Universidad de San Carlos de Guatemala

Centro universitario de occidente

División de ciencias de la ingeniería

PRACTICA 1

Anthony Samuel Ordoñez Son 201931185

Complejidad de algoritmos:

Agregar apuesta:

```
public void generarApuestas(int numeroLineas, String texto) {
    long inicio = System.nanoTime();
    String[] lineas = arrayUtil.split(texto, '\n', numeroLineas);
    this.totales = new Apuesta[numeroLineas];
    int pasos = 0;
    for (int i = 0; i < lineas.length; i++) {
        if (lineas[i] == null) {
            break;
        }
        String[] singleLine = arrayUtil.split(lineas[i], ',', 12);
        Apuesta apuesta = new Apuesta(singleLine[0], singleLine[1], singleLine[2],
                singleLine[3], singleLine[4], singleLine[5],
                singleLine[6], singleLine[7], singleLine[8],
                singleLine[9], singleLine[10], singleLine[11]);
        this.totales[i] = apuesta;
        System.out.println("GENRACION: " + apuesta.toString());
        pasos++;
    }
    if (pasos > 0) {
        this.pasosIngreso.add(pasos);
    }
    long finale = System.nanoTime();
    this.tiemposIngreso.add((finale - inicio));

}
```

Se usa un ciclo para leer los datos encontrados en el documento, sin embargo el método de agregación de estos resultados es de un solo paso, ese paso se da en la construcción del objeto, por lo tanto podemos decir que la complejidad es de O(1).

Verificar apuesta:

Para la verificación de una apuesta, se usa el método “Verificar Arreglo”

```
public boolean verificarArreglo(int[] numeros) {
    int pasos = 0;
    for (int i = 0; i < 10; i++) {
        this.pasos++;
        int actual = numeros[i] - 1;
        if (actual < 0 || actual >= 10) {
            checkers = new boolean[10];
            return false;
        }
        if (checkers[actual]) {
            System.out.println("El numero: " + numeros[i] + " esta repetido," + "En posicion " + i);
            checkers = new boolean[10];
            return false;
        }
        checkers[numeros[i] - 1] = true;
    }
    System.out.println("Ningun numero se repite");
    checkers = new boolean[10];
    return true;
}
```

se tiene un solo ciclo y se hace usó de un arreglo de booleans, cuya posición representa un número, al encontrarse un número, por ejemplo el 3, el boolean en la posición correspondiente a ese número cambia su estado, si este número se repite, el boolean ya habrá cambiado, por lo tanto se asume que el número está repetido y se procede a rechazar la apuesta, al ser un solo ciclo, con solo un par de comprobaciones adicionales, podemos reducir la complejidad del algoritmo a O(n) para el arreglo.

Cálculo de resultados:

```
public void calcularResultados() {
    limpiarResultados();
    int pasos = 0;
    long inicio = System.nanoTime();
    for (int i = 0; i < resultados.length; i++) {
        for (Apuesta temporal : totales) {
            pasos++;
            if (temporal == null) {
                continue;
            }

            if (temporal.isRechazo() == false) {
                if (temporal.getPosiciones()[i] == resultados[i]) {
                    temporal.addToPunteo(10 - i);
                }
                System.out.println(temporal.toString());
            }
        }
    }
    if (pasos > 0) {
        this.pasosProceso.add(pasos);
    }
    long finale = System.nanoTime();
    this.tiemposProceso.add((finale - inicio));
}
```

Para la funcionalidad de calcular los resultados se usan dos ciclos, uno para recorrer la lista de resultados y otro para recorrer la lista de apuestas, en cada ciclo se verifica una posición de cada apuesta y se compara con la misma posición en los resultados, si las posiciones coinciden, se suma el punteo acordado a la apuesta, al tener dos ciclos, uno dentro de otro, pero al ser el primer ciclo de tamaño fijo 10, llegamos a la complejidad de O(10N)

OTROS ALGORITMOS:

Generación de apuestas:

```
public void generarApuestas(int numeroLineas, String texto) {
    long inicio = System.nanoTime();
    String[] lineas = arrayUtil.split(texto, '\n', numeroLineas);
    this.totales = new Apuesta[numeroLineas];
    int pasos = 0;
    for (int i = 0; i < lineas.length; i++) {
        if (lineas[i] == null) {
            break;
        }
        String[] singleLine = arrayUtil.split(lineas[i], ',', 12);
        Apuesta apuesta = new Apuesta(singleLine[0], singleLine[1], singleLine[2],
                singleLine[3], singleLine[4], singleLine[5],
                singleLine[6], singleLine[7], singleLine[8],
                singleLine[9], singleLine[10], singleLine[11]);
        this.totales[i] = apuesta;
        System.out.println("GENRACION: " + apuesta.toString());
        pasos++;
    }
    if (pasos > 0) {
        this.pasosIngreso.add(pasos);
    }
    long finale = System.nanoTime();
    this.tiemposIngreso.add((finale - inicio));

}
```

Se lee la información almacenada en arreglos y se convierte en 1 solo paso a apuesta con datos, la complejidad de este algoritmo es de o(n).

Split

Este método es utilizado para extraer información útil de una cadena en forma de array, se usa un StringBuilder para recolectar esta información, separada por un símbolo, se llena un array de tamaño N con un ciclo, por lo que su complejidad es de O(n).

```
public String[] split(String texto, char caracter, int tamaño) {
    String[] datos = new String[tamaño];
    StringBuilder palabra = new StringBuilder("");
    int posicion = 0;
    for (int i = 0; i < texto.length(); i++) {
        char temp = texto.charAt(i);
        if (true) {

        }
        if (posicion >= tamaño) {

            return datos;
        }
        if (temp == caracter) {
            datos[posicion] = palabra.toString();
            palabra.setLength(0);
            posicion++;
            continue;
        }
        palabra.append(texto.charAt(i));
        if (i == texto.length() - 1) {
            datos[posicion] = palabra.toString();
            palabra.setLength(0);
        }
    }
    return datos;
}
```