	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Edgar Tista García

Asignatura: Estructura de Datos y Algoritmos II

Grupo: 3

No de Práctica(s): 3

Integrante(s): Román Ramos María Fernanda

No. de lista o brigada: 31

Semestre: 2026-1

Fecha de entrega: 13/09/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Objetivos

- **Objetivo general:** El estudiante identificará la estructura de los algoritmos de ordenamiento MergeSort, 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.

**Nota: Para el desarrollo de la práctica usaré las herramientas:*

- *Visual Studio Code: para crear e implementar los programas en C y Java.*
- *Overleaf: para crear este documento en LaTeX del reporte.*

2. Desarrollo

2.1. Ejercicio 1. MergeSort

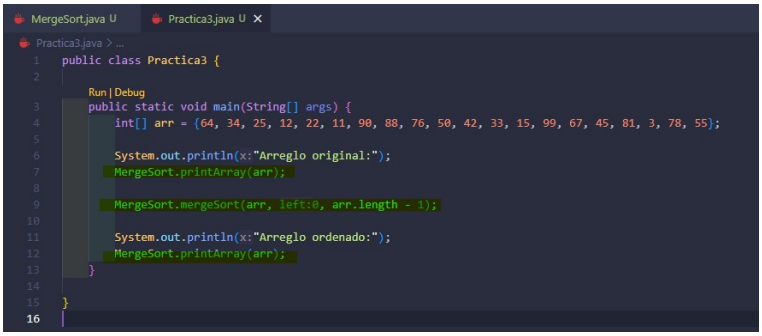
1. **Qué diferencias hay entre el algoritmo de merge sort proporcionado y el pseudo código visto en clase**

```java
static void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[] = new int[n1];
    int R[] = new int[n2];
    for (int i = 0; i < n1; ++i)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; ++j)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0;
    int k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
```

```java
void mergeSort(int list[], int first, int last) {
    int mid;
    if (first < last) {
        mid = (first + last) / 2;
        mergeSort(list, first, mid);
        mergeSort(list, mid + 1, last);
        merge(list, first, mid, last);
    }
}

void merge(int list[], int first, int mid, int last) {
    int list2[last - first + 1];
    int i, j, k;
    k = 0;
    i = first;
    j = mid + 1;

    while (i <= mid && j <= last) {
        if (list[i] < list[j])
            list2[k++] = list[i++];
        else
            list2[k++] = list[j++];
    }
    while (i <= mid)
        list2[k++] = list[i++];

    while (j <= last)
        list2[k++] = list[j++];

    for (i = last; i >= first; i--)
        list[i] = list2[--k];
}
```

- **Arreglos auxiliares**:
  - MergeSort.java: Usa dos arreglos (L[] y R[]) para las mitades izquierda (L) y derecha (R)
  - MergeSort pseudocódigo: Usa un solo arreglo temporal (list2[])
- **Copia del arreglo**:
  - MergeSort.java: Copia de izquierda a derecha el arreglo original (de i=0 hasta el tamaño-1)
  - MergeSort pseudocódigo: Copia de derecha a izquierda en el arreglo original (de i=last hasta 0)
- **For donde se copia el arreglo**:
  - MergeSort.java: Copia al inicio el arreglo original en los sub arreglos, y en los while va llenando el arreglo original ordenado con la estrategia de mergesort (ver los primeros indices de cada subarreglo y poner primero el que sea menor)
  - MergeSort pseudocódigo: trabaja en list2 con la estrategia de mergesort (ver los primeros de list2 y list y el que sea menor lo pone en el inicio), luego al final, copia de derecha a izquierda en el arreglo original (de i=last hasta 0)

2. **En segundo archivo (Practica3.java) escribe la función principal del programa para crear un arreglo de 20 elementos y ordenarlo con mergesort (agrega aquí una captura del código y de la ejecución)**



Conservé la clase MergeSort para sólo llamar sus métodos (PrintArray y mergeSOrt) en Practica3.java y pasarle los argumentos que necesita (arr y su tamaño con lenght)

3. **¿Para qué sirven n1 y n2?**
n1: Tamaño del subarray izquierdo
n2: Tamaño del subarray derecho
También se usan en el control de los for para copiar el contenido

