Informe tarea 2

Nombre: Diego Alejandro Torreblanca Cordova

Rut: 19.699.326-2

Profesor: Travis Gagie

Este informe trata sobre el desempeño de distintos algoritmos los cuales fueron usados en la tarea Nº2

1. Descripción de datos:

¿ Qué algoritmo tiene mejor y peor desempeño?

|  |  |
| --- | --- |
| BenchmarkBubbleSortRandom1k-4 | 1017 ns/op - Mejor |
| BenchmarkSelectionSortRandom1k-4 | 1110531 ns/op - Peor |
| BenchmarkInsertionSortRandom1k-4 | 1999 ns/op |
| BenchmarkMergeRandom1k-4 | 425737 ns/op |
| BenchmarkQuickSortRandom1k-4 | 283118 ns/op |

|  |  |
| --- | --- |
| BenchmarkBubbleSortRandom10k-4 | 11894 ns/op- Mejor |
| BenchmarkSelectionSortRandom10k-4 | 106471083 ns/op - Peor |
| BenchmarkInsertionSortRandom10k-4 | 20194 ns/op |
| BenchmarkMergeRandom10k-4 | 5767112 ns/op |
| BenchmarkQuickSortRandom10k-4 | 4393264 ns/op |

|  |  |
| --- | --- |
| BenchmarkBubbleSortRandom100k-4 | 21496706023 ns/op- Peor |
| BenchmarkSelectionSortRandom100k-4 | 10445142490 ns/op |
| BenchmarkInsertionSortRandom100k-4 | 10445142490 ns/op |
| BenchmarkMergeRandom100k-4 | 69142771 ns/op |
| BenchmarkQuickSortRandom100k-4 | 47161317 ns/op - Mejor |

|  |  |
| --- | --- |
| BenchmarkBubbleSortRandom1m-4 | 2270193793665 ns/op - Peor |
| BenchmarkSelectionSortRandom1m-4 | 1044150635168 ns/op |
| BenchmarkInsertionSortRandom1m-4 | 417827673252 ns/op |
| BenchmarkMergeRandom1m-4 | 767578220 ns/op |
| BenchmarkQuickSortRandom1m-4 | 542239359 ns/op - Mejor |

Según estos datos consideraría que Quicksort seria el mejor ya que es lineal, mientras que Bubblesort funciono considerablemente peor a mayor cantidad de números ya que aumenta cuadráticamente su tiempo con respecto a la cantidad de numeros.

¿Tomando en cuenta la pregunta anterior, los mismos algoritmos se comportan de igual forma para los cuatro tipos de input?

Se puede observar que Duplicates y Random se comportan prácticamente igual, en cambio, Reversed demora bástate menos y Sorted al ya estar ordenado demora considerablemente menos que todos los demás inputs.

Por lo tanto se puede concluir que los tiempos de demora de cada input varían mas con respecto a la mayor cantidad de orden que estos tengan.

En este caso Duplicates, Random y Sorted no varian mucho, mientras que Reversed se nota un aumento de tiempo

Cada input tiene bastante variación con respecto al otro, el orden tiene mucha importancia en el funcionamiento de este algoritmo.

Todos se comportan prácticamente igual.

Duplicates demora un poco menos que los demos, pero no es una diferencia de tiempo considerable.

¿Qué algoritmos recomendaría usted, si alguien buscase ordenar de la manera mas rápida posible, para cada uno de los tipos de inputs entregados?

Con respecto a los gráficos anteriores para Duplicates recomendaría QuickSort, para Random recomendaría QuickSort, para Reversed recomendaría QuickSort y para Sorted recomendaría BubbleSort.

1. Descripción de algoritmos:

BubbleSort:

Recorre el arreglo comprobando si el numero en la posición que tiene es mayor al siguiente, en el caso de que se cumpla ,se cambian entre ellos las posición, en cambio si no se cumple, se dejan tal cual estaban.

[Imagen](https://upload.wikimedia.org/wikipedia/commons/c/c8/Bubble-sort-example-300px.gif)

SelectionSort:

Recorre el arreglo tomando la primera posición, después busca cual es el menor de todo números que vienen después, una vez que encuentra al menor de todos esos números, lo cambia con el que había seleccionado al principio y pasa al siguiente numero para repetir el proceso.

