	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M.I. Edgar Tista García

Asignatura: Estructura de datos y algoritmos II -1317

Grupo: 10

No. de práctica(s): 3

Integrante(s): Mendoza Hernández Carlos Emiliano

No. de lista o brigada: 26

Semestre: 2023-1

Fecha de entrega: 20 de septiembre del 2022

Observaciones:

CALIFICACIÓN: _____

Práctica 3.

Algoritmos de ordenamiento. Parte3.

OBJETIVOS

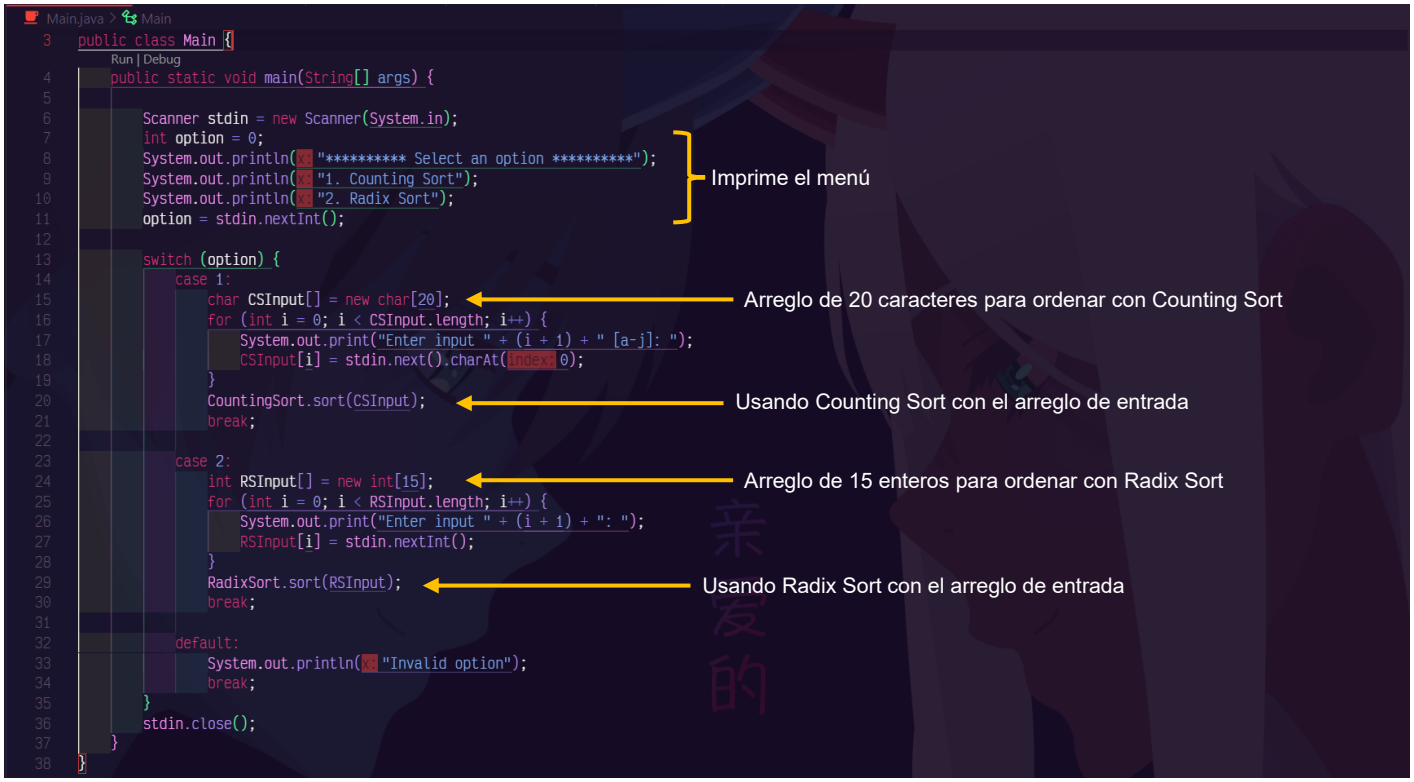
- **Objetivo general:** El estudiante identificará la estructura de los algoritmos de ordenamiento Counting Sort y Radix Sort.
- **Objetivo de la clase:** El alumno implementará casos particulares de estos algoritmos para entender mejor su funcionamiento a nivel algorítmico.