4. **Para qué sirven los arreglos L y R dentro de Merge Sort (agrega en el código instrucciones para imprimir los arreglos L y R en cada iteración para ayudar a responder esta pregunta)**

```
public class MergeSort {

    static void merge(int arr[], int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;
        int L[] = new int[n1];
        int R[] = new int[n2];
        for (int i = 0; i < n1; ++i)
            L[i] = arr[left + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[mid + 1 + j];

        //IMPRIMIR L Y R
        System.out.print(s:"L: ");
        printArray(L);
        System.out.print(s:"R: ");
        printArray(R);
```

Esta parte imprime todo L y R después de llenarlos con sus for

- L: es la copia de la mitad izquierda del arreglo original (desde left hasta mid)
- R: es la copia de la mitad derecha del arreglo original (desde mid+1 hasta right)

Función:

```java
while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
    k++;
}
```

```java
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
```

```java
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
```

While 1 / While 2 / While 3

While 1: compara elementos de L y R y coloca el menor en la posición actual de arr
While 2: si quedan elementos en L (porque R se acabó), los copia todos directamente
While 3: si quedan elementos en R (porque L se acabó), los copia todos directamente

5. **Consideras adecuado que los métodos de MergeSort sean estáticos, (si o no y por qué?)**
   Sí, ya que no se necesita crear un objeto para usarlo, por lo que no debe crear una nueva referencia en memoria con new, pueden solo llamarse directamente como en ***Principal.java*** *que se llama solo al método MergeSort.mergeSort(arr)*

6. **Dónde consideras que inicia y termina una "Iteración" de MergeSort. Agrega en tu programa impresiones de esas "iteraciones"**
   Considero que cada ïteraciónïnicia en cada llamada recursiva que se hace de mergeSort, ya que procesa un array relativamente diferente, que son partes cada vez más pequeñas del mismo, pero finalmente se procesan entradas diferentes, y termina la iteración cuando se hace merge después de que las llamadas recursivas terminan
   Funciones para imprimir:

```java
static int iterac = 0;

static void printSubArray(int arr[], int inicio, int fin) {
    for (int i = inicio; i < fin; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

static void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        System.out.println("ITERACIÓN " + (++iterac) );

        System.out.print(s:"   sublista actual: ");
        printSubArray(arr, left, right);
        System.out.println(x:"---");

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
        System.out.println("iteración " + iterac + " terminada");
        System.out.print(s:"   después del merge: ");
        printSubArray(arr, left, right);
        System.out.println(x:"---");
    }
}
```

- iterac: sirve para enumerar cada llamada a mergeSort, la hice static porque si no, indicaba un error, ero lo solucioné e esta manera ya que pertenece como tal a la clase, no a un objeto
- printSubarray: hice esta funcion a parte ed printArray para que reciba los índices de la parte actuala
- Se usa printSubarray antes de dividirlo y mientras se divide la parte actual

En las iteraciones:
IIteracion 1: Todo el arreglo
Iteraciones 2-10: mtad izquierda
Iteraciones 11-19: mitad derecha

7. **Agrega a la función principal las instrucciones para que el arreglo se llene con valores aleatorios de 1 a 100**

```java
import java.util.Random;
public class Practica3 {

    Run | Debug
    public static void main(String[] args) {


        int[] arr =  new int[20];
        Random rand = new Random();

        for (int i = 0; i < 20; i++) {
            arr[i] = rand.nextInt(bound:100) + 1;
        }

```

Con la librería Random, se generan numeros aleatorios, en ***rand.nextInt(100) + 1*** *se generan del 1 al 100 y se va llenando arr con ellos*

2.1.1. Resultados

```
roman@RomL MINGW64 ~/EDA 2/PRACTICA 3 - TODO/Practica (main)
$ javac *.java

roman@RomL MINGW64 ~/EDA 2/PRACTICA 3 - TODO/Practica (main)
$ java Practica3
Arreglo original:
75 15 12 73 15 99 15 18 68 61 84 8 3 99 59 85 1 55 33 11
Tamaño del arreglo: 20
---------------------------
ITERACIÓN 1
   sublista actual: 75 15 12 73 15 99 15 18 68 61 84 8 3 99 59 85 1 55 33
---
ITERACIÓN 2
   sublista actual: 75 15 12 73 15 99 15 18 68
```

— iteraciones —

```
---
L: 12 15 15 15 18 61 68 73 75 99
R: 1 3 8 11 33 55 59 84 85 99
iteración 19 terminada
   después del merge: 1 3 8 11 12 15 15 15 18 33 55 59 61 68 73 75 84 85 99
---
Arreglo ordenado:
1 3 8 11 12 15 15 15 18 33 55 59 61 68 73 75 84 85 99 99
```