[Imagen](https://upload.wikimedia.org/wikipedia/commons/9/94/Selection-Sort-Animation.gif)

InsertionSort:

Recorre el arreglo comprobando si el numero en la posición que tiene es menor a alguno que tenga hacia atrás, si lo encuentra los cambia entre ellos, sino pasa al siguiente numero y repita el proceso

[Imagen](https://upload.wikimedia.org/wikipedia/commons/0/0f/Insertion-sort-example-300px.gif)

MergeSort:

Divide el arreglo en sub-listas, hasta tener sub-listas de longitud 0 ó 1, luego toma dos sub-listas y las ordena, y las une en una sub-lista de dos elementos. Asi hasta que quede una lista ordenada.

[Imagen](https://upload.wikimedia.org/wikipedia/commons/c/cc/Merge-sort-example-300px.gif)

QuickSort:

Se toma un elemento cualquiera, el cual servirá como pivote, se harán dos sub-listas, una la derecha y otra a la izquierda del pivote, la idea es que a su izquierda estén todos los elementos menores al pivote y a la derecha los mayores al pivote. Entonces se recorre la lista de la izquierda dejando todos los menores al pivote y todos los mayores se llevan a la lista de la derecha, una vez que este listo el recorrido se vuelve a elegir otro pivote al azar.

[Imagen](https://upload.wikimedia.org/wikipedia/commons/6/6a/Sorting_quicksort_anim.gif)

1. Resultados experimentales:

Estos fueron los resultados de los benchmarks, ordenados con respecto al algoritmo y el input utilizado.  
  
