USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala
INGENIERIA
CUNOC

Laboratorio de Estructura de Datos

Práctica 1: Documentación técnica

Sección: A

Nombre: Registro académico:

Mariano Francisco Camposeco Camposeco 202030987

Quetzaltenango, 05 de marzo de 2022.

ÍNDICE

Complejidad de servicios críticos (con código): 1
→**Ingreso de las apuestas antes del inicio de la carrera O(1)** 1
→**Verificación de apuestas O(n)** 1
→**Cálculo de los resultados al finalizar la carrera O(n)** 3
→**Ordenamiento de resultados O(n^2)** 5
Método burbuja en orden por punteo: 5
Método burbuja en orden alfabético: 5
Complejidad métodos implementados (con código): 6
→**Verificar repitencia en datos ingresados como ganadores** 6
→**Guardar Resultados:** 7
→**Cerrar apuestas y verificar error de léxico** 8
Complejidad de servicios críticos (reducido): 10
→**Ingreso de las apuestas antes del inicio de la carrera O(1)** 10
→**Verificación de apuestas O(n)** 10
→**Cálculo de los resultados al finalizar la carrera O(n)** 10
→**Ordenamiento de resultados O(n^2)** 10
Método burbuja en orden por punteo: 10
Método burbuja en orden alfabético: 11
Complejidad métodos implementados (reducido) : 11
→**Verificar repitencia en datos ingresados como ganadores** 11
→**Guardar Resultados:** 11
→**Cerrar apuestas y verificar error de léxico** 11
Método de ordenamiento argumentación: 12

Complejidad de servicios críticos (con código):

→Ingreso de las apuestas antes del inicio de la carrera O(1)

```
FileReader leerA = new FileReader(archivo);//ingresamos archivo//3 pasos
BufferedReader leerTA = new BufferedReader(leerA);//lector del archivo
String linea;
    while ((linea = leerTA.readLine()) != null) {//1 paso
            texto.append(linea+"\n");
    }
```

Cantidad de pasos= 4=1 = **O(1) complejidad R//**

→Verificación de apuestas O(n)

```
for (int i = 0; i < apostadoresTemporal.length; i++) {//n paso
        boolean verificador = true;//2 pasos
        if (apostadoresTemporal[i] != null) {//10 pasos
           contadorTemporal=contadorTemporal+10;
           espacio[0] = String.valueOf(apostadoresTemporal[i].getPrimero());
           espacio[1] = String.valueOf(apostadoresTemporal[i].getSegundo());
           espacio[2] = String.valueOf(apostadoresTemporal[i].getTercero());
           espacio[3] = String.valueOf(apostadoresTemporal[i].getCuarto());
           espacio[4] = String.valueOf(apostadoresTemporal[i].getQuinto());
           espacio[5] = String.valueOf(apostadoresTemporal[i].getSexto());
           espacio[6] = String.valueOf(apostadoresTemporal[i].getSeptimo());
           espacio[7] = String.valueOf(apostadoresTemporal[i].getOctavo());
           espacio[8] = String.valueOf(apostadoresTemporal[i].getNoveno());
           espacio[9] = String.valueOf(apostadoresTemporal[i].getDecimo());
           for (int j = 0; j < 9; j++) {//9,8,7,6,5,4,3,2,2 pasos
              if (espacio[j].equals(espacio[j + 1])) {
                 verificador = false;
                 break;
              }
              if (j < 8) {
                 if (espacio[j].equals(espacio[j + 2])) {
```

```
            verificador = false;
            break;
        }
    }
    if (j < 7) {
        if (espacio[j].equals(espacio[j + 3])) {
            verificador = false;
            break;
        }
    }
    if (j < 6) {
        if (espacio[j].equals(espacio[j + 4])) {
            verificador = false;
            break;
        }
    }
    if (j < 5) {
        if (espacio[j].equals(espacio[j + 5])) {
            verificador = false;
            break;
        }
    }
    if (j < 4) {
        if (espacio[j].equals(espacio[j + 6])) {
            verificador = false;
            break;
        }
    }
    if (j < 3) {
        if (espacio[j].equals(espacio[j + 7])) {
            verificador = false;
            break;
```

```
                    }
                }
                if (j < 2) {
                    if (espacio[j].equals(espacio[j + 8])) {
                        verificador = false;
                        break;
                    }
                }
                if (j < 1) {
                    if (espacio[j].equals(espacio[j + 9])) {
                        verificador = false;
                        break;
                    }
                }
            }
            if (verificador) {//3 pasos
                apostadores[contador] = apostadoresTemporal[i];
                contador++;
            }
        }
        }
    }
```