2.2. Ejercicio 2. CountigSort

Requerimientos:

1. Utiliza un arreglo de 20 elementos, los cuales serán solicitados al usuario (asume que el usuario los va a ingresar correctamente) Para ello considera solo números enteros del 5 al 20

2. Crea un segundo arreglo donde cada posición será asociada a uno de los posibles valores del rango indicado (arreglo para hacer la cuenta)

```c
#include <stdio.h>
#define SIZE 20
#define VALMIN 5
#define VALMAX 20
#define RANGO (VALMAX - VALMIN + 1)

int main() {
    int arr[SIZE]; //original
    int cont[RANGO] = {0}; // contar las apariciones
    int sorted[SIZE]; //ordenado

    // solicitar los 20 elem
    printf("ingresa 20 numeros enteros entre %d y %d:\n", VALMIN, VALMAX);
    for(int i = 0; i < SIZE; i++) {
        printf("elemento %d: ", i+1);
        scanf("%d", &arr[i]);

        // que solo estén en el rango
        if(arr[i] < VALMIN || arr[i] > VALMAX) {
            printf("Error: El numero debe estar entre %d y %d\n", VALMIN, VALMAX);
            i--; // si pone uno que no está en el rango, lo pide otra vez
        }
    }
```

1) y 2)

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)

```c
// contar las apariciones de cada elem
for(int i = 0; i < SIZE; i++) {
    cont[arr[i] - VALMIN]++;
}
// imprimir el arreglo donde los cuenta
printf("\nArreglo de conteo (despues de primera pasada):\n");
for(int i = 0; i < RANGO; i++) {
    printf("valor %d: aparce %d veces \n", i + VALMIN, cont[i]);
}
```

4. Realiza la suma de los elementos del arreglo; en cada índice se considera la cantidad de elementos actuales y los anteriores. (Muestra en pantalla la suma del arreglo)

```c
// suma de actual y anteriores
for(int i = 1; i < RANGO; i++) {
    cont[i] += cont[i-1];
}
//mostrar la suma
printf("\nSuma:\n");
for(int i = 0; i < RANGO; i++) {
    printf("Hasta valor %d: %d elementos\n", i + VALMIN, cont[i]);
}
```

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.

```c
// llenar el arreglo ordenado
printf("\nAcmodo al arreglo final:\n");
for(int i = SIZE - 1; i >= 0; i--) {
    int valor = arr[i];
    int posic = cont[valor - VALMIN] - 1;

    sorted[posic] = valor;
    cont[valor - VALMIN]--;

    printf("elemnto %d (%d) colocado en posicicion %d", i, valor, posic);
    printf("\n");
}

printf("\nArreglo ordenado: ");
for(int i = 0; i < SIZE; i++) {
    printf("%d ", sorted[i]);
}
```

   posic es la posición correcta donde debe ponerse el elemento, se calcula del arreglo cont donde tiene las posiciones de los elemenros y resta 1

Para verificar el funcionamiento y los pasos del algoritmo, agrega impresiones en pantalla en el acomodo del arreglo final ordenado

2.2.1. Resultados

```
ingresa 20 numeros enteros entre 5 y 20:
elemento 1: 7
elemento 2: 8
elemento 3: 9
elemento 4: 5
elemento 5: 15
elemento 6: 6
elemento 7: 12
elemento 8: 17
elemento 9: 18
elemento 10: 15
elemento 11: 19
elemento 12: 19
elemento 13: 19
elemento 14: 20
elemento 15: 20
elemento 16: 17
elemento 17: 8
elemento 18: 5
elemento 19: 9
elemento 20: 3
Error: El numero debe estar entre 5 y 20
elemento 20: 12

Arreglo original: 7 8 9 5 15 6 12 17 18 15 19 19 19 20 20 17 8 5 9 12
```

Elementos ingresados

```
Arreglo de conteo (despues de primera pasada):
valor 5: aparce 2 veces
valor 6: aparce 1 veces
valor 7: aparce 1 veces
valor 8: aparce 2 veces
valor 9: aparce 2 veces
valor 10: aparce 0 veces
valor 11: aparce 0 veces
valor 12: aparce 2 veces
valor 13: aparce 0 veces
valor 14: aparce 0 veces
valor 15: aparce 2 veces
valor 16: aparce 0 veces
valor 17: aparce 2 veces
valor 18: aparce 1 veces
valor 19: aparce 3 veces
valor 20: aparce 2 veces
```