DESARROLLO

Ejercicio 0. Menú de usuario

Crea un menú para los algoritmos de ordenamiento de la práctica del día.

Implementación del menú:



Nota: Se asumirá que el usuario introducirá los valores correctamente.

Ejercicio 1. Counting Sort

En el lenguaje de tu preferencia (C o Java), realiza la implementación de Counting Sort aplicado a un caso específico. Toma en cuenta los siguientes requerimientos:

1) Utiliza un arreglo de 20 elementos (caracteres), los cuales serán solicitados al usuario (asume que el usuario los va a ingresar correctamente). Para ello considera solo letras minúsculas entre [a-j].
2) Crea un segundo arreglo donde cada posición será asociada a uno de los posibles valores del rango indicado (arreglo para hacer la cuenta).
3) En una primera pasada al arreglo que llenó el usuario, realiza la cuenta de las apariciones de cada uno de los valores. (Muestra en pantalla el arreglo de la cuenta al finalizar la primera pasada).

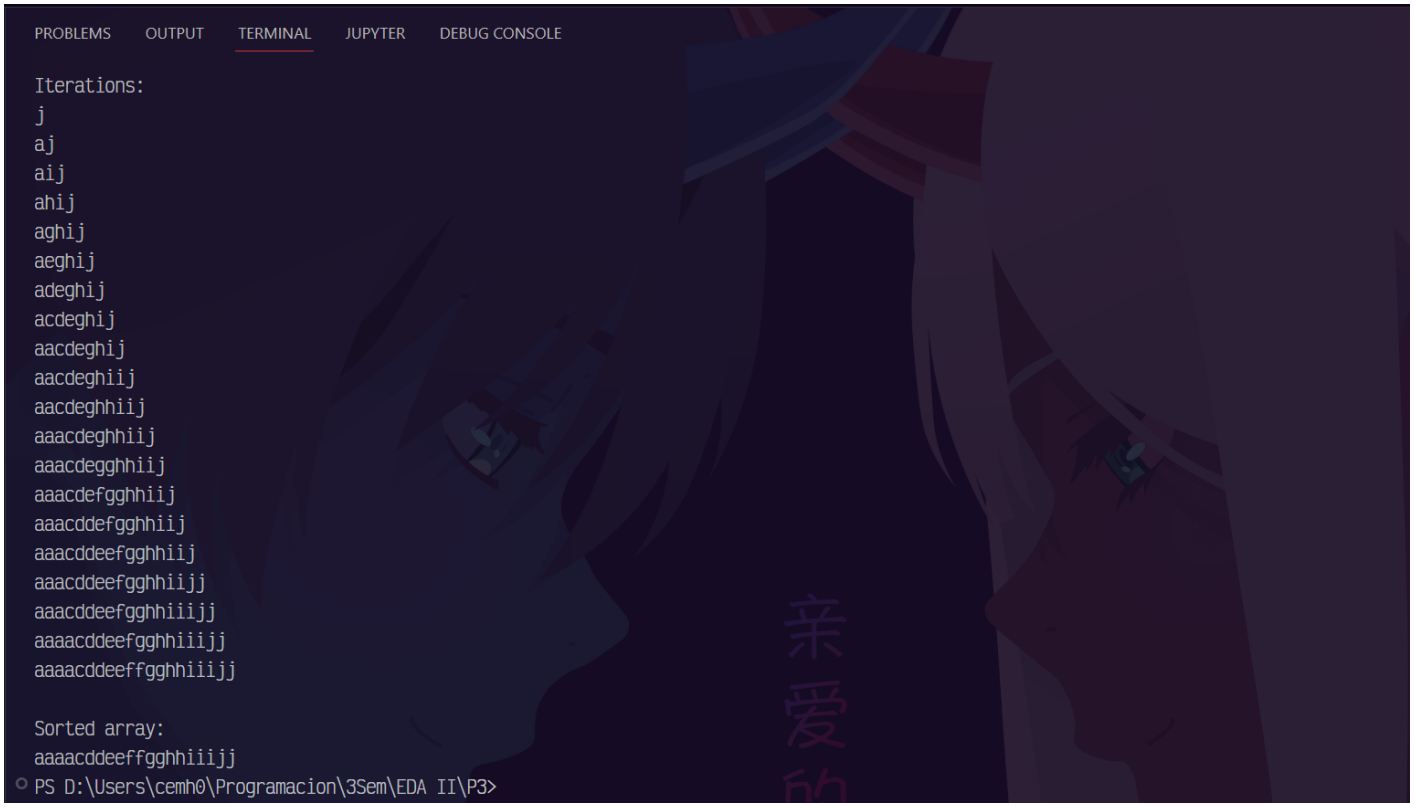
4) Realiza la cuenta del arreglo; en cada índice se considera la cantidad de los elementos actuales y los anteriores. (Muestra en pantalla la suma del arreglo).
5) Ingresa en un tercer arreglo los elementos ordenados de acuerdo con el funcionamiento del algoritmo, realizando una segunda pasada al primer arreglo, partiendo desde el final y para cada elemento, verifica la posición que le corresponde en el segundo arreglo y establece su posición final en el tercero. Para verificar el funcionamiento y los pasos del algoritmo, agrega impresiones de pantalla del arreglo de la cuenta, y del arreglo final conforme se van añadiendo los elementos.