Cantidad de pasos= n(1+2+10+9+8+7+6+5+4+3+2+2+3)=62n = **O(n) complejidad R//**

→Cálculo de los resultados al finalizar la carrera O(n)

```
for (int i = 0; i < apostadores.length; i++) {//n pasos
            if (apostadores[i] != null) {//1 paso
                if (apostadores[i].getPrimero() == Integer.parseInt(orden[0])) {//2 pasos
                    apostadores[i].setPunteo(apostadores[i].getPunteo() + 10);
                }
                if (apostadores[i].getSegundo() == Integer.parseInt(orden[1])) {//20 pasos
                    apostadores[i].setPunteo(apostadores[i].getPunteo() + 9);
```

```
            }
            if (apostadores[i].getTercero() == Integer.parseInt(orden[2])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 8);
            }
            if (apostadores[i].getCuarto() == Integer.parseInt(orden[3])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 7);
            }
            if (apostadores[i].getQuinto() == Integer.parseInt(orden[4])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 6);
            }
            if (apostadores[i].getSexto() == Integer.parseInt(orden[5])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 5);
            }
            if (apostadores[i].getSeptimo() == Integer.parseInt(orden[6])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 4);
            }
            if (apostadores[i].getOctavo() == Integer.parseInt(orden[7])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 3);
            }
            if (apostadores[i].getNoveno() == Integer.parseInt(orden[8])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 2);
            }
            if (apostadores[i].getDecimo() == Integer.parseInt(orden[9])) {
                apostadores[i].setPunteo(apostadores[i].getPunteo() + 1);
            }
        }
    }
    {
    }
```

Cantidad de pasos= n(1+2+20)=23n = **O(n) complejidad R//**

→Ordenamiento de resultados O(n^2)

Método burbuja en orden por punteo:

```
Apostador temp[]=new Apostador[1]; //1 paso
boolean finalizar=true;//1 paso
boolean ordenar; //1 paso
for (ordenar = false;  !ordenar;) {//n pasos
        for (int i = 0; i <apostadores.length-1; i++) {//n pasos
           if(apostadores[i]!=null && apostadores[i+1]!=null){//1 paso
              if(apostadores[i].getPunteo()<apostadores[i+1].getPunteo()){//1 paso
              temp[0] = apostadores[i+1];//intercambiamos posiciones// 1paso
              apostadores[i+1]=apostadores[i];//1 paso
              apostadores[i]=temp[0];//1 paso
              finalizar=false;//1paso
              }
           }
        }if(finalizar){//1paso
           ordenar=true;//1paso
        }finalizar=true;//1paso
}
```

Cantidad de pasos= 3+n(n(1+5+2+1)= 3+9n^2 = **O(n^2) complejidad R//**

Método burbuja en orden alfabético:

```
Apostador temp[]=new Apostador[1]; //1 paso
boolean finalizar=true;//1 paso
boolean ordenar; //1 paso
for (ordenar = false;  !ordenar;) {//n
        for (int i = 0; i <apostadores.length-1; i++) {//n
           if(apostadores[i]!=null && apostadores[i+1]!=null){//1 paso
          if(apostadores[i].getNombre().compareToIgnoreCase(apostadores[i+1].getNombre())>0){//1paso
              temp[0] = apostadores[i+1];//intercambiamos posiciones//1paso
              apostadores[i+1]=apostadores[i];//1paso
              apostadores[i]=temp[0];//1paso
```

```
            finalizar=false;//1paso
          }
          }
        }if(finalizar){//1paso
          ordenar=true;//1paso
        }finalizar=true;//1paso
 }
```

Cantidad de pasos= 3+n(n(1+5+2+1)= 3+9n^2 = **O(n^2) complejidad R//**

Complejidad métodos implementados (con código):

→Verificar repitencia en datos ingresados como ganadores

```
for (int i = 0; i < 10; i++) {//10(10)+10=110 pasos
        if (c[i].equals("1")) {
            orden[0] = "" + (i + 1);
        }
        if (c[i].equals("2")) {
            orden[1] = "" + (i + 1);
        }
        if (c[i].equals("3")) {
            orden[2] = "" + (i + 1);
        }
        if (c[i].equals("4")) {
            orden[3] = "" + (i + 1);
        }
        if (c[i].equals("5")) {
            orden[4] = "" + (i + 1);
        }
        if (c[i].equals("6")) {
            orden[5] = "" + (i + 1);
        }
        if (c[i].equals("7")) {
```

```
            orden[6] = "" + (i + 1);
        }
        if (c[i].equals("8")) {
            orden[7] = "" + (i + 1);
        }
        if (c[i].equals("9")) {
            orden[8] = "" + (i + 1);
        }
        if (c[i].equals("10")) {
            orden[9] = "" + (i + 1);
        }
    }
```

