	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 III
Grupo:	Grupo 1
No de Práctica(s):	Práctica 3 – Algoritmos de ordenamiento. Parte III
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:	1 marzo 2022
Observaciones:	

CALIFICACIÓN: _____

Algoritmos de ordenamiento. Parte III

Objetivos

1. Implementar los algoritmos CountingSort y RadixSort para ordenar colecciones de números enteros positivos.
2. Compara los tiempos de ejecución de los algoritmos antes mencionados.
3. Modificar el algoritmo de RadixSort para ordenar cadenas de manera lexicográfica.

Desarrollo

Programa 1

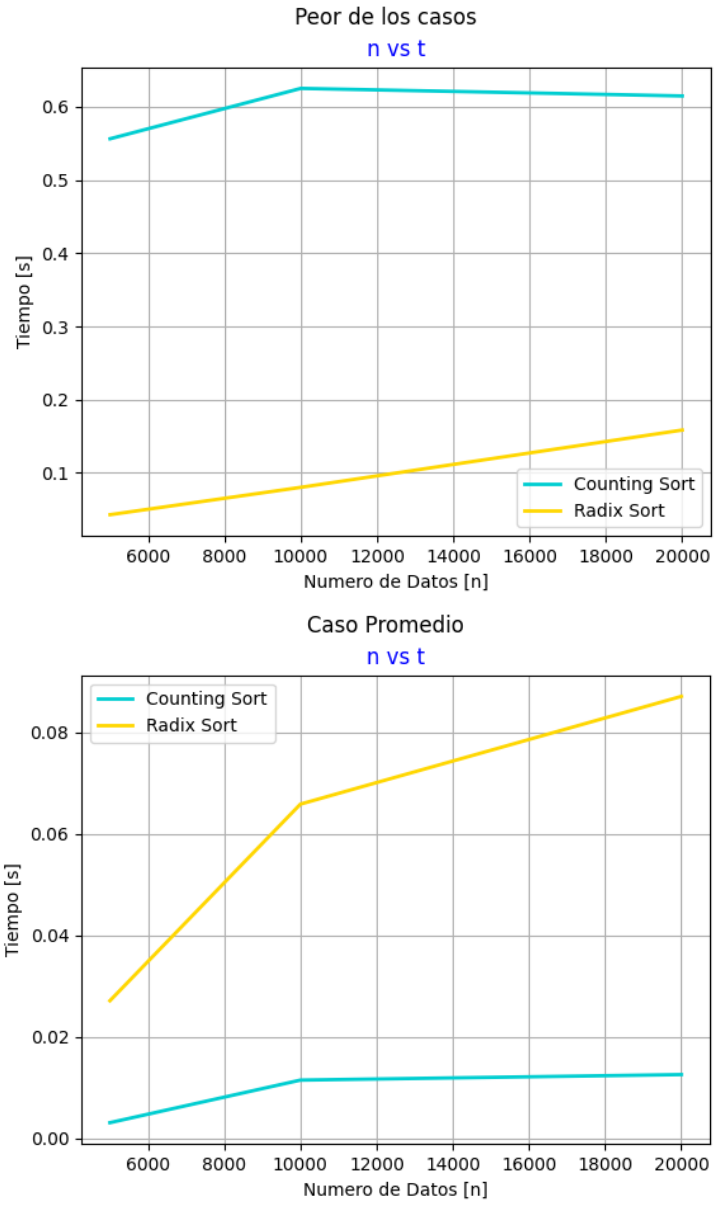
Se desarrolló en el programa con el nombre "*algoritmosIII.py*" en el cual los algoritmos de ordenamiento CountingSort y RadixSort son desarrollados, después, se obtienen los tiempos de ejecución de cada algoritmo (se imprimen) para tamaños de listas de 5000, 10000 y 20000 datos y para el peor de los casos, mejor de los casos y el caso promedio, se ejecutará 3 veces cada caso y cada tamaño de lista para obtener un promedio del tiempo que tardan los algoritmos.

