UNIVERSIDAD SIMÓN BOLÍVAR

Integrantes:

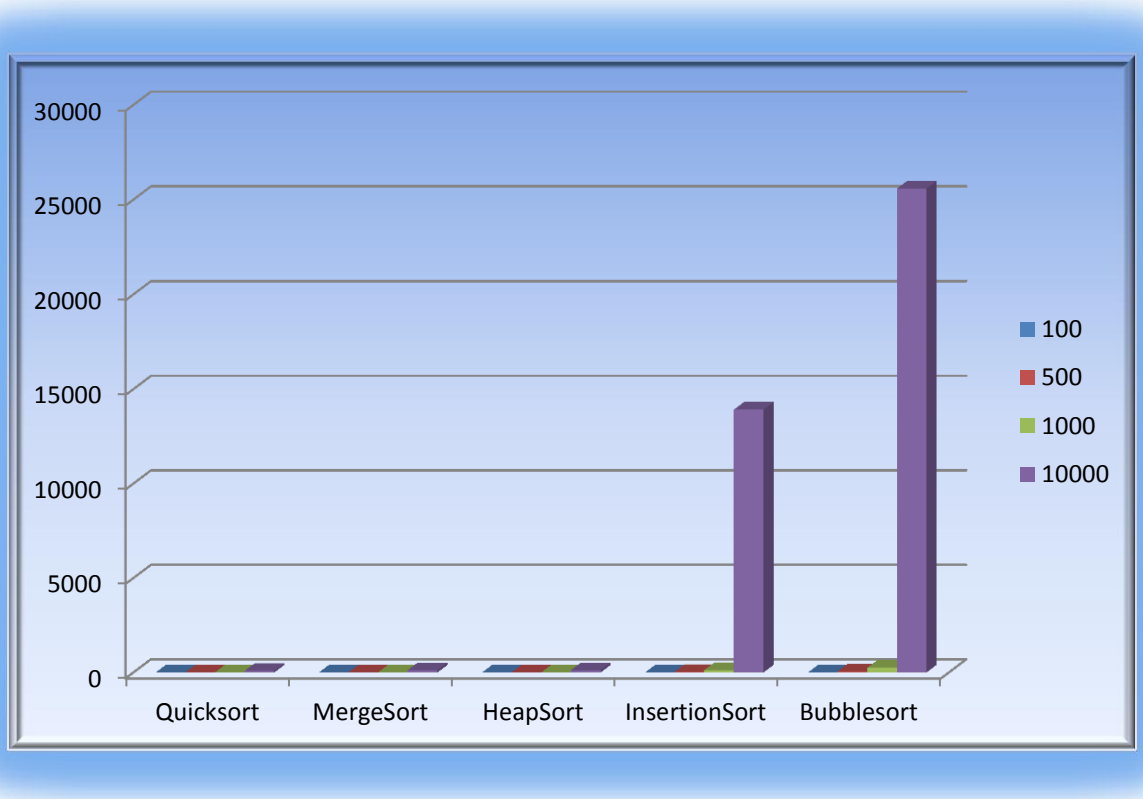
Fernando González 08-10464

Bruno Colmenares 12 -10551

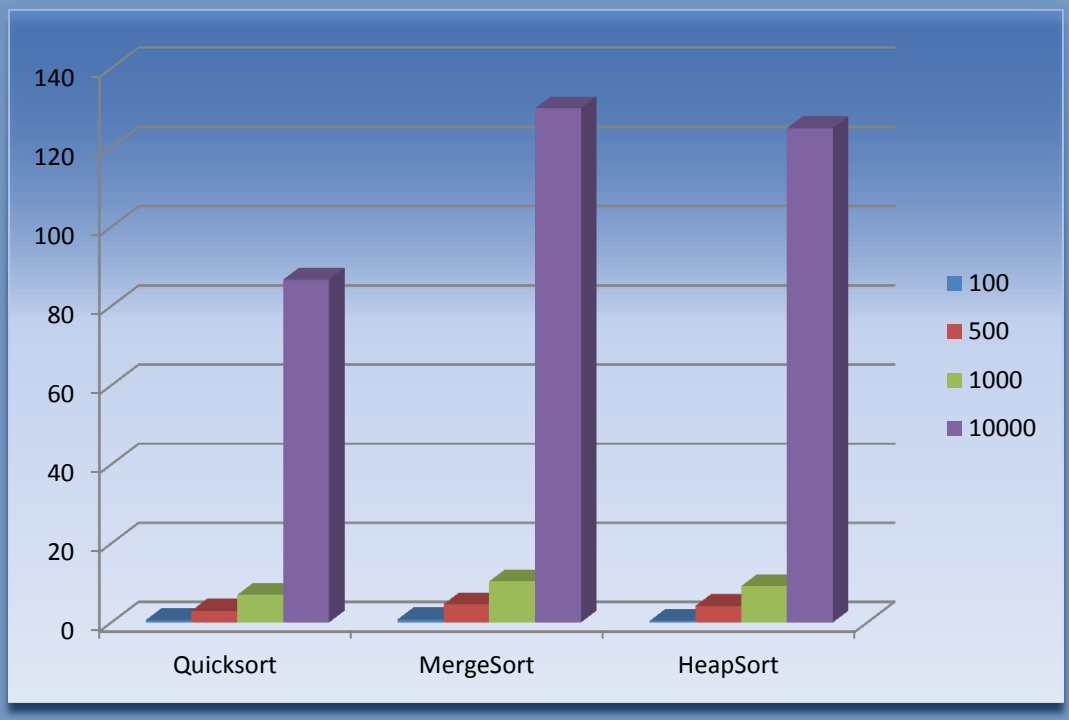
Tabla 1

	100	*500*	*1000*	*10000*
Quicksort	*0.6*	*3*	*7*	*86.66*
MergeSort	*0.8*	*4.6*	*10.41*	*129.89*
HeapSort	*0.4*	*4.2*	*9.21*	*124.88*
InsertionSort	*1.4*	*31.82*	*132.09*	*13917.64*
BubbleSort	*2.4*	*59.44*	*247.57*	*25582.38*

Grafica 1



Grafica 2



Para estas corridas fueron utilizados alfabetos de caracteres aleatorios de cardinalidad 7. Con lo expresado en la gráfica de los algoritmos que se corrieron en la Tabla 1, usando los resultados de tiempo de ejecución promedio, en milisegundos, de las 5 corridas por algoritmo de ordenamiento a cada tamaño del arreglo dado. Podemos notar que los algoritmos clásicos de ordenamiento (Insertionsort, Bubblesort) son mucho más lentos en tiempo de ejecución que los algoritmos (Quicksort, Mergesort, Heapsort) que utilizan la técnica de divide y vencerás para el desarrollo de su implementación. Esto se debe a que los algoritmos clásicos de ordenamiento se basan en simples iteraciones para realizar el desarrollo de su implementación, ocasionando que el problema de ordenar se tenga que resolver mediante la evaluación y ordenamiento de cada elemento de la lista se uno a uno. En cambio los algoritmos que utilizan la técnica de divide y vencerás, que consiste en dividir un problema grande en varios pequeños para que sea más fácil resolverlos. Mediante llamadas recursivas a sí mismos, haciendo posible que el tiempo de ejecución sea más rápido. Aprovechándose de que en esos casos el tiempo que usan para dividir y combinar el problema grande, ordenar en los subproblemas pequeños; es menor al de evaluar y ordenar elemento por elemento en un solo problema.

En el caso de la Tabla 1, podemos notar que el tiempo de corrida del Insertionsort es notablemente más rápido que los Bubblesort. Esto se debe a que el Insertionsort tiene bajo costo operativo y aunque tanto el Insertionsort como el Bubblesort su complejidad es adaptable a O(n) en los casos en que la secuencia está casi ordenada, los ordenamientos por burbuja requieren al menos dos pases a través de los datos de la secuencia para poderlos ordenar, mientras que el Insertionsort requiere algo parecido a un pase a través de los datos de la secuencia.

