

Analisis de algoritmos de ordenamiento

Bejar Merma Ángel Andrés

Universidad Nacional de San Agustín

andresbj97@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:**
- c) **Insertion sort**
- d) **Shell sort**
- 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 $O(n \log n)$, ya que podría usarse para conjuntos de datos grandes y peores. La ordenación por fusión utiliza los siguientes tres pasos [?]

- 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 entorno 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:

The screenshot shows a Jupyter Notebook titled 'Funcional20.ipynb'. The main code cell contains the following Python code:

```

[3] 29 A=[27,17,2,4,52,3,4,3]
    30 B=mergesort(A)
    31 print(B)
    32

```

The output of the code cell is:

```

[2, 3, 3, 4, 4, 17, 27, 52]

```

Below the output, there is a code cell with the following code:

```

1 n_values_merge,t_values_merge=mimodulo(mergesort,1,1000000,1,numTrials=1,listMax
2
3 #n_values_insertion,t_values_insertion=mimodulo(InsertionSort,1,1000000,1,numT
4
5 plt.plot(n_values_merge,t_values_merge,color="green",label="mergesort")
6 #plt.plot(n_values_insertion,t_values_insertion,color="blue",label="insertion"
7
8 plt.xlabel(" n")
9 plt.ylabel("tiempo")
10 plt.legend()

```

On the right side, there is a code cell titled 'mod.py' with the following code:

```

17 tValues = []
18 for n in range
19     # run myFn
20     runtime =
21     for t in r
22         global
23         lst =
24         start
25         myFn(
26         end =
27         runtim
28         runtime =
29         nValues.ap
30         tValues.a
31
32 with open("inp
33
34     for item i
35         f.writ
36     return nValues
37
38 def printe():
39     print(lst)
40
41 lst=[]
42

```

At the bottom, there is a status bar showing the execution time: 'En ejecución (2 h 15 min 59 s)'. The CPU usage is indicated as 'Celeron'.

Figura 3:

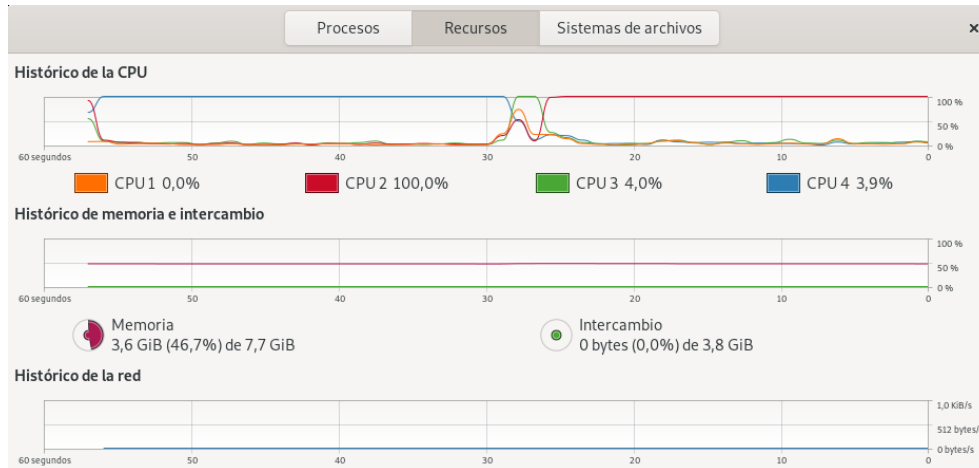


Figura 4:

cuaderno de trabajo [1]

5. Conclusiones

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 [3].

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