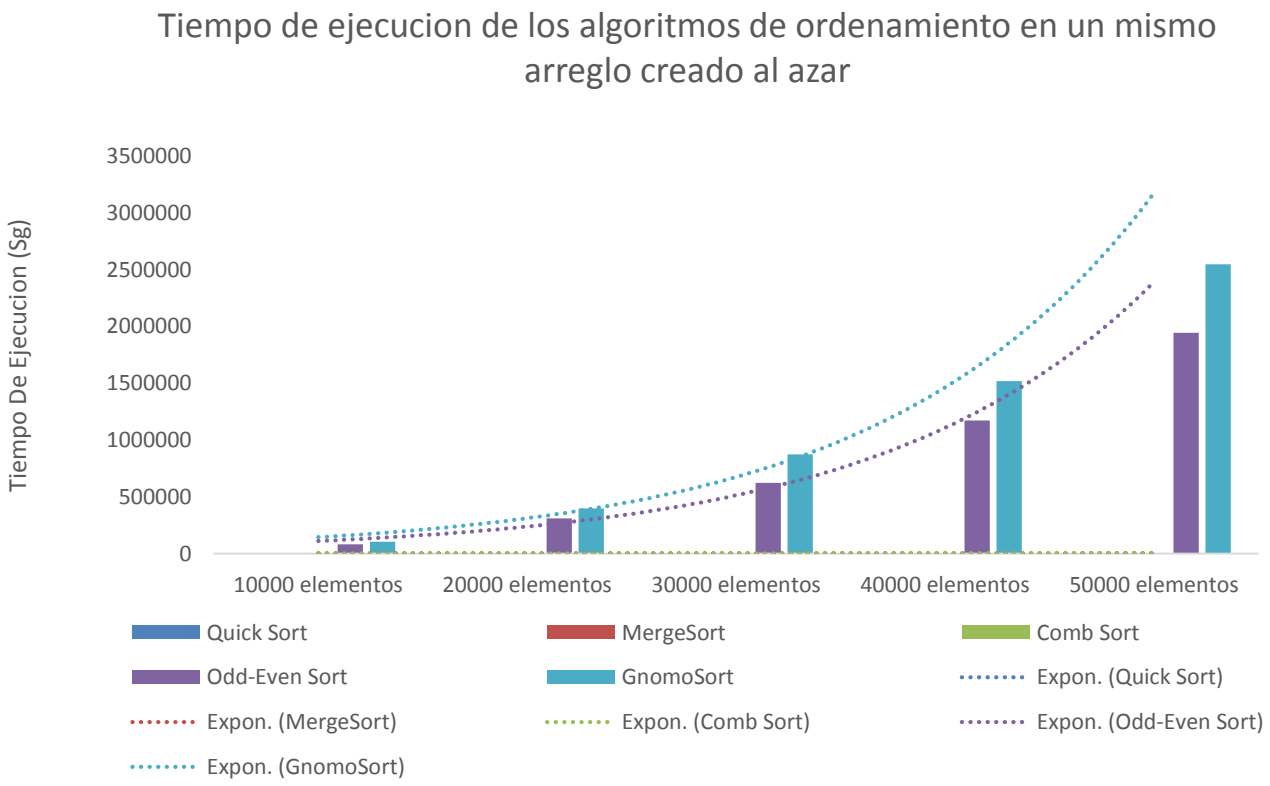
INFORME PROYECTO N.3

1. Comparación de algoritmos de Ordenamiento

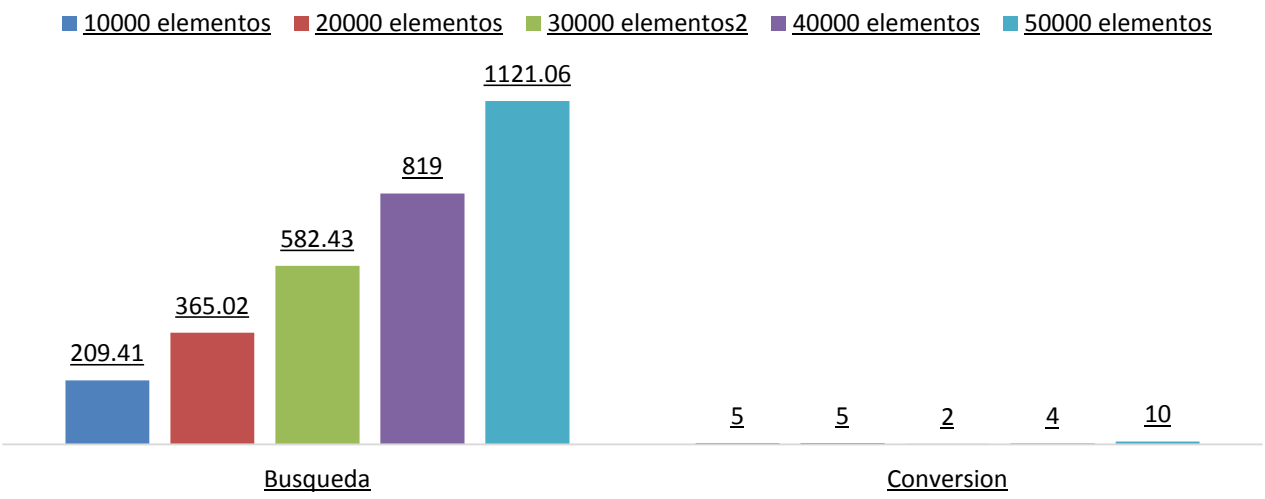
Num. de elementos	Quick Sort	MergeSort	Comb Sort	Odd-Even Sort	Gnomo Sort
10000	340.22	541.83	834.85	80693.42	103595.33
20000	627.84	962.65	1420.88	309657.51	398039.57
30000	870.05	1336.48	2237.93	622878.63	873312.92
40000	1224.60	1833.00	2776.81	1171983.24	1518342.85
50000	1497.60	2371.20	4274.41	1944943.20	2548055.57

2. Medición de construcción y búsqueda

Num. De elementos	Conversión	Búsqueda
10000	209.41	5.00
20000	365.02	5.00
30000	582.43	2.00
40000	819.00	4.00
50000	1121.06	10.00



Tiempo de conversion y busqueda del arbol binario



Análisis

Los resultados reflejan la eficiencia de los algoritmos ya que calculamos diferentes casos y calculamos un promedio entre los casos y una tendencia de cada método. Los métodos en orden de eficiente serían los siguientes:

1. Quicksort ($O(n \cdot \log n)$)
2. Mergesort ($O(n \log n)$.)
3. Combsort ($\Omega(n^2)$)
4. Oddeven-sort (O(n^2))
5. Gnome-sort ($O(n^2)$.)

El Gnome-Sort resulta ser el menos eficiente mientras el quicksort el más eficiente, esto se debe a que en el quicksort el arreglo se divide en dos subarreglos ordenando los mayores al pivote del lado derecho y del lado izquierdo los menores, aplicando esto de manera