	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Jesus Cruz Navarro
Asignatura:	Estructura de Datos y Algoritmos II
Grupo:	Grupo 1
No de Práctica(s):	Práctica 2 – Algoritmos de ordenamiento. Parte II
Integrante(s):	Edwin Jaret Santiago Díaz
No. de Equipo de cómputo empleado:	22
No. de Lista o Brigada:	22
Semestre:	2022 - 2
Fecha de entrega:	21 febrero 2022
Observaciones:	

CALIFICACIÓN: _____

Algoritmos de ordenamiento. Parte II

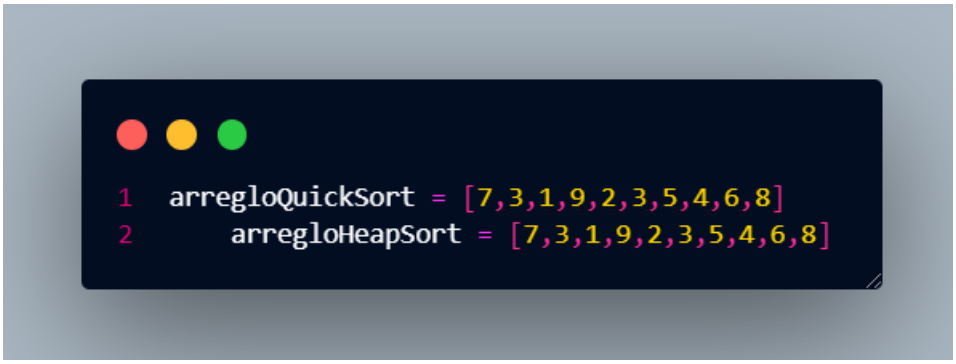
Objetivos

1. Implementar los algoritmos QuickSort, QuickSort Aleatorio y HeapSort para ordenar colecciones de datos.
2. Comparar los tiempos de ejecución de los algoritmos antes mencionados.

Desarrollo

Los algoritmos **QuickSort** y **HeapSort** se encuentran en el archivo **AlgoritmosII** en donde cada algoritmo tiene su función. El algoritmo **QuickSort** recibe como parámetro la lista de los números enteros y el algoritmo **HeapSort** recibe como parámetro la lista de números enteros, el índice donde inicia la lista y la longitud de la lista. Los algoritmos se implementaron gracias a la explicación del profesor y a la ayuda de la guía del laboratorio.

Se ejecutan los algoritmos para ordenar una lista de números aleatorios enteros con un rango de 0 a 10 y se obtienen los tiempos que tarda cada algoritmo. Esto fueron los resultados:



```
1  arregloQuickSort = [7,3,1,9,2,3,5,4,6,8]
2      arregloHeapSort = [7,3,1,9,2,3,5,4,6,8]
```

```
Lista ordenada por QuickSort:  [1, 2, 3, 3, 4, 5, 6, 7, 8, 9]
El algoritmo tardó: 0.00002880
```

```
Lista ordenada por HeapSort:  [1, 2, 3, 3, 4, 5, 6, 7, 8, 9]
El algoritmo tardó: 0.00001260
```

Para obtener resultados más notorios en los tiempos de ejecución se pone a prueba los algoritmos con arreglos de números más grandes con 1000, 3000, 4000 y 5000 datos, además, se pondrán a prueba los algoritmos en el mejor de los casos (lista ordenada ascendente), en el caso promedio (lista aleatoria) y en el peor de los casos (lista ordenada descendiente) y por cada tamaño de arreglo el programa se ejecutará 3 veces para obtener el promedio del tiempo que tardan.

