**Universidad Tecnológica Centroamericana**

**UNITEC**

[**Comparación de Algoritmos de Ordenamiento**](https://lms2.unitec.edu/mod/assign/view.php?id=9740511)

**Estudiante:**

|  |  |
| --- | --- |
| Dennis Cheong Cheong | 11241278 |

**Análisis de Algoritmo, Sección: 610**

**Catedrático: Jorge Antonio García Bolaños**

**Tegucigalpa, M.D.C., 31 /08/2016**

**Contenido**

[**Introducción** 3](#_Toc459494746)

[**Marco Teórico** 4](#_Toc459494747)

[**Implementación** 5](#_Toc459494748)

[**Resultados** 6](#_Toc459494749)

[**Conclusiones** 12](#_Toc459494750)

[**Anexos** 13](#_Toc459494751)

[**Referencias** 14](#_Toc459494752)

# **Introducción**

El siguiente informe técnico detalla la eficiencia de cada uno de los diferentes algoritmos de ordenamiento que se discutió durante el transcurso de la clase. Dichos algoritmos de ordenamiento son: BUBBLE SORT, INSERTION SORT, SELECTION SORT, MERGE SORT, HEAP SORT (en un mismo arreglo), QUICK SORT (en un mismo arreglo), RADIX SORT.

Para realizar el análisis de eficiencia se requirió ejecutar cada uno de los algoritmos con un arreglo de 10, 100, 1000, 10000, 100000 y 1000000 de elementos. En cada uno de los casos 10 veces, esto con el fin de tener un análisis estadístico básico de la eficiencia de los algoritmos implementados.

# **Marco Teórico**

Dado un arreglo de elementos desordenados de tamaño 10, 100, 1000, 10000, 100000 o 1000000, se implementaron varios algoritmos que ordenaran dicho arreglo de forma ascendente. Para cada tamaño se realizó 10 veces cada algoritmo y en cada iteración se obtuvo el tiempo promedio que se tardó en hacer el ordenamiento.

Cada algoritmo tiene su estrategia para ordenar:

**Bubble Sort**, la idea básica de este algoritmo consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que todos se encuentren ordenados. Se realizan N-1 pasadas, transportando en cada una de las mismas el menor o mayor elemento (Según sea el caso) su posición ideal. Al final de las **N-1** pasadas los elementos del arreglo estarán ordenados. (Anónimo, MIS ALGORITMOS, 2016)

**Insertion Sort**, consiste en ir insertando un elemento de la lista o un arreglo en la parte ordenada de la misma, asumiendo que el primer elemento es la parte ordenada, el algoritmo ira comparando un elemento de la parte desordenada de la lista con los elementos de la parte ordenada, insertando el elemento en la posición correcta dentro de la parte ordenada, y así sucesivamente hasta obtener la lista ordenada. (Anónimo, Conoce 3000, 2016)

**Selection Sort**, encontrar el menor de todos los elementos del arreglo o vector e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenarlo todo. Su implementación requiere O (n2) comparaciones e intercambios para ordenar una secuencia de elementos. (Anónimo, EcuRed, 2016)

**Merge Sort**, se ordena la primera mitad de la lista, se ordena la segunda mitad de la lista y una vez ordenadas su mezcla da lugar a una lista de elementos ya ordenada.  A su vez la ordenación de la sublista mitad sigue los mismos pasos, ordenar la primera mitad, ordenar la segunda mitad y mezclar.  La sucesiva división de la lista actual en dos hace que el problema (número de elementos) cada vez sea más pequeño; así hasta que la lista actual tenga un elemento por lo tanto se considera ordenada, es el caso base, y a partir de 2 sublistas de un número mínimo de elementos, se mezclan dando cada vez a listas ordenadas de cada vez más elementos hasta alcanzar la lista total. (Anónimo, UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA , 2016)