BenchmarkBubbleSortDuplicates1k-4 1000000 1054 ns/op

BenchmarkBubbleSortDuplicates10k-4 100000 12495 ns/op

BenchmarkBubbleSortDuplicates100k-4 1 21830373385 ns/op

BenchmarkBubbleSortDuplicates1m-4 1 2311573532811 ns/op

BenchmarkBubbleSortRandom1k-4 1000000 1050 ns/op

BenchmarkBubbleSortRandom10k-4 100000 12224 ns/op

BenchmarkBubbleSortRandom100k-4 1 23213561302 ns/op

BenchmarkBubbleSortRandom1m-4 1 2288851467415 ns/op

BenchmarkBubbleSortReversed1k-4 1000000 1031 ns/op

BenchmarkBubbleSortReversed10k-4 100000 11641 ns/op

BenchmarkBubbleSortReversed100k-4 1 13490861538 ns/op

BenchmarkBubbleSortReversed1m-4 1 1485277422389 ns/op

BenchmarkBubbleSortSorted1k-4 1000000 1059 ns/op

BenchmarkBubbleSortSorted10k-4 200000 10469 ns/op

BenchmarkBubbleSortSorted100k-4 20000 102941 ns/op

BenchmarkBubbleSortSorted1m-4 1000 1320387 ns/op

BenchmarkSelectionSortDuplicates1k-4 2000 1191155 ns/op

BenchmarkSelectionSortDuplicates10k-4 10 114842418 ns/op

BenchmarkSelectionSortDuplicates100k-4 1 10436544704 ns/op

BenchmarkSelectionSortDuplicates1m-4 1 1187374191034 ns/op

BenchmarkSelectionSortRandom1k-4 2000 1190107 ns/op

BenchmarkSelectionSortRandom10k-4 10 111791692 ns/op

BenchmarkSelectionSortRandom100k-4 1 10937433041 ns/op

BenchmarkSelectionSortRandom1m-4 1 1050828609878 ns/op

BenchmarkSelectionSortReversed1k-4 2000 1227509 ns/op

BenchmarkSelectionSortReversed10k-4 10 117699053 ns/op

BenchmarkSelectionSortReversed100k-4 1 17419204671 ns/op

BenchmarkSelectionSortReversed1m-4 1 1697593806198 ns/op

BenchmarkSelectionSortSorted1k-4 2000 1191256 ns/op

BenchmarkSelectionSortSorted10k-4 10 109813236 ns/op

BenchmarkSelectionSortSorted100k-4 1 10447357305 ns/op

BenchmarkSelectionSortSorted1m-4 1 1128130407365 ns/op

BenchmarkInsertSortDuplicates1k-4 1000000 2104 ns/op

BenchmarkInsertSortDuplicates10k-4 100000 21507 ns/op

BenchmarkInsertSortDuplicates100k-4 1 4309859975 ns/op

BenchmarkInsertSortDuplicates1m-4 1 530947556736 ns/op

BenchmarkInsertSortRandom1k-4 1000000 2227 ns/op

BenchmarkInsertSortRandom10k-4 100000 22980 ns/op

BenchmarkInsertSortRandom100k-4 1 4233672994 ns/op

BenchmarkInsertSortRandom1m-4 1 433987140463 ns/op

BenchmarkInsertSortReversed1k-4 1000000 2147 ns/op

BenchmarkInsertSortReversed10k-4 50000 22133 ns/op

BenchmarkInsertSortReversed100k-4 1 8632951464 ns/op

BenchmarkInsertSortReversed1m-4 1 888060466031 ns/op

BenchmarkInsertSortSorted1k-4 1000000 2148 ns/op

BenchmarkInsertSortSorted10k-4 100000 20857 ns/op

BenchmarkInsertSortSorted100k-4 5000 211104 ns/op

BenchmarkInsertSortSorted1m-4 500 2425609 ns/op

BenchmarkMergeSortDuplicates1k-4 3000 439587 ns/op

BenchmarkMergeSortDuplicates10k-4 300 5838534 ns/op

BenchmarkMergeSortDuplicates100k-4 20 70437195 ns/op

BenchmarkMergeSortDuplicates1m-4 2 758593732 ns/op

BenchmarkMergeSortRandom1k-4 3000 445676 ns/op

BenchmarkMergeSortRandom10k-4 200 6212734 ns/op

BenchmarkMergeSortRandom100k-4 20 77115909 ns/op

BenchmarkMergeSortRandom1m-4 2 775713433 ns/op

BenchmarkMergeSortReversed1k-4 3000 431126 ns/op

BenchmarkMergeSortReversed10k-4 200 6149602 ns/op

BenchmarkMergeSortReversed100k-4 20 78898597 ns/op

BenchmarkMergeSortReversed1m-4 2 714152559 ns/op

BenchmarkMergeSortSorted1k-4 3000 444914 ns/op

BenchmarkMergeSortSorted10k-4 200 6539488 ns/op

BenchmarkMergeSortSorted100k-4 20 87895403 ns/op

BenchmarkMergeSortSorted1m-4 2 718006929 ns/op

BenchmarkQuickSortDuplicates1k-4 10000 189181 ns/op

BenchmarkQuickSortDuplicates10k-4 500 3214889 ns/op

BenchmarkQuickSortDuplicates100k-4 30 34086411 ns/op

BenchmarkQuickSortDuplicates1m-4 3 346145146 ns/op

BenchmarkQuickSortRandom1k-4 3000 409222 ns/op

BenchmarkQuickSortRandom10k-4 300 6234956 ns/op

BenchmarkQuickSortRandom100k-4 20 57418781 ns/op

BenchmarkQuickSortRandom1m-4 2 543186101 ns/op

BenchmarkQuickSortReversed1k-4 3000 423054 ns/op

BenchmarkQuickSortReversed10k-4 200 6254363 ns/op

BenchmarkQuickSortReversed100k-4 20 55414411 ns/op

BenchmarkQuickSortReversed1m-4 2 511809326 ns/op

BenchmarkQuickSortSorted1k-4 3000 452032 ns/op

BenchmarkQuickSortSorted10k-4 200 6489233 ns/op

BenchmarkQuickSortSorted100k-4 20 58379598 ns/op

BenchmarkQuickSortSorted1m-4 2 513191318 ns/op

1. Conclusión:

En conclusión con este informe se ve el aprendizaje de algunos de los algoritmos de ordenamiento, aparte de poder comparar los algoritmos entre si y la importancia de los elementos dentro del arreglo que va ser utilizado para ordenar.