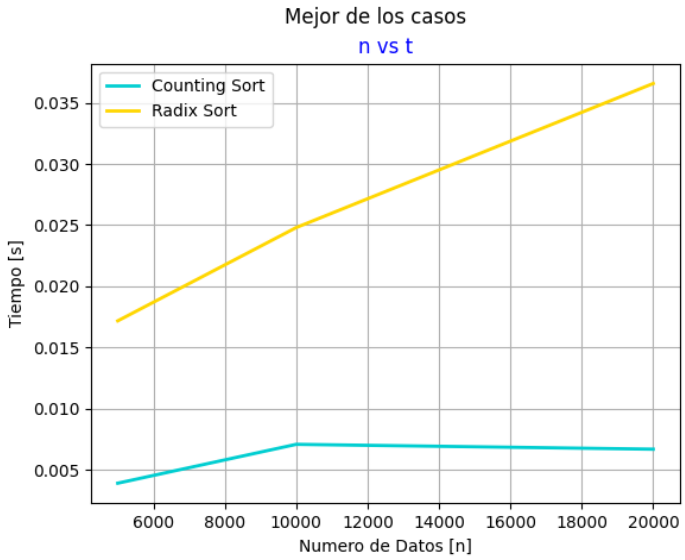
5,000	Tiempo promedio
Caso Promedio El tiempo que tarda CountingSort es de 0.00509560 El tiempo que tarda RadixSort es de 0.04559210 Caso Promedio El tiempo que tarda CountingSort es de 0.00226660 El tiempo que tarda RadixSort es de 0.02016400 Caso Promedio El tiempo que tarda CountingSort es de 0.00180150 El tiempo que tarda RadixSort es de 0.01556870	**Caso promedio:** CoutningSort = 0.003054 [s] RadixSort = 0.027108 [s]
Mejor Caso El tiempo que tarda CountingSort es de 0.00662200 El tiempo que tarda RadixSort es de 0.03255350 Mejor Caso El tiempo que tarda CountingSort es de 0.00316730 El tiempo que tarda RadixSort es de 0.00827050 Mejor Caso El tiempo que tarda CountingSort es de 0.00193540 El tiempo que tarda RadixSort es de 0.01071130	**Mejor de los casos:** CoutningSort = 0.003908 [s] RadixSort = 0.017178 [s]
Peor de los casos El tiempo que tarda CountingSort es de 0.54198900 El tiempo que tarda RadixSort es de 0.03848550 Peor de los casos El tiempo que tarda CountingSort es de 0.61607810 El tiempo que tarda RadixSort es de 0.03725760 Peor de los casos El tiempo que tarda CountingSort es de 0.51081740 El tiempo que tarda RadixSort es de 0.05278480	**Peor de los casos:** CoutningSort = 0.556294 [s] RadixSort = 0.042842 [s]