**Heap Sort**, consiste en insertar todos los elementos del vector en un montículo y aprovechar que de esta forma siempre tendremos el mayor elemento en la raíz. Luego para obtener los elementos ordenados de mayor a menor, solo debemos tomar el valor de la raíz, eliminar la raíz y reordenar el montículo cada vez hasta agotar sus nodos. (Anónimo, ALGUIEN EN LA FISI, 2016)

**Radix Sort**, es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual (Anónimo, EstructuradedatosITP, 2016). Consiste en ir contando la cantidad de veces que aparece un número con determinada cifra para luego acumular los contadores y obtener la nueva posición de los números a ordenar. (Anónimo, Conoce 3000, 2016)

**Quick Sort**, tomando un elemento cualquiera de una posición cualquiera del arreglo. Este será nuestro elemento Pivote. Ubica el Pivote en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales al pivote y todos los elementos que se encuentren a su derecha sean mayores o iguales. Se repiten los pasos anteriores pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta del pivote en el arreglo. (Hernandez, 2016)

# **Implementación**

Puede encontrar el código fuente del proyecto en el siguiente enlace: <https://github.com/elco45/sort_project>

# **Resultados**

Gráfica 1. No se logra apreciar el resultado de Heap Sort, Merge Sort, Radix Sort puesto que los valores están muy cercanos al Quick Sort.

Gráfica 2. Dándole un “zoom” se puede apreciar de una mejor manera el comportamiento de los diferentes algoritmos.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Promedio (segundos) | | | | | | |
|  | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| Bubble Sort | 0.000006069 | 0.000198 | 0.000872 | 0.110166 | 14.51769 | 145.1769 |
| Selection Sort | 2.6774E-06 | 0.00016 | 0.001159 | 0.135517 | 13.69025 | 136.9025 |
| Insertion Sort | 1.4724E-06 | 7.52E-05 | 0.000149 | 0.013165 | 1.288533 | 12.88533 |
| Heap Sort | 2.6328E-06 | 2.36E-05 | 0.000197 | 0.001699 | 0.014653 | 0.205177 |
| Merge Sort | 8.6126E-06 | 4.78E-05 | 0.000248 | 0.001482 | 0.015382 | 0.183468 |
| Radix Sort | 2.06614E-05 | 0.000174 | 0.00022 | 0.001417 | 0.01515 | 0.154137 |
| Quick Sort | 2.1866E-06 | 3.09E-05 | 0.000118 | 0.000823 | 0.009766 | 0.114376 |

Tabla 1. Promedio en segundo por cada algoritmo y su respectiva cantidad de elementos en el arreglo.