Implementación de Counting Sort:

Ejecución del programa:

```
PROBLEMS  OUTPUT  TERMINAL  JUPYTER  DEBUG CONSOLE

PS D:\Users\cemh0\Programacion\3Sem\EDA II\P3> javac *.java
PS D:\Users\cemh0\Programacion\3Sem\EDA II\P3> java Main
********** Select an option **********
1. Counting Sort
2. Radix Sort
1
Enter input 1 [a-j]: f
Enter input 2 [a-j]: a
Enter input 3 [a-j]: i
Enter input 4 [a-j]: j
Enter input 5 [a-j]: e
Enter input 6 [a-j]: d
Enter input 7 [a-j]: f
Enter input 8 [a-j]: g
Enter input 9 [a-j]: a
Enter input 10 [a-j]: h
Enter input 11 [a-j]: i
Enter input 12 [a-j]: a
Enter input 13 [a-j]: c
Enter input 14 [a-j]: d
Enter input 15 [a-j]: e
Enter input 16 [a-j]: g
Enter input 17 [a-j]: h
Enter input 18 [a-j]: i
Enter input 19 [a-j]: a
Enter input 20 [a-j]: j
```

```
PROBLEMS  OUTPUT  TERMINAL  JUPYTER  DEBUG CONSOLE

Input array:
faijedfgahiacdeghiaj

Range: [a-j]

Count:
4 0 1 2 2 2 2 2 3 2
Sum:
4 4 5 7 9 11 13 15 18 20
```



Ejercicio 2. Radix Sort

En el lenguaje de tu preferencia (C o Java), realiza la implementación de Radix Sort basada en un caso particular similar al visto en clase, con las siguientes restricciones:

1) Los datos de entrada serán números de longitud máxima de 4, de entre los cuales solo aparecerán dígitos del 1 al 4.
2) Deberás implementar una cola para cada uno de estos dígitos (1,2,3,4).
3) Se deberán solicitar al usuario los valores a ordenar (mínimo 15), los cuales podrás almacenar en un arreglo o una lista (ejemplo de entrada: 1221, 4313, 2342, 1331, etc.)
4) Es necesario realizar varias pasadas sobre esta lista o arreglo, y en cada una de ellas utilizar las colas para almacenar los elementos de acuerdo con su posición (unidades, decenas, etc.). El programa deberá mostrar la lista resultante en cada iteración.
5) Al finalizar, el programa deberá mostrar la lista ordenada.

