Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Laboratorio de Algoritmos y Estructuras II - CI2692

Profesor: Guillermo Palma
Estudiantes:
Haydeé Castillo Borgo. Carnet: 16-10209
Jesús Prieto. Carnet: 19-10211

Trimestre Abril-Julio 2023

Comparación de algunos algoritmos implementados en la librería Sortlib.kt
Laboratorio de la semana 5

En el presente laboratorio se llevó a cabo un estudio experimental para comparar el tiempo de ejecución de los siguientes algoritmos de ordenamiento, los cuales se encuentran implementados en la librería Sortlib.kt: Mergesort, desarrollado en [1]; Heapsort, desarrollado en [2]; Smoothsort, desarrollado en [3]; Quicksort Cásico, desarrollado en [2]; Quicksort Three Way, desarrollado en [4]; Quicksort Dual Pivot, desarrollado en [5]; Counting Sort y Radix Sort, desarrollados en [2].

Para dicho estudio se empleó un computador Intel® Core™ i5-2450M CPU @ 2.50GHz × 4, con 8Gb de RAM y sistema operativo Ubuntu 20.04.6 LTS. Además, se empleó el lenguaje de programación Kotlin en su versión 1.8.21 y Java Virtual Machine JVM, versión 11.0.19.

Es importante destacar que en la implementación de Radix Sort se empleó como algoritmo de ordenamiento estable una variante de Counting Sort, la cual utiliza un arreglo auxiliar para ordenar los elementos del arreglo principal de acuerdo a un dígito determinado. Dicho arreglo auxiliar contiene los dígitos de una posición especificada, de los elementos del arreglo a ordenar; y esta variante se diseñó en vista de que el algoritmo Counting Sort ordena un arreglo empleando las frecuencias de sus elementos, y para la ejecución de Radix Sort es necesario ordenarlo de acuerdo a las frecuencias de los dígitos de sus elementos.

Asimismo, es importante destacar que en la implementación de Mergesort se utiliza como algoritmo de ordenamiento auxiliar Insertion Sort, para ordenar arreglos de tamaño $n \leq 90$. Insertion Sort también se encuentra en la librería Sortlib.kt y es desarrollado en [6]; y el tamaño apropiado para implementar dicho algoritmo como auxiliar en Mergesort se determinó en un estudio experimental realizado anteriormente.

Todos los algoritmos analizados en el presente estudio se emplean para ordenar arreglos de tamaño n, con $n \geq 1$; y para comparar sus tiempos de ejecución en la práctica, se consideraron cuatro tamaños de arreglos: $n = 1000000$, $n = 2000000$, $n = 3000000$ y $n = 4000000$.

Para cada tamaño n se generaron 3 arreglos aleatorios, cuyos elementos venían dados por valores en el intervalo $[0, 1000000)$, se ejecutó cada algoritmo en cada uno de los 3 arreglos, se tomó el tiempo correspondiente a cada prueba (en segundos) y se calcularon los promedios y desviaciones estándar, agrupando las pruebas de acuerdo al tamaño del arreglo.

Los resultados obtenidos se presentan en la siguiente tabla y en el siguiente gráfico.