Grafos y Tablas ara cada Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bubble Sort | | | | | | |
|  | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 1.87E-05 | 2.14E-04 | 9.29E-04 | 0.110652 | 13.51852 | 135.1852 |
| 2 | 2.68E-06 | 1.75E-04 | 8.84E-04 | 0.110255 | 14.14936 | 141.4936 |
| 3 | 2.23E-06 | 1.70E-04 | 8.79E-04 | 0.108485 | 15.02747 | 150.2747 |
| 4 | 2.68E-06 | 1.81E-04 | 8.83E-04 | 0.108528 | 14.51471 | 145.1471 |
| 5 | 1.16E-05 | 1.84E-04 | 8.58E-04 | 0.109223 | 14.55582 | 145.5582 |
| 6 | 1.29E-05 | 2.06E-04 | 8.50E-04 | 0.107274 | 14.48346 | 144.8346 |
| 7 | 2.23E-06 | 2.09E-04 | 8.52E-04 | 0.10909 | 14.8592 | 148.592 |
| 8 | 2.68E-06 | 2.16E-04 | 8.63E-04 | 0.109802 | 14.7901 | 147.901 |
| 9 | 2.23E-06 | 2.08E-04 | 8.51E-04 | 0.115428 | 14.72129 | 147.2129 |
| 10 | 2.68E-06 | 2.18E-04 | 8.73E-04 | 0.112927 | 14.55694 | 145.5694 |
| Average | 6.07E-06 | 0.000198 | 0.000872 | 0.110166 | 14.51769 | 145.1769 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Selection Sort | | | | | | |
|  | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 2.68E-06 | 1.75E-04 | 0.001168 | 0.131049 | 13.72027 | 137.2027 |
| 2 | 2.68E-06 | 1.80E-04 | 0.00118 | 0.133364 | 13.855 | 138.55 |
| 3 | 2.68E-06 | 1.74E-04 | 0.001189 | 0.136514 | 13.50318 | 135.0318 |
| 4 | 2.68E-06 | 1.56E-04 | 0.001126 | 0.136078 | 13.4856 | 134.856 |
| 5 | 2.23E-06 | 1.60E-04 | 0.001324 | 0.133838 | 13.82054 | 138.2054 |
| 6 | 3.57E-06 | 1.55E-04 | 0.00108 | 0.134223 | 14.0965 | 140.965 |
| 7 | 2.23E-06 | 1.54E-04 | 0.001119 | 0.134696 | 13.94344 | 139.4344 |
| 8 | 2.68E-06 | 1.51E-04 | 0.001094 | 0.142301 | 13.57419 | 135.7419 |
| 9 | 2.68E-06 | 1.47E-04 | 0.001135 | 0.1345 | 13.4831 | 134.831 |
| 10 | 2.68E-06 | 1.50E-04 | 0.001171 | 0.138603 | 13.42068 | 134.2068 |
| Average | 2.68E-06 | 0.00016 | 0.001159 | 0.135517 | 13.69025 | 136.9025 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Insertion Sort | | | | | | |
|  | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 1.79E-06 | 6.38E-05 | 1.49E-04 | 0.012993 | 1.295372 | 12.95372 |
| 2 | 1.34E-06 | 8.34E-05 | 1.57E-04 | 0.014175 | 1.291754 | 12.91754 |
| 3 | 1.34E-06 | 7.27E-05 | 1.47E-04 | 0.013004 | 1.290325 | 12.90325 |
| 4 | 1.79E-06 | 7.72E-05 | 1.58E-04 | 0.013572 | 1.266798 | 12.66798 |
| 5 | 1.79E-06 | 7.72E-05 | 1.50E-04 | 0.013185 | 1.280413 | 12.80413 |
| 6 | 1.79E-06 | 6.38E-05 | 1.45E-04 | 0.012909 | 1.31825 | 13.1825 |
| 7 | 1.34E-06 | 7.63E-05 | 1.47E-04 | 0.012722 | 1.318772 | 13.18772 |
| 8 | 1.34E-06 | 7.81E-05 | 1.40E-04 | 0.012846 | 1.278212 | 12.78212 |
| 9 | 8.92E-07 | 7.36E-05 | 1.47E-04 | 0.013262 | 1.280853 | 12.80853 |
| 10 | 1.34E-06 | 8.57E-05 | 1.47E-04 | 0.012987 | 1.264584 | 12.64584 |