10,000	
Caso Promedio El tiempo que tarda CountingSort es de 0.00710140 El tiempo que tarda RadixSort es de 0.06206660 Caso Promedio El tiempo que tarda CountingSort es de 0.01145270 El tiempo que tarda RadixSort es de 0.08981960 Caso Promedio El tiempo que tarda CountingSort es de 0.01580320 El tiempo que tarda RadixSort es de 0.10069360 Mejor Caso El tiempo que tarda CountingSort es de 0.00722200 El tiempo que tarda RadixSort es de 0.02822530 Mejor Caso El tiempo que tarda CountingSort es de 0.00903870 El tiempo que tarda RadixSort es de 0.02377000 Mejor Caso El tiempo que tarda CountingSort es de 0.00499370 El tiempo que tarda RadixSort es de 0.02241710 Peor de los casos El tiempo que tarda CountingSort es de 0.52346130 El tiempo que tarda RadixSort es de 0.06532240 Peor de los casos El tiempo que tarda CountingSort es de 0.75943660 El tiempo que tarda RadixSort es de 0.09817890 Peor de los casos El tiempo que tarda CountingSort es de 0.59245840 El tiempo que tarda RadixSort es de 0.07711770	**Caso promedio:** CoutningSort = 0.011452 [s] RadixSort = 0.065871 [s] **Mejor de los casos:** CoutningSort = 0.007084 [s] RadixSort = 0.024804 [s] Peor de los casos: CoutningSort = 0.625118 [s] RadixSort = 0.080205 [s]
20,000	
Caso Promedio El tiempo que tarda CountingSort es de 0.00652850 El tiempo que tarda RadixSort es de 0.05331220 Caso Promedio El tiempo que tarda CountingSort es de 0.02206620 El tiempo que tarda RadixSort es de 0.13957280 Caso Promedio El tiempo que tarda CountingSort es de 0.00903320 El tiempo que tarda RadixSort es de 0.06845730 Mejor Caso El tiempo que tarda CountingSort es de 0.00682200 El tiempo que tarda RadixSort es de 0.03483380 Mejor Caso El tiempo que tarda CountingSort es de 0.00622740 El tiempo que tarda RadixSort es de 0.03967880 Mejor Caso El tiempo que tarda CountingSort es de 0.00701970 El tiempo que tarda RadixSort es de 0.03522490 Peor de los casos El tiempo que tarda CountingSort es de 0.73609110 El tiempo que tarda RadixSort es de 0.19372160	**Caso promedio:** CoutningSort = 0.012542 [s] RadixSort = 0.087120 [s] **Mejor de los casos:** CoutningSort = 0.006689 [s] RadixSort = 0.036579 [s]

Peor de los casos El tiempo que tarda CountingSort es de 0.64505140 El tiempo que tarda RadixSort es de 0.13486190 Peor de los casos El tiempo que tarda CountingSort es de 0.46329200 El tiempo que tarda RadixSort es de 0.14674400	**Peor de los casos:** CoutningSort = 0.614811 [s] RadixSort = 0.158442[s]

Se grafica el tiempo que les tomó cada algoritmo al ordenar 5000, 10000 y 20000 números.





¿Por qué el último for de CountingSort se debe realizar de atrás para adelante?

Porque hace que el siguiente elemento de entrada con un valor igual a A[j], cuando existe, vaya a a posición inmediatamente anterior a A[j] en la matriz de salida.

Programa 2

El programa 2 con el nombre de *"ordenamiento_cadenas.py"* ordena un arreglo de palabras, números o letras por el método RadixSort, se modificó al algoritmo visto en clase y éste, cambia las letras, números o caracteres a su número correspondiente al código ASCII y ordenará la lista conforme a los números. No importa el tamaño de la cadena de caracteres o de los números.

Como ejemplo, este es el arreglo que se va a ordenar:

```
1  arreglo = ["Z", "A", "Pedro", "Edwin", "Apple", "hqdwa0","0"]
```

Y este es el resultado:

```
['0']
['A']
['A', 'p', 'p', 'l', 'e']
['E', 'd', 'w', 'i', 'n']
['P', 'e', 'd', 'r', 'o']
['Z']
['h', 'q', 'd', 'w', 'a', '0']
```

Conclusiones

Los algoritmos de ordenamiento por conteo aceptan números positivos y ordena esa lista desordenada mucho más rápido que los algoritmos previamente vistos en clase. Counting Sort es bueno para el mejor de los casos y el caso promedio, Radix Sort mejor para el peor de los casos ya que es una optimización del algoritmo de Counting Sort.

Ambos algoritmos crean 2 arreglos más para guardar el conteo de números (arreglo C) y el

arreglo ordenado (arreglo B).

Los algoritmos fueron capaces de ordenar un arreglo de 20,000 números en menos de 1 segundo.

Por último, la modificación del algoritmo de Radix Sort para que ordene palabras, me tomó tiempo para diseñar el algoritmo e implementar la lógica adecuada, pero se cumplió con el objetivo, ordena los caracteres o palabras de un arreglo conforme a su número ASCII.