Nombre del algoritmo	**n = 1.000.000**	**n = 2.000.000**	**n = 3.000.000**	**n = 4.000.000**
Mergesort	$0{,}8590 \pm 0{,}495$	$1{,}8540 \pm 0{,}1051$	$2{,}6300 \pm 0{,}0434$	$3{,}6373 \pm 0{,}1474$
Heapsort	$1{,}265 \pm 0{,}0455$	$3{,}0140 \pm 0{,}2804$	$4{,}8003 \pm 0{,}2543$	$6{,}6343 \pm 0{,}1323$
Smoothsort	$3{,}0293 \pm 0{,}3819$	$6{,}7053 \pm 0{,}7065$	$12{,}3640 \pm 0{,}4299$	$19{,}0557 \pm 0{,}3984$
Quicksort Clásico	$0{,}5220 \pm 0{,}1191$	$1{,}0170 \pm 0{,}1196$	$1{,}7347 \pm 0{,}2718$	$2{,}9820 \pm 0{,}2451$
Quicksort Three Way	$\mathbf{0{,}3660 \pm 0{,}1481}$	$\mathbf{0{,}7393 \pm 0{,}1481}$	$1{,}400 \pm 0{,}2742$	$1{,}917 \pm 0{,}2911$
Quicksort Dual Pivot	$0{,}4590 \pm 0{,}0850$	$0{,}8677 \pm 0{,}2111$	$1{,}4140 \pm 0{,}1940$	$2{,}1487 \pm 0{,}1059$
Counting Sort	$0{,}4423 \pm 0{,}0830$	$0{,}8033 \pm 0{,}1183$	$\mathbf{1{,}1016 \pm 0{,}0627}$	$\mathbf{1{,}5763 \pm 0{,}1628}$
Radix Sort	$0{,}8380 \pm 0{,}1764$	$1{,}510 \pm 0{,}0624$	$2{,}5550 \pm 0{,}0493$	$4{,}1786 \pm 0{,}1763$