En el caso de la Tabla 1, podemos analizar los algoritmos de ordenamiento recursivo. Ya que dichos algoritmos son de orden NLog(N), y los algoritmos clásicos son de orden N^2, haciendo que en la Grafica 1 no se logre apreciar con claridad la comparación de los tiempos de ejecución de los algoritmos recursivos, puesto a el tiempo de corrida de estos es muy pequeño en comparación con los N^2 y a la vez se intenta comparar los tiempos de ejecución con los algoritmos de ordenamiento clásicos. En la Tabla 1 podemos apreciar que el algoritmo de ordenamiento más veloz en cuanto a tiempo de ejecución es el Quicksort, aun si este en el peor de los casos puede alcanzar un tiempo de ejecución con orden N^2 debido a que el procedimiento de particionado que posee produce una recursividad equilibrada cuando hay muchos valores iguales al pivote, dando garantías probabilísticas de O(Nlog(N)) en tiempo de ejecución y O(Lg(N)) en cuanto a espacio de todas las entradas, a diferencia del Mergesort que hace la partición dividiendo siempre la secuencia en dos y el Heapsort que hace la partición siguiendo un esquema de árbol binario, produciéndose en los dos una recursividad no equilibrada que se ve de manera más evidente en el Heapsort, causando que el ordenamiento requiera O(N) más espacio para la pila de la recursividad o en el peor de los casos.

Por otra parte, se puede observar un giro inesperado en el caso en el cual utilizamos un Árbol binario para generar una lista en preOrden y someterla a los algoritmos de ordenamiento, podemos apreciar que los algoritmos N^2 aumentan dramáticamente su efectividad, esto se debe a que, una vez se construye el árbol binario con la relación {HijoIzq <= Padre < HijoDer} y de allí ser creada una lista en PreOrden, tenemos que la lista mantendrá un cierto orden en el cual habrá un menor número probabilístico de elementos a ordenar dentro de la lista y la comparación de elemento por elemento es una elección sabia para ordenar este tipo de arreglos, en la Tabla 2 se pueden apreciar los resultados.

Tabla 2

	100	*500*	*1000*	*10000*
Quicksort	*0.2*	*3.2*	*7.01*	*82.45*
MergeSort	*0.8*	*4.6*	*10.01*	*118.28*
HeapSort	*0.8*	*4.2*	*9.41*	*128.08*
InsertionSort	*0.2*	*1.8*	*4.4*	*54.23*
BubbleSort	*0.2*	*1.8*	*5*	*60.04*