Estos fueron los resultados:

1,000	Tiempo promedio
¡Caso Promedio! Tiempo QuickSort : 0.029010024002 Tiempo HeapSort : 0.081202807996	**Caso promedio:** QuickSort = 0.016471 [s] HeapSort = 0.065715 [s]

¡Caso Promedio! Tiempo QuickSort : 0.006939409999 Tiempo HeapSort : 0.028527648996 ¡Caso Promedio! Tiempo QuickSort : 0.013464083997 Tiempo HeapSort : 0.087417961004 ¡Mejor de los casos! Tiempo QuickSort : 0.027782984995 Tiempo HeapSort : 0.679101367001 ¡Mejor de los casos! Tiempo QuickSort : 0.012410017000 Tiempo HeapSort : 0.708509500997 ¡Mejor de los casos! Tiempo QuickSort : 0.007654071000 Tiempo HeapSort : 0.399006591004 ¡Peor de los casos! Tiempo QuickSort : 0.02729605 Tiempo HeapSort : 0.16327981 ¡Peor de los casos! Tiempo QuickSort : 0.07854700 Tiempo HeapSort : 0.29766691 ¡Peor de los casos! Tiempo QuickSort : 0.00619741 Tiempo HeapSort : 0.19724518	**Mejor de los casos:** QuickSort = 0.015948 [s] HeapSort = 0.595536 [s] **Peor de los casos:** QuickSort = 0.037344 [s] HeapSort = 0.219396 [s]
3,000	
¡Caso Promedio! Tiempo QuickSort : 0.081353514000 Tiempo HeapSort : 0.323050198000 ¡Caso Promedio! Tiempo QuickSort : 0.206576858996 Tiempo HeapSort : 0.992294964031 ¡Caso Promedio! Tiempo QuickSort : 0.068813823001 Tiempo HeapSort : 0.320862450000 ¡Mejor de los casos! Tiempo QuickSort : 0.083426180000 Tiempo HeapSort : 4.486180549000	**Caso promedio:** QuickSort = 0.118914 [s] HeapSort = 0.545402 [s] **Mejor de los casos:** QuickSort =0.156481 [s] HeapSort = 4.561465 [s]

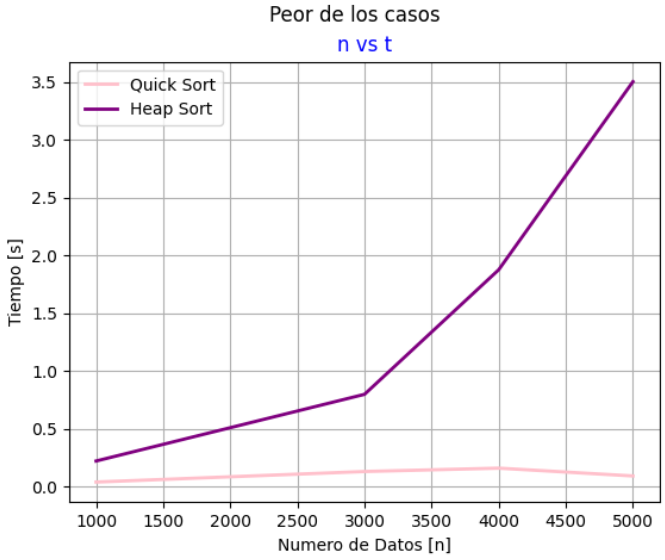
¡Mejor de los casos! Tiempo QuickSort : 0.302648792975 Tiempo HeapSort : 4.809449300985 ¡Mejor de los casos! Tiempo QuickSort : 0.083369651000 Tiempo HeapSort : 4.388767720000 ¡Peor de los casos! Tiempo QuickSort : 0.02140595 Tiempo HeapSort : 0.57757112 ¡Peor de los casos! Tiempo QuickSort : 0.08559620 Tiempo HeapSort : 0.91487675 ¡Peor de los casos! Tiempo QuickSort : 0.02152137 Tiempo HeapSort : 0.89660714	Peor de los casos: QuickSort = 0.128522 [s] HeapSort = 0.796351 [s]
4,000	
¡Caso Promedio! Tiempo QuickSort : 0.047098024050 Tiempo HeapSort : 1.000979411998 ¡Caso Promedio! Tiempo QuickSort : 0.072714260896 Tiempo HeapSort : 0.812710572034 ¡Caso Promedio! Tiempo QuickSort : 0.073625147000 Tiempo HeapSort : 1.755812464000 ¡Mejor de los casos! Tiempo QuickSort : 0.080862738076 Tiempo HeapSort : 8.520097661996 ¡Mejor de los casos! Tiempo QuickSort : 0.180402289028 Tiempo HeapSort : 8.174784422969 ¡Mejor de los casos! Tiempo QuickSort : 0.478978973000 Tiempo HeapSort : 29.032159721000 ¡Peor de los casos! Tiempo QuickSort : 0.08455713 Tiempo HeapSort : 0.80180250 ¡Peor de los casos! Tiempo QuickSort : 0.02593621 Tiempo HeapSort : 1.30837210	**Caso promedio:** QuickSort = 0.064479 [s] HeapSort = 1.189833 [s] **Mejor de los casos:** QuickSort = 0.246747 [s] HeapSort = 15.242346 [s] **Peor de los casos:** QuickSort = 0.157660 [s] HeapSort = 1.874456 [s]