Conteo de cada elemento

```
Suma:
Hasta valor 5: 2 elementos
Hasta valor 6: 3 elementos
Hasta valor 7: 4 elementos
Hasta valor 8: 6 elementos
Hasta valor 9: 8 elementos
Hasta valor 10: 8 elementos
Hasta valor 11: 8 elementos
Hasta valor 12: 10 elementos
Hasta valor 13: 10 elementos
Hasta valor 14: 10 elementos
Hasta valor 15: 12 elementos
Hasta valor 16: 12 elementos
Hasta valor 17: 14 elementos
Hasta valor 18: 15 elementos
Hasta valor 19: 18 elementos
Hasta valor 20: 20 elementos
```

Suma de el actual con el anterior

```
Acmodo al arreglo final:
elemnto 19 (12) colocado en posicicion 9
elemnto 18 (9) colocado en posicicion 7
elemnto 17 (5) colocado en posicicion 1
elemnto 16 (8) colocado en posicicion 5
elemnto 15 (17) colocado en posicicion 13
elemnto 14 (20) colocado en posicicion 19
elemnto 13 (20) colocado en posicicion 18
elemnto 12 (19) colocado en posicicion 17
elemnto 11 (19) colocado en posicicion 16
elemnto 10 (19) colocado en posicicion 15
elemnto 9 (15) colocado en posicicion 11
elemnto 8 (18) colocado en posicicion 14
elemnto 7 (17) colocado en posicicion 12
elemnto 6 (12) colocado en posicicion 8
elemnto 5 (6) colocado en posicicion 2
elemnto 4 (15) colocado en posicicion 10
elemnto 3 (5) colocado en posicicion 0
elemnto 2 (9) colocado en posicicion 6
elemnto 1 (8) colocado en posicicion 4
elemnto 0 (7) colocado en posicicion 3

Arreglo ordenado: 5 5 6 7 8 8 9 9 12 12 15 15 17 17 18 19 19 19 20 20
PS C:\Users\roman\EDA 2\PRACTICA 3 - TODO\Practica\Ejericicio 2>
```

Acomodo de elemenots en sorted

2.3. Ejercicio 3. RadixSort

***Nota: A pesar de que en el Avance pude implementarlo en C, para este ejercico me parece más útil y/o más facil implementarlo en Java, ya que recientemente (después del dia del avance) he aprendido varias funciones en java más directas que en C, usando Colas o Queues de java*
Restricciones:

1. **Los datos de entrada serán números de longitud máxima de 4 , de entre los cuales solo aparecerán dígitos del 0 al 4**
   En el main, puse las restricciones para aceptar cada valor, deben cumplir con que sean de 4 dígitos (en el caso de los que tienen menos se deberían ingresar con ceros a la izquierda, como 120 sería 0120) y que los dígitos que ponga sean de 0 a 4

```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int[] arr = new int[15];
    System.out.println(x:"Ingresa 15 números de 4 dígitos (solo dígitos del 0 al 4):");
    //verif que cumplan y los agrega al arreglo
    for (int i = 0; i < 15; i++) {
        boolean valEnt = false;
        while (!valEnt) {
            System.out.print("Número " + (i + 1) + ": ");
            String ent = sc.next();

            if (ent.length() != 4) {
                System.out.println(x:"Debe ser de 4 dígitos, intenta otra vez");
                continue;
            }

            boolean digVal = true;
            for (char c : ent.toCharArray()) {
                if (c < '0' || c > '4') {
                    digVal = false;
                    break;
                }
            }

            if (!digVal) {
                System.out.println(x:"los dígitos deben estar entre 0 y 4, intenta otra vez");
                continue; //regresa para volver a pedirlo
            }

            arr[i] = Integer.parseInt(ent);
            valEnt = true; //salir del while
        }
    }
```

1) y 3)

Explicación:

- con el for, se itera 15 veces para recibir los 15 valores
- **boolean valEnt = false** : sirve para checar si es una entrada válida, inicia como falso porqueasume que no lo es, si pasa las restricciones se cambia para salir del while
- **while (!valEnt) {...** : dentro se hacen las verificaciones de que cumpla
- Después de recibir el número como string (esto lohice para que pueda recorrer sus posiciones), **ent.length** mide la longitud de la entrada del scanner, si no cumple, vuelve al inicio del while para volver a pedirlo (por eso puse que reciba el numero dentro del while)

