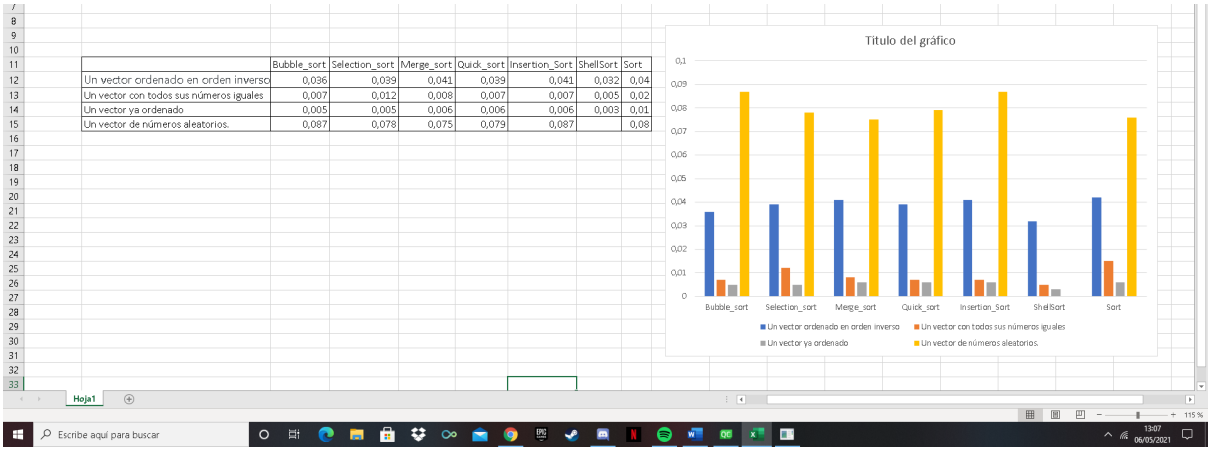
Una memoria donde se muestran los resultados obtenidos y un análisis de los mismos:



	Bubble_sort	Selection_sort	Merge_sort	Quick_sort	Insertion_Sort	ShellSort	Sort
Un vector ordenado en orden inverso	0,036	0,039	0,041	0,039	0,041	0,032	0,04
Un vector con todos sus números iguales	0,007	0,012	0,008	0,007	0,007	0,005	0,02
Un vector ya ordenado	0,005	0,005	0,006	0,006	0,006	0,003	0,01
Un vector de números aleatorios.	0,087	0,078	0,075	0,079	0,087		0,08

¿Se corresponden los tiempos obtenidos con los esperados según la complejidad teórica?:

En algunos casos sí, por ejemplo en el caso de la burbuja, el método de selección etc. Pero el método de shell sort me ha sorprendido porque es el más rápido de todos los métodos de ordenación.

¿Los resultados son diferentes según la ordenación inicial del vector?:

Si, ya que el método de ordenación funciona de una cierta manera y no funcionan iguales.

Una memoria en la que se describa el problema que se pretende resolver con el código, se describa y justifique la opción elegida para resolverlo, así como los algoritmos y estructuras de datos utilizados:

El código pretende ordenar de 4 formas distintas los vectores, las opciones que he elegido para resolver son los siguientes metodos de ordenacion:

- Bubble Sort
- Selection Sort
- Merge Sort
- Quick Sort
- Insertion Sort

- Shell Sort
- Sort

BUBBLE SORT: Dado un vector con n elementos, se realizan n-1 pasadas comparando valores adyacentes, si dos valores no están ordenados, se intercambian.

SELECTION SORT: Dado un vector con n elementos, se divide en una parte ordenada (izquierda) y una desordenada (derecha), se realizan n-1 pasadas en la parte derecha extrayendo el menor valor e insertándose en la parte izquierda.

MERGE SORT: dado un vector con n elementos, lo divido en dos mitades y aplicó merge_sort sobre esas mitades. Después fusionar/mergeo las dos listas ordenadas.

Pasos para resolver el merge Sort:

1. Una lista más corta se ordena más rápidamente que una lista larga.

2. Es más sencillo construir una lista ordenada a partir de dos listas ordenadas que a partir de dos listas desordenadas.

QUICK SORT: Dado un vector con n elementos, elijo un pivote y agrupó los elementos mayores a un lado y los menores al otro, repito quick_sort sobre cada una de las partes.

Pasos para resolver el Quick Sort:

1. Es más rápido ordenar dos listas pequeñas que una grande.

2. Una lista está ordenada cuando todos los elementos más pequeños de uno dado están a la izquierda de todos los elementos mayores (para cada elemento).

INSERTION SORT: La matriz se divide virtualmente en una parte ordenada y otra sin clasificar. Los valores de la parte sin clasificar se seleccionan y colocan en la posición correcta en la parte clasificada.

Los usos del Insertion Sort: Usos: la ordenación por inserción se utiliza cuando el número de elementos es pequeño. También puede ser útil cuando la matriz de entrada está casi ordenada, solo unos pocos elementos están fuera de lugar en una matriz grande completa.

SHELL SORT: El método de ordenamiento Shell consiste en dividir el arreglo (o la lista de elementos) en intervalos (o bloques) de varios elementos para organizarlos después por medio del ordenamiento de inserción directa. El proceso se repite, pero con intervalos cada vez más pequeños, de tal manera que al final, el ordenamiento se haga en un intervalo de una sola posición.

SORT: Es una función genérica en la biblioteca estándar de C ++ para realizar ordenación por comparación

Su función es comparar elementos de la primera posición hasta la final, es decir, recorrer del principio hasta el final. Los elementos con menor valor estarán al principio y los que tengan más valor al final, y en el caso de que sean caracteres sería como el abecedario.