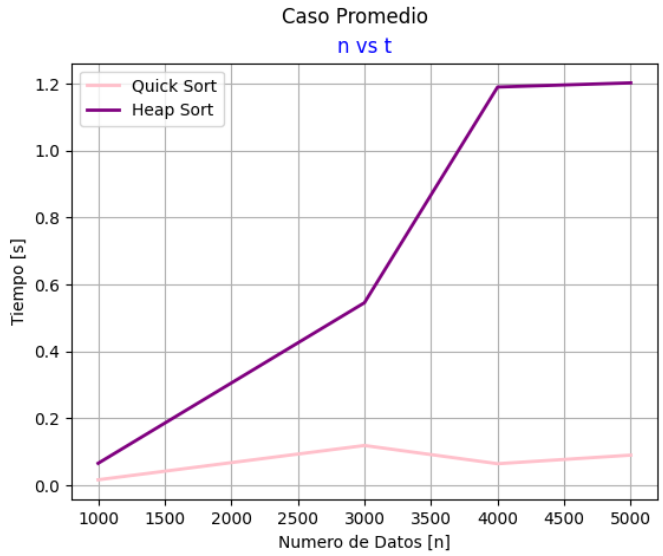
¡Peor de los casos! Tiempo QuickSort : 0.36248790 Tiempo HeapSort : 3.51319553	
5,000	
¡Caso Promedio! Tiempo QuickSort : 0.089242885006 Tiempo HeapSort : 1.650132191018 ¡Caso Promedio! Tiempo QuickSort : 0.097994641998 Tiempo HeapSort : 1.073508668000 ¡Caso Promedio! Tiempo QuickSort : 0.082916058018 Tiempo HeapSort : 0.882928531035 ¡Mejor de los casos! Tiempo QuickSort : 0.152582070092 Tiempo HeapSort : 16.075495754951 ¡Mejor de los casos! Tiempo QuickSort : 0.104814225000 Tiempo HeapSort : 21.198526495005 ¡Mejor de los casos! Tiempo QuickSort : 0.097309555043 Tiempo HeapSort : 12.932560993009 ¡Peor de los casos! Tiempo QuickSort : 0.19189037 Tiempo HeapSort : 2.80520925 ¡Peor de los casos! Tiempo QuickSort : 0.09909128 Tiempo HeapSort : 4.03437678 ¡Peor de los casos! Tiempo QuickSort : 0.09856890 Tiempo HeapSort : 3.66543807	**Caso promedio:** QuickSort = 0.090050 [s] HeapSort = 1.202189 [s] **Mejor de los casos:** QuickSort = 0.118235 [s] HeapSort = 16.735227 [s] Peor de los casos: QuickSort = 0.129849 [s] HeapSort = 3.501674 [s]