- Si pasó la verificación anterior, ahora:
  boolean digVal checa si el dígito es válido, con **for (char c : ent.toCharArray()) { ...** recorre cada caracter de ent (toCharArray lo convierte a un arreglo de caracteres), verifica que está en el rango, si no lo está cambia *digVal* y hace el **break** para terminar el for con el primero que encuentre que no está en el rango.
  En **if (!digVal) {** si encontró que hay valores fuera del rango, manda el mensaje y con **continue** regresa para volver a pedirlo
- Si pasa todas las validaciones, asigna el valor en la posición i del arreglo, primero lo castea a un entero con **parseInt** y para salir del while cambia valEnt a true y puede meterse a la siguiente iteración del for

2. **Deberás implementar una cola para cada uno de estos dígitos (0,1,2,3,4)**
   Usé las Queues que incluye Java al impotrtar *java.util*, que son una interfaz de Java, que se comporta como una cola con sus funciones de encolar (add) y desencolar (poll) [1]

```java
// crea y retorna 5 colas de enteros, para los dígitos del 0 al 4
private static List<Queue<Integer>> crearQueues() {
    List<Queue<Integer>> queues = new ArrayList<>(); //aquí almacenará las colas
    for (int i = 0; i < 5; i++) {
        queues.add(new LinkedList<>());
    }
    return queues;
}
```

   Cree una funcion crearQueues para tener mejor control de cuántas crea, si se admitieran más elementos, solo se tendría que cambiar del valor dentro del for de la función

   a) En la primera parte, creo un ArrayList donde almacenará las colas creadas
   b) En **queues.add(new LinkedList<>())** añade 5 colas como LinkedList a la lista
   c) Por último, retorna las queues creadas

3. **Se deberán solicitar al usuario los valores a ordenar (15 elementos), los cuales podrás almacenar en un arreglo o una lista (ejemplo de entrada: 4203, 0313, 2302, 1031, 2021, 1143, etc.)**

   (Imagen en el punto 1)