recursiva en pequeños grupos a través del arreglo y este quedara ordenado haciendo muchas pequeñas tareas pero que cada una requiere poco esfuerzo por parte del ordenador. El Gnome-sort en cambio en cada recursión recorrerá un arreglo más y más grande, haciendo que las recursiones sean más costosas a medida que progrese, en un arreglo de tamaño pequeño este aumento de recursos es despreciable pero el tiempo de corrida aumentara exponencialmente mientras tenemos más y más arreglos. Con el merge sort tenemos una eficiencia parecida a la del merge sort y se debe a que al igual que en el quick sort su naturaleza es de dividir el arreglo en partes más pequeños y aplicar pequeños procesos de ordenamiento que al terminar todas las recursiones nos dejan un arreglo ordenado. En el caso del combsort su tiempo de corrida es alto ya que este algoritmo recorre pequeñas partes del arreglo en su inicio pero luego va aumentando de tamaño, esto con arreglos grandes aumenta el tiempo de corrida de manera exponencial porque mientras más largo sea el arreglo más recursiones largas habrán pero sus recorridos son más cortos que los de gnome sort. Con el oddeven-sort tenemos la repetición de pequeños algoritmos de ordenamiento y de mezcla por todo el arreglo, resultando en menos tiempo de corrida que el oddeven y gnome pero mayor que los demás métodos.