*Universidad de los Andes*

*Estructuras de Datos*

*Daniel Yamin*

*Sebastian Puerto*

*Febrero 24, 2019*

*ISIS 1206*

# **Taller 4: Ordenamiento**

1. **Análisis Teórico de los Algoritmos:**

**i) Shellsort:**

* El mejor caso es cuando es cuando la lista ya está ordenada ascendentemente. En términos generales, el algoritmo es muy bueno para listas parcialmente ordenadas. En dicho caso el número de comparaciones ~es decir . .
* El peor caso se presenta cuando la lista está ordenada descendentemente. En dicho caso, el número de comparaciones ~es decir .

En este caso no se tiene un promedio de comparaciones para el algoritmo puesto que depende principalmente de la secuencia de incremento. Al momento no se ha podido demostrar la secuencia óptima. Sin embargo, comúnmente se utiliza la secuencia . Los resultados teóricos presentados corresponden a Shellsort utilizando la secuencia en cuestión.

**ii) Mergesort:**

* En el mejor caso el número de comparaciones ~es decir . .
* En el peor caso el número de comparaciones ~es decir .
* En el caso promedio *(average)* del algoritmo, MergeSort realiza ~ comparaciones. Es decir

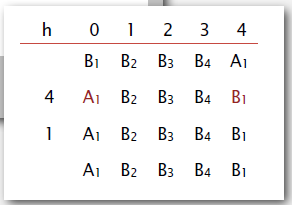
Nótese que en los tres casos se obtiene el mismo resultado. Esto se debe a que en el Mergesort, el input (la muestra) que se desea ordenar no influye en el número de comparaciones. Así, no se tienen situaciones en las cuales ocurrirán el mejor caso o el peor.

**iii) Quicksort:**

* El mejor caso es Quicksort se da para una lista aleatoriamente ordenada. En dicho caso el número de comparaciones ~½ es decir .
* El peor caso se presenta cuando la lista está ordenada ascendentemente o cuando se tienen muchos elementos repetidos. En dichos casos, el número de comparaciones~ es decir .
* También es importante mencionar el caso promedio *(average)* del algoritmo. En promedio, Quicksort realiza ~1.39 comparaciones. Es decir

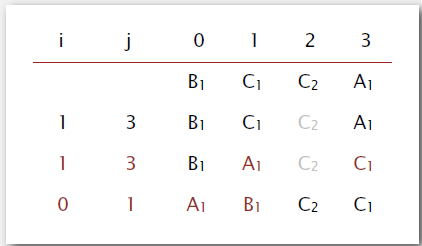
**2. Propiedades de los Algoritmos:**

Shellsort es un algoritmo ***inPlace*** puesto que se opera sobre el mismo arreglo para realizar el ordenamiento. Este algoritmo **no es adaptativo** pues no cambia su conducta durante la ejecución. Por otro lado, el algoritmo **no es Estable**, como se muestra a continuación:



MergeSort es **no es un algoritmo *inPlace*** puesto que requiere de un arreglo auxiliar para lograr el ordenamiento de los datos. Por otro lado, este algoritmo **sí es Estable**, es decir, se preserva el orden relativo para *items* con la misma *key*. Este algoritmo **no es adaptativo** pues no cambia su conducta durante la ejecución.

QuickSort es un algoritmo ***inPlace*** pues el ordenamiento se realiza en el mismo arreglo. Es decir, no requiere de la creación de arreglos duplicados o auxiliares pues opera sobre el arreglo original. Este algoritmo **no es adaptativo** pues no cambia su conducta durante la ejecución. Más aún, el algoritmo **no es Estable**. Esto quiere decir que los elementos con el mismo valor no ocupan la misma posición relativa entre ellos en el arreglo ordenado comparándola con el arreglo desordenado. A continuación se muestra un prueba:



**3. Resultados:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tamaño Muestra | Shellsort (mseg) | Mergesort (mseg) | Quicksort (mseg) |
| 30000 | 52 | 58.7 | 65.3 |
| 60000 | 104.75 | 100.5 | 76.75 |
| 90000 | 168 | 143/75 | 84.5 |
| 120000 | 230 | 180.3 | 123.7 |
| 150000 | 428.7 | 325 | 206.3 |
| 180000 | 552 | 499 | 250 |
| 210000 | 562 | 376 | 288 |
| 240000 | 652 | 420 | 314 |

* Datos Ordenados (ascendentemente):

|  |  |  |  |
| --- | --- | --- | --- |
| Tamaño Muestra | Shellsort (mseg) | Mergesort (mseg) | Quicksort (mseg) |
| 30000 | 27.3 | 40 | 39 |
| 60000 | 46.7 | 74 | 56 |
| 90000 | 68.3 | 106.3 | 91 |
| 120000 | 96.7 | 136.7 | 117.7 |
| 150000 | 125.7 | 298.3 | 219 |
| 180000 | 156 | 321 | 252 |
| 210000 | 160 | 332 | 284 |
| 240000 | 184 | 358 | 319 |

* Datos Ordenados (descendentemente):

|  |  |  |  |
| --- | --- | --- | --- |
| Tamaño Muestra | Shellsort (mseg) | Mergesort (mseg) | Quicksort (mseg) |
| 30000 | 40.7 | 39 | 38.7 |
| 60000 | 69.7 | 67.3 | 59 |
| 90000 | 116 | 102.7 | 93 |
| 120000 | 156.7 | 140.7 | 122.7 |
| 150000 | 246.5 | 276.75 | 229.7 |
| 180000 | 305 | 311 | 245 |
| 210000 | 342 | 371 | 298 |
| 240000 | 370 | 359 | 332 |

**4. Conclusión:** Teóricamente, a partir de la complejidad, el algoritmo más rápido es el Mergesort. Sin embargo, Mergesort tiene más intercambios que el Quicksort y consume mayor cantidad de memoria puesto que requiere de un arreglo auxiliar (no es inPlace). Teniendo en cuenta lo anterior, en la práctica, el Quicksort es el algoritmo de ordenamiento más eficiente en términos generales. Es importante resaltar que evidentemente la eficiencia depende de los objetivos del cliente por lo que se debe evaluar los requerimientos y los datos para determinar cuál algoritmo implementar.