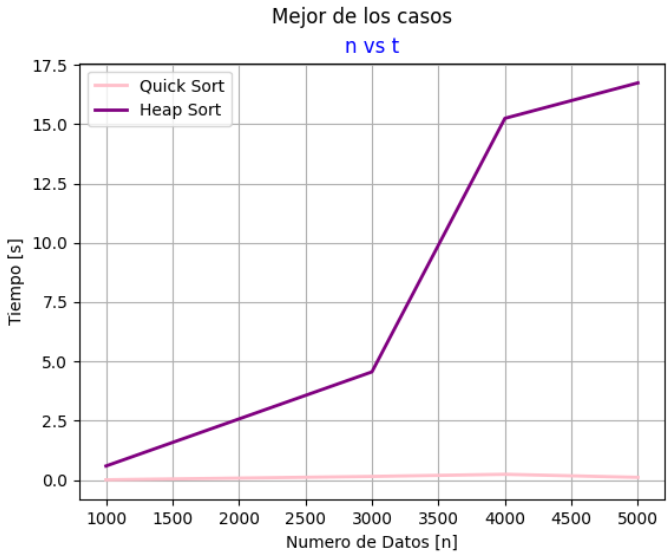
Los algoritmos no pudieron ordenar listas de números mayores de 5,000 datos, por eso, en la práctica se utilizó 1,000, 3,000, 4,000 y 5,000 datos. Tuve este problema en mi IDE (Visual Studio Code) y en un IDE online donde se ejecuta Python. En ambos IDE aumenté el límite de regresiones que puede hacer Python, pero no funcionó.

Para que el algoritmo **QuickSort** sea más óptimo, se agrega 2 líneas de código en la función *Particionar()*. La primera, escoge un pivote al azar y la segunda línea, este pivote es intercambiado con el último elemento de la lista. Después de haber implementado estas 2 líneas se notó el cambio en el tiempo de ejecución del programa.

Con los datos obtenidos de los 3 casos previamente implementados se realizaron 3 gráficas. Cada gráfica corresponde a cada caso y representa el tiempo que tardan el algoritmo **QuickSort** (línea rosa) y **HeapSort** (línea morada).







La variable auxiliar **i** en la función Particionar del algoritmo QuickSort inicia fuera del arreglo (p-1) debido a que cuando se encuentre un numero menor al pivote se moverá hacia el lado "izquierdo" del arreglo, aumentará el valor de la variable i (1 unidad) y intercambiará el elemento del índice **i** con el elemento del índice **j,** pero si fuera en el caso de que se ejecuta por primera vez el algoritmo, después de que i aumente su valor, valdrá 0 y sería el primer índice del arreglo en donde intercambiará el elemento del índice **j** con el **i.** En el algoritmo **HeapSort**, es necesario iterar la función **MaxHeapify** desde atrás hacia adelante debido a que, como es un árbol (estructura de datos) que antes era una lista, se va a comparar 3 nodos (el padre, el hijo izquierdo y el hijo derecho) y el hijo derecho será el último nodo en ser llenado en que pueda recibir otro nodo, esto es, que probable que tenga menos padres encima de ese nodo y será más fácil encontrar el elemento mayor y ordenar todos los nodos de la parte derecha y luego de la parte izquierda.

Conclusiones

Los algoritmos **QuickSort y HeapSort** son más rápidos que **BubbleSort y BubbleSortOptimizad,** sin embargo, el algoritmo más rápido por el momento ha sido **QuickSort**. En las gráficas se muestra que el tiempo que le tomó a **QuickSort** siempre fue bajo de 1 segundo al resolver 1000 datos hasta los 5000 datos, mientras que **HeapSort** superó el segundo en los 3 casos. Así mismo, **QuickSort** es más fácil de implementar en el código.

La complejidad computacional de **QuickSort** en:

- En el mejor caso es de **O(n*logn)**
- En el peor caso es de **O(n^2)**
- En el caso promedio es de **O(n*logn**)

La complejidad computacional de **HeapSort** en:

- En el mejor caso es de **O(n*logn)**
- En el peor caso es de **O(n^2)**
- En el caso promedio es de **O(n*logn)**