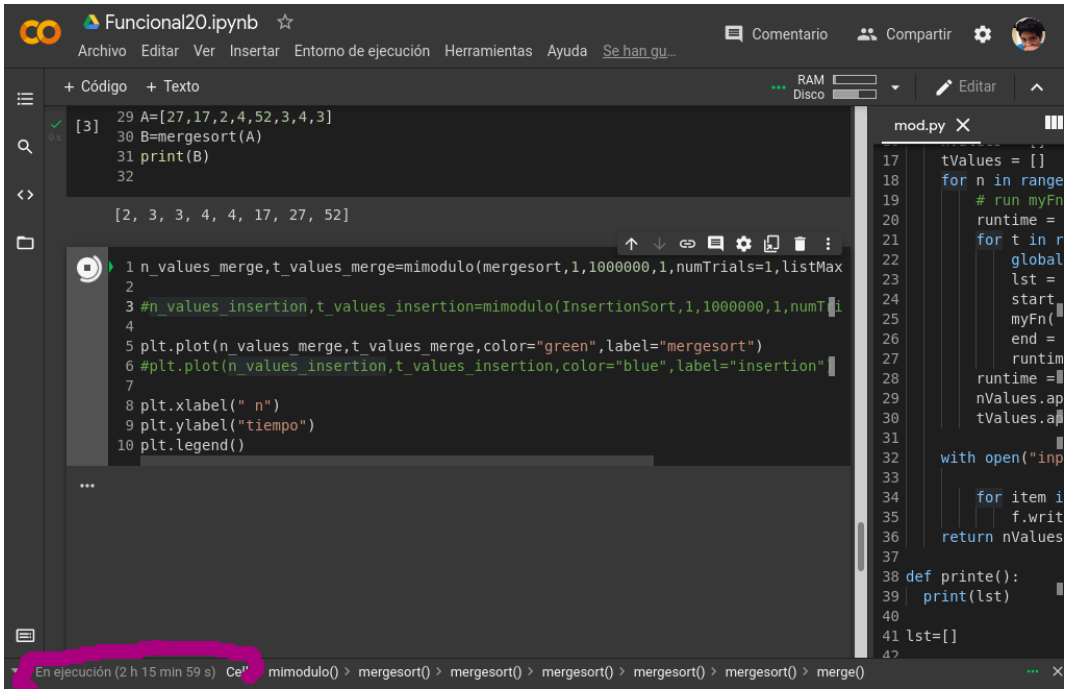

Figura 3:

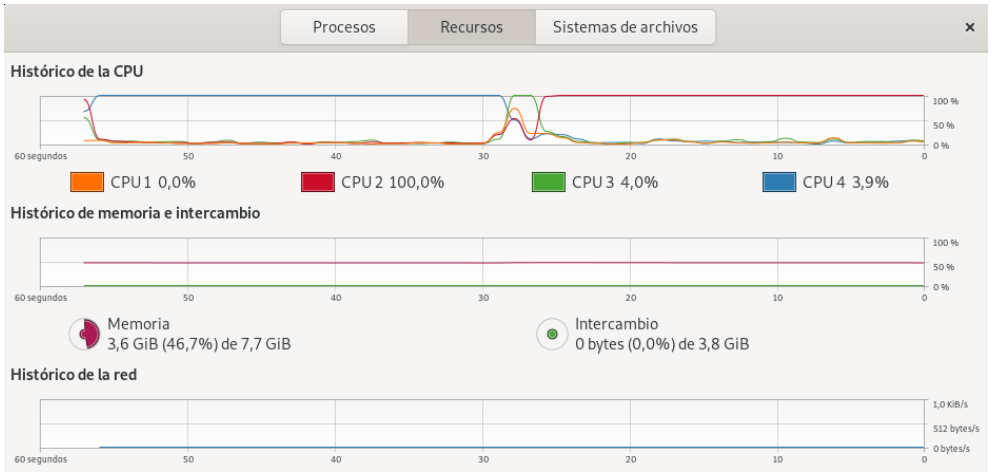

Figura 4:

cuaderno de trabajo [1]

```
angel@debianita:~$ gcc ordenacionlab.c -o ejeor
angel@debianita:~$ ./ejeor 100000 1
Bolha com Flag (100000 elementos - Aleatório)
Tiempo: 43464 ms

Insercion (100000 elementos - Aleatório)
Tiempo: 8618 ms

Seleccion (100000 elementos - Aleatório)
Tiempo: 15889 ms

Shell (100000 elementos - Aleatório)
Tiempo: 29 ms

Merge (100000 elementos - Aleatório)
Tiempo: 22 ms

Quick (100000 elementos - Aleatório)
Tiempo: 17 ms
```

Figura 5:

5. Conclusiones

Es importante establecer el paralelismo entre el ordenamiento por selección y el ordenamiento por inserción. La selección realiza menos cambios que la inserción, ya que solo hay un cambio por iteración, es decir, en el orden de selección total hace n intercambios. La ordenación por inserción, por otro lado, realiza al menos un cambio por iteración, ya que debe realizar cambios para eliminar cada elemento evaluado.

La ordenación por inserción, por otro lado, realiza menos comparaciones que la ordenación por selección, ya que el elemento que se va a insertar de forma ordenada no siempre tiene que llegar hasta el final. De hecho, esto solo ocurre en el peor de los casos, donde la matriz se ordena en orden inverso. La clasificación por selección necesita comparar todos los elementos restantes cada vez para determinar quién es el más pequeño.

En teoría, ambos pertenecen a la misma clase de complejidad, a saber, O (n^2)
. En la práctica, la ordenación por inserción funciona mejor que la ordenación por selección.

Finalmente, la clasificación por selección no se considera un algoritmo eficiente para entradas grandes. Hay alternativas O (nlogn) , como Quick Sort y Merge Sort, así como alternativas lineales como Counting Sort.

[3]

La complejidad de Quick Sort en los casos mejores y medios es O(nlogn) y en el peor de los casos O(n^2) . La complejidad media del tiempo de caso es O(nlogn) que es un poco costosa si se utiliza para grandes conjuntos de datos, también en el peor de los casos la complejidad es O(n^2), lo que lo hace impráctico para los peores conjuntos de datos grandes.

6. Discusión

Documentar sus hallazgos en un formato de artículo de investigación (formato de libre elección), considerando aspectos del marco teórico (estado del arte), metodología, resultados, conclusiones, discusión y bibliografía [4].

Referencias

[1] Bejar Merma Angel Andres. Cuaderno de trabajo. https://colab.research.google.com/drive/1FZjcnchMuRvmnAxCTwf54DTL1DiJmsNm?usp=sharing, 2021. [Online; accessed read].

[2] Varinder Kumar Bansal, Rupesh Srivastava, and Pooja. Indexed array algorithm for sorting. In *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pages 34–36, 2009.

[3] Arthur Brunet. Estructura de datos. https://joaoarthurbm.github.io/eda/posts/selection-sort/, 2021. [Online; accessed read].

[4] Jose Fager. *Estructura de datos*. Proyecto Latin. latin, 2014.

[5] geekforgeek. geekforgeek. https://www.geeksforgeeks.org/shellsort/, 2021. [Online; accessed read].

[6] heapsort. geekforgeek. https://www.geeksforgeeks.org/heap-sort/, 2021. [Online; accessed read].

[7] insertionsort. wikipedia. https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n, 2021. [Online; accessed read].

[8] Wikipedia. Definición de merge sort. https://es.wikipedia.org/wiki/Ordenamiento_por_mezcla, 2021. [Online; accessed].