Tabla 1: Tiempos de ejecución de los algoritmos para arreglos de tamaño n

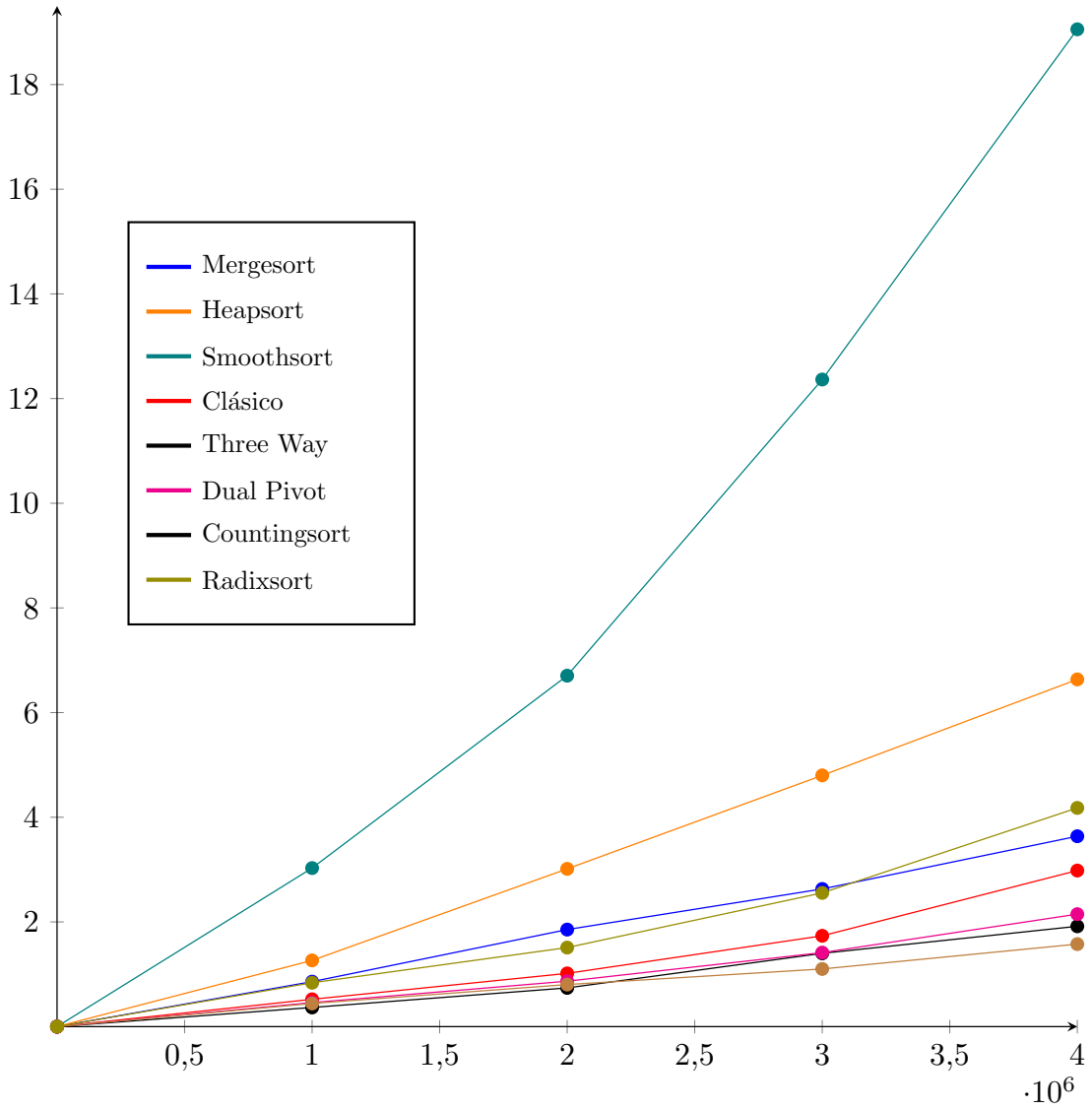

Figura 1: Comportamiento de los algoritmos en función del tamaño del arreglo. Con el eje Y representando en tiempo(*segundos*) y el eje X representando el tamaño(*millones*)

En el estudio se obtuvó que el algoritmo Quicksort Three Way resultó ser el más rápido para los tamaños $n = 1000000$ y $n = 2000000$, mientras que Counting Sort fue el más rápido para los tamaños $n = 3000000$ y $n = 4000000$.

Como era de esperarse, el algoritmo de ordenamiento lineal Counting Sort resulta ser muy eficiente en la práctica para arreglos de gran tamaño, siendo además el segundo más rápido para los tamaños $n = 1000000$ y

$n = 2000000$. Sin embargo, a pesar de ser de orden lineal, dicho algoritmo es menos efectivo que Quicksort Three Way para $n = 1000000$ y $n = 2000000$; esto se debe a que, a diferencia de la variante de Quicksort, Counting Sort no ordena el arreglo en el lugar (*in place*) sino que requiere de estructuras auxiliares que representan un incremento en el tiempo de ejecución en la práctica.

Por su parte, Radix Sort no probó ser tan eficiente como se esperaría al ser un algoritmo de ordenamiento lineal. Esto se debe a que las estructuras auxiliares requeridas para poder ordenar los elementos de acuerdo a sus dígitos producen un incremento en el tiempo de ejecución en la práctica; además del incremento generado por las estructuras auxiliares empleadas por la variante de Counting Sort (el cual, como ya se mencionó, no ordena el arreglo *in place*).

De todo lo anterior se concluye que, entre los algoritmos implementados en Sortlib.kt, los algoritmos Quicksort Three Way y Counting Sort resultan ser los más eficientes y recomendados al momento de ordenar arreglos de gran tamaño.

Referencias

[1] Brassard, G., and Bratley, P. *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[2] Cormen, T., Leirserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms*, 3ra ed. McGraw Hill, 2009.

[3] Dijkstra, E. W. Smoothsort, an alternative for sorting in situ. Tech. rep., Burroughs Corporation, 1981.

[4] Sedgewick, R., and Bentley, J. Quicksort is optimal. https://sedgewick. io/wp-content/uploads/2022/03/2002QuicksortIsOptimal.pdf, 2002. KnuthFest, Stanford University.

[5] Wild, S., and Nebel, M. E. Average case analysis of java 7's dual pivot quicksort. In *Algorithms–ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, Sep- tember 10-12, 2012. Proceedings 20* (2012), Springer, pp. 825–836.

[6] Aho, A., Hopcroft, J., and Ullman, J. *Data structures and algorithms*. Addison- Wesley, 1983.