| Average | 1.47E-06 | 7.52E-05 | 0.000149 | 0.013165 | 1.288533 | 12.88533 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Heap Sort | | | | | | |
|  | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 5.36E-06 | 4.51E-05 | 1.72E-04 | 0.00185 | 0.015387 | 0.195439 |
| 2 | 3.12E-06 | 2.05E-05 | 1.67E-04 | 0.002049 | 0.014539 | 0.194786 |
| 3 | 2.68E-06 | 1.96E-05 | 1.61E-04 | 0.001864 | 0.014613 | 0.206214 |
| 4 | 2.23E-06 | 1.92E-05 | 1.54E-04 | 0.001832 | 0.014606 | 0.215234 |
| 5 | 1.79E-06 | 1.92E-05 | 2.19E-04 | 0.001823 | 0.014804 | 0.216397 |
| 6 | 2.23E-06 | 2.32E-05 | 2.31E-04 | 0.001488 | 0.014842 | 0.211329 |
| 7 | 2.23E-06 | 1.87E-05 | 2.30E-04 | 0.001868 | 0.014504 | 0.213245 |
| 8 | 1.79E-06 | 2.32E-05 | 2.23E-04 | 0.001241 | 0.014387 | 0.197435 |
| 9 | 3.12E-06 | 2.14E-05 | 2.33E-04 | 0.001248 | 0.014534 | 0.201109 |
| 10 | 1.79E-06 | 2.54E-05 | 1.78E-04 | 0.001728 | 0.014313 | 0.200579 |
| Average | 2.63E-06 | 2.36E-05 | 0.000197 | 0.001699 | 0.014653 | 0.205177 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Merge Sort | | | | | | |
|  | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 2.45E-05 | 7.01E-05 | 2.00E-04 | 0.001444 | 0.017113 | 0.192306 |
| 2 | 4.91E-06 | 8.30E-05 | 1.90E-04 | 0.001386 | 0.018646 | 0.171943 |
| 3 | 5.36E-06 | 6.29E-05 | 1.76E-04 | 0.001434 | 0.01471 | 0.185374 |
| 4 | 1.20E-05 | 6.87E-05 | 8.44E-04 | 0.001637 | 0.014571 | 0.187427 |
| 5 | 1.47E-05 | 5.36E-05 | 1.80E-04 | 0.001426 | 0.015633 | 0.18241 |
| 6 | 4.91E-06 | 3.53E-05 | 1.80E-04 | 0.001598 | 0.014487 | 0.187442 |
| 7 | 5.80E-06 | 3.84E-05 | 1.76E-04 | 0.001654 | 0.014539 | 0.191569 |
| 8 | 4.46E-06 | 3.48E-05 | 1.85E-04 | 0.001383 | 0.015566 | 0.175222 |
| 9 | 4.46E-06 | 1.70E-05 | 1.70E-04 | 0.001425 | 0.014535 | 0.191773 |
| 10 | 4.91E-06 | 1.43E-05 | 1.80E-04 | 0.001433 | 0.014024 | 0.169213 |
| Average | 8.61E-06 | 4.78E-05 | 0.000248 | 0.001482 | 0.015382 | 0.183468 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Radix Sort | | | | | | |
|  | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 3.97E-05 | 1.73E-04 | 2.99E-04 | 0.001473 | 0.015794 | 0.145952 |
| 2 | 1.92E-05 | 1.82E-04 | 2.61E-04 | 0.001384 | 0.01529 | 0.158713 |
| 3 | 2.23E-05 | 1.61E-04 | 2.04E-04 | 0.00135 | 0.014433 | 0.143182 |
| 4 | 1.79E-05 | 1.81E-04 | 1.99E-04 | 0.001302 | 0.01553 | 0.151856 |
| 5 | 1.83E-05 | 1.91E-04 | 2.22E-04 | 0.001515 | 0.015779 | 0.140292 |
| 6 | 1.79E-05 | 1.64E-04 | 1.98E-04 | 0.001653 | 0.016589 | 0.184044 |
| 7 | 1.83E-05 | 1.74E-04 | 2.01E-04 | 0.001414 | 0.015973 | 0.167635 |
| 8 | 1.74E-05 | 1.68E-04 | 1.98E-04 | 0.001446 | 0.015878 | 0.147659 |
| 9 | 1.74E-05 | 1.66E-04 | 2.23E-04 | 0.001311 | 0.013106 | 0.139439 |
| 10 | 1.83E-05 | 1.79E-04 | 1.98E-04 | 0.001323 | 0.013123 | 0.162595 |
| Average | 2.07E-05 | 0.000174 | 0.00022 | 0.001417 | 0.01515 | 0.154137 |