4. **Es necesario realizar 4 iteraciones 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**

```java
// 4 iteraciones
public static void radixSort(int[] arr) {
    List<Queue<Integer>> queues = crearQueues();
    // 4 iteraciones
    for (int digitPos = 0; digitPos < 4; digitPos++) {
        // poner cada número en la cola correspondiente a su dígito
        for (int num : arr) {
            int digit = getDigit(num, digitPos);
            queues.get(digit).add(num);
        }
        // desencolar números de las colas en orden
        int index = 0;
        for (Queue<Integer> queue : queues) {
            while (!queue.isEmpty()) { //sale cuando la cola está vacía
                arr[index++] = queue.poll(); //desencolar
            }
        }
        // Mostrar la lista después de esta iteración
        System.out.print("después de la iteración " + (digitPos + 1) + " (dígito " + digitPos + "): ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

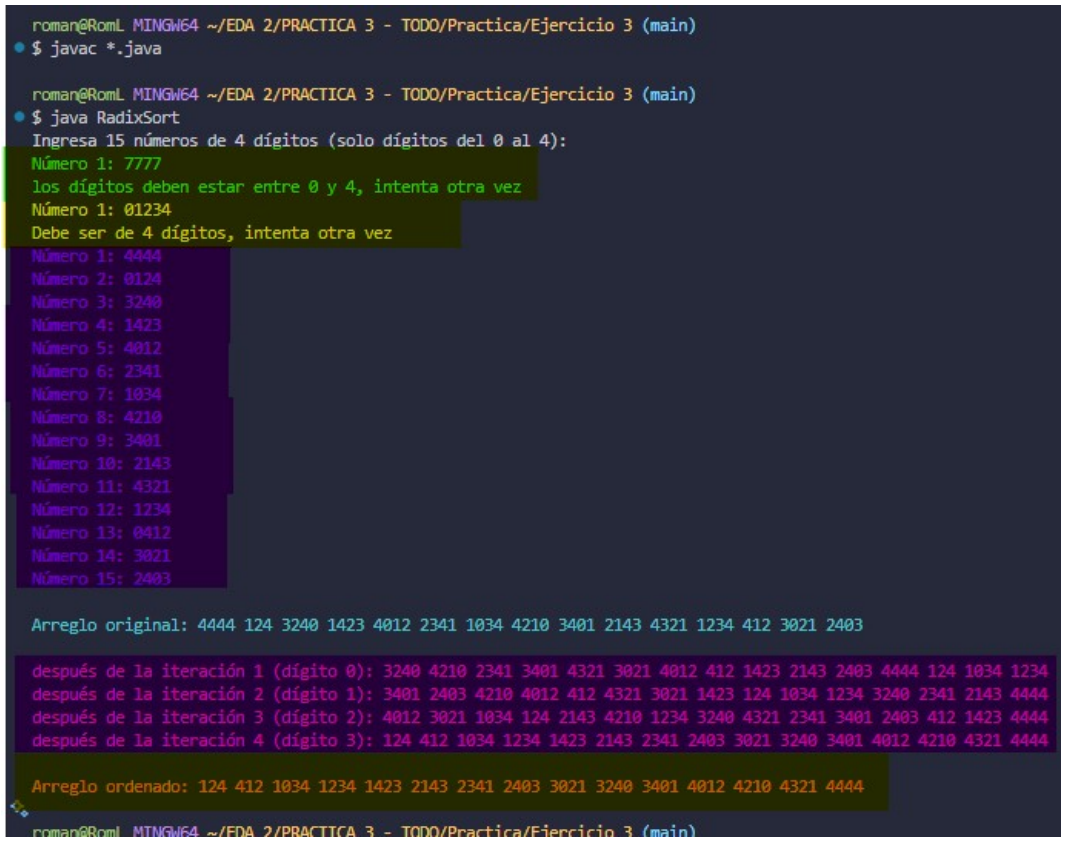
- Primer for (para encolar):
  - **digitPos** es una variable que controla qué dígito estamos tratando en cada iteración (unidades, decenas, centenas, etc)
  - **getDigit(num, digitPos)**: obtiene el dígito en la posición actual
  - **queues.get(digit)**: selecciona a la cola correspondiente al dígito que se encontró
  - **.add(num)**: encola el número en esa cola
- Segundo for (para desencolar):
  - **for (Queue<Integer>queue : queues)**: recorre cada cola en orden (0,1,2,3,4)
  - **while (!queue.isEmpty())**: mientras la cola **no** esté vacía
  - **queue.poll()**: desencola el primer número de la cola
  - **arr[index++]**: Coloca el número en el arreglo en orden
- Al último con digitPos muestra el arreglo hasta el momento

5. Al finalizar, el programa deberá mostrar la lista ordenada

```java
System.out.print(s:"\nArreglo original: ");
for (int num : arr) {
    System.out.print(num + " ");
}
System.out.println(x:"\n");

radixSort(arr);

System.out.print(s:"\nArreglo ordenado: ");
for (int num : arr) {
    System.out.print(num + " ");
}
System.out.println();

sc.close();
```

Impresiones del arreglo antes y después de ordenar

2.3.1. Resultados



Resultados:

1. *Prueba de digitos no válidos*
2. *Prueba de número muy largo*
3. *Llenar el arreglo*
4. *arreglo original*
5. *Mostrar iteraciones de desencolar*
6. *Arreglo ordenado*

3. Conclusiones

En general, se cumplieron los objetivos de la práctica, ya que me fué muy útil para realmente pensar en cómo implementar los algoritmos vistos en clase (Merge, Counting y Radix), a pesar de comprendido los conceptos y las estrategias que usan, a la hora de implementarlos en un lenguaje de programación, logré comprender, por ejemplo, las desventajas de Counting y Radix Sort, que a pesar de que en mi opinión siguen lógicas o pasos bastante sencillos de entender, al momento de aplicarlos en el programa, honestamente me costó mucho trabajo justamente por los componentes extra de memoria que necesitan (los arreglos auxiliares y queues), a parte de que comprendí que no hay una ÚNICA forma de hacer las implementaciones, ya que en el avance de la práctica usé arreglos en lugar de colas para Radix Sort, y en este reporte elegí usar java y la interfaz de queue que tiene, y esto también me ayudó a comprender más conceptos del lenguaje Java.
** También pude usar conceptos que vi en la materia de Programación Orientada a Objetos, como las linkedLists, y mi profesor de esta materia fue quien mencionó que existía la interfaz de queue en Java, así que quise investigarla para esta práctica.

4. Referencias

[1] Geeksforgeeks. Queue interface in java. (https://www.geeksforgeeks.org/java/queue-interface-java/), 21 Agosto, 2025. Nota: Fuente para usar las Queues de Java.