Implementación de Radix Sort:

Ejecución del programa:

```
PROBLEMS  OUTPUT  TERMINAL  JUPYTER  DEBUG CONSOLE

Q1: [1221, 1331, 4321, 1111, 4331]
Q2: [2342, 2222, 4132]
Q3: [4313, 3333]
Q4: [1234, 4444, 3114, 2214, 2314]
Array: 1221 1331 4321 1111 4331 2342 2222 4132 4313 3333 1234 4444 3114 2214 2314
End of iteration 1

Q1: [1111, 4313, 3114, 2214, 2314]
Q2: [1221, 4321, 2222]
Q3: [1331, 4331, 4132, 3333, 1234]
Q4: [2342, 4444]
Array: 1111 4313 3114 2214 2314 1221 4321 2222 1331 4331 4132 3333 1234 2342 4444
End of iteration 2

Q1: [1111, 3114, 4132]
Q2: [2214, 1221, 2222, 1234]
Q3: [4313, 2314, 4321, 1331, 4331, 3333, 2342]
Q4: [4444]
Array: 1111 3114 4132 2214 1221 2222 1234 4313 2314 4321 1331 4331 3333 2342 4444
End of iteration 3

Q1: [1111, 1221, 1234, 1331]
Q2: [2214, 2222, 2314, 2342]
Q3: [3114, 3333]
Q4: [4132, 4313, 4321, 4331, 4444]
Array: 1111 1221 1234 1331 2214 2222 2314 2342 3114 3333 4132 4313 4321 4331 4444
End of iteration 4
```

```
PROBLEMS  OUTPUT  TERMINAL  JUPYTER  DEBUG CONSOLE

Sorted array:
1111 1221 1234 1331 2214 2222 2314 2342 3114 3333 4132 4313 4321 4331 4444
PS D:\Users\cemh0\Programacion\3Sem\EDA II\P3>
```

CONCLUSIONES

Algunos puntos que se concluyen de la primera actividad son los siguientes:

- Counting Sort es eficiente (O(n)) en casos muy particulares (cuando hay muchos elementos repetidos en un rango reducido [0,k]).
- Para este caso particular de Counting Sort, se conocía el rango y el conteo no supuso un gasto de memoria muy grande. Sin embargo, mientras más grande sea el rango, mayor es el compromiso en el rendimiento debido a la memoria, dado que se usan dos colecciones del mismo tamaño que la original.

Para el segundo ejercicio se puede concluir que:

- Radix Sort es eficiente (O(nk)) en casos muy particulares (cuando las entradas no tienen muchos dígitos y se encuentran en un rango reducido de dígitos).
- Para este caso particular de Radix Sort, se conocían los posibles valores de los dígitos de los elementos, por lo que solo se necesitó de 4 colas para ordenar los elementos. Sin embargo, en el supuesto de tener todos los dígitos posibles se necesitarían 10 colas, y cada posición de los dígitos implica una iteración).

Dicho todo lo anterior, se ha cumplido con cada uno de los objetivos planteados para el desarrollo de esta práctica dado que se identificaron las diferencias entre ambos algoritmos, así como se comprendió su funcionamiento al codificar y realizar los ejercicios.

Como comentario final, considero que para esta práctica fue de más facilidad utilizar Java gracias a sus clases implementadas. Además, permitió resolver la práctica con la metodología de la programación orientada a objetos.