# **Conclusiones**

Según el análisis realizado, se pudo concluir que los algoritmos ordenados por su eficiencia de mejor a peor son: Quick Sort, Radix Sort, Merge Sort, Heap Sort, Insertion Sort,Selection Sort, Bubble Sort.

El Insertion Sort llegó a ser mucho más eficiente que el Bubble Sort y el Selection Sort inclusive cuando los tres eran O(n^2).

Tras realizar el análisis se comprobó que Radix Sort realmente es mejor que Heap Sort y Merge Sort como se había discutido en clase.

El algoritmo Quick Sort como se mencionó en clase, resultó ser el más eficiente en comparación a los demás algoritmos.

# 

# **Anexos**

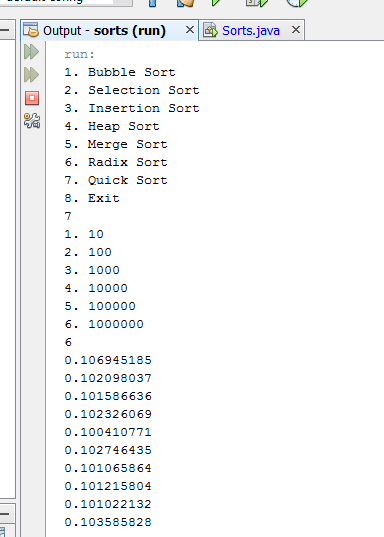


Imagen 1. Quick Sort con 1, 000,000 de elementos. Resultado en segundos.

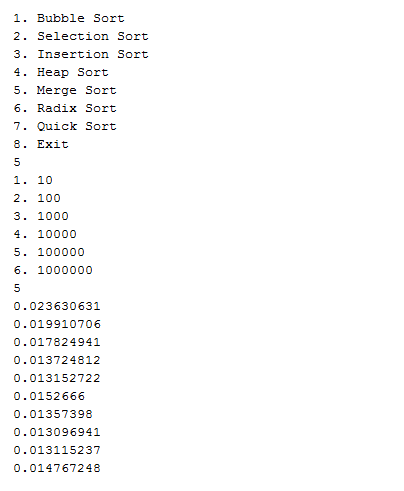


Imagen 2. Merge Sort con 100,000 elementos. Resultado en segundos.

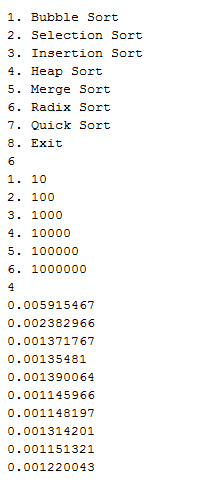


Imagen 3. Radix Sort con 10,000 elementos. Resultados en segundos.

# **Referencias**

Anónimo. (20 de 8 de 2016). *ALGUIEN EN LA FISI*. Obtenido de http://blog.alguien.site/2011/10/ordenamiento-por-monticulos-heapsort.html

Anónimo. (20 de 08 de 2016). *Conoce 3000*. Obtenido de http://www.conoce3000.com/html/espaniol/Libros/PascalConFreePascal/Cap08-02-Ordenamiento%20por%20insercion%20(Insertion%20sort).php

Anónimo. (20 de 8 de 2016). *Conoce 3000*. Obtenido de http://www.conoce3000.com/html/espaniol/Libros/PascalConFreePascal/Cap08-05-Ordenamiento%20por%20bases%20(Radix%20sort).php

Anónimo. (20 de 8 de 2016). *EcuRed*. Obtenido de http://www.ecured.cu/Algoritmo\_de\_ordenamiento\_por\_selecci%C3%B3n

Anónimo. (20 de 8 de 2016). *EstructuradedatosITP*. Obtenido de https://estructuradedatositp.wikispaces.com/5.1.4.+M%C3%A9todo+Radix

Anónimo. (20 de 08 de 2016). *MIS ALGORITMOS*. Obtenido de http://www.mis-algoritmos.com/bubble-sort-intercambio-directo

Anónimo. (20 de 8 de 2016). *UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA* . Obtenido de http://yaqui.mxl.uabc.mx/~eherrera/merge.htm

Hernandez, C. (20 de 8 de 2016). *Quicksort*. Obtenido de https://quicksort.wordpress.com/2008/11/21/fundamento-del-algoritmo/