Cantidad de pasos= 10(10)+10=110 = **O(1) complejidad R//**

→Guardar Resultados:

```
for (int i = 0; i < apostadores.length; i++) {//O(n) pasos
            if (apostadores[i] != null) {//2 pasos
                subirResultados.write("Nombre: " + apostadores[i].getNombre() + ",
Monto"+apostadores[i].getMonto()+", Punteo: "+apostadores[i].getPunteo()+"\n");
            }
        }
for (int i = 0; i < apostadores.length; i++) {//O(n) pasos
            if (apostadores[i] != null) {//2
                subirResultados.write("Nombre: " + apostadores[i].getNombre() + ",
Monto"+apostadores[i].getMonto()+", Punteo: "+apostadores[i].getPunteo()+"\n");
            }
        }
```

Cantidad de pasos= n(2)+n(2)=2n+2n=4n = **O(n) complejidad R//**

→Cerrar apuestas y verificar error de léxico

```
while ((linea = leerTA.readLine()) != null) {//O(n) pasos
                if (linea.length() > 0) {//2 pasos
                    String[] espacio = linea.split(",");
                    if (espacio.length == 12) {//1 paso
                        try {//10 pasos
                            double a = Double.parseDouble(espacio[1].replaceAll(" ", ""));
                            int b = Integer.parseInt(espacio[2].replaceAll(" ", ""));
                            int c = Integer.parseInt(espacio[3].replaceAll(" ", ""));
                            int d = Integer.parseInt(espacio[4].replaceAll(" ", ""));
                            int e = Integer.parseInt(espacio[5].replaceAll(" ", ""));
                            int f = Integer.parseInt(espacio[6].replaceAll(" ", ""));
                            int g = Integer.parseInt(espacio[7].replaceAll(" ", ""));
                            int h = Integer.parseInt(espacio[8].replaceAll(" ", ""));
                            int i = Integer.parseInt(espacio[9].replaceAll(" ", ""));
                            int j = Integer.parseInt(espacio[10].replaceAll(" ", ""));
                            int k = Integer.parseInt(espacio[11].replaceAll(" ", ""));
                            apostadoresTemporal[contador] = (new Apostador(espacio[0],//1paso
                                    a, b,
                                    c, d,
                                    e, f,
                                    g, h,
                                    i, j,
                                    k, Integer.parseInt("0")));
                            contador++;//1paso
                        } catch (NumberFormatException e) {
                            verificador = false;
                            apuestasRechazadas.write("Apuesta: " + linea + " rechazada por caracteres
incorrecto en el apartado de posiciones \n");
                        }
```

```
                } else {
                    verificador = false;
                    apuestasRechazadas.write("Apuesta: " + linea + " rechazada por no contar con todos los campos solicitados \n");
                    //Character.is(espacio[0]);
                }
            }
        }
```

Cantidad de pasos= n(2+1+10+1+1)=25n = **O(n) complejidad R//**

Complejidad de servicios críticos (reducido):

→Ingreso de las apuestas antes del inicio de la carrera O(1)

File reader, buffer reader y una variable String ***//3 pasos***

Ciclo while con append para el área de lectura***//***sería n, pero como se mencionó que no se contaría entonces **1 paso**

Cantidad de pasos= 4=1 = **O(1) complejidad R*//***

→Verificación de apuestas O(n)

Ciclo for con limíte de arreglo de apostadores temporal ***//n paso***

Variable boolean y un if ***//2 pasos***

Pasar variables a arreglo espacio[] ***//10 pasos***

Ciclo for con limite constante de 0 a 9 y condicionales if y verificación de repitencia***// 9,8,6,5,4,3,2,2 pasos*** respectivamente en reducción en su pasar

Comprobar si hubo repitencia y crear objeto ***// 3 pasos***

Cantidad de pasos= n(1+2+10+9+8+7+6+5+4+3+2+2+3)=62n = **O(n) complejidad R*//***

→Cálculo de los resultados al finalizar la carrera O(n)

Ciclo for con limite apostadores que hay ***//n pasos***

Ciclo if ***//1 paso***

Ciclo if con algo interno***// 2 pasos***

Ciclo if con modificación de punteo si cumple ***//***2 pasos cada uno y son 10, entonces **20 pasos**

Cantidad de pasos= n(1+2+20)=23n = **O(n) complejidad R*//***

→Ordenamiento de resultados O(n^2)

Método burbuja en orden por punteo:

Crear variables necesarias boolean y arreglo ***//3 pasos***

Ciclo for con condicional distinto de variable ordenar***//n pasos***, esto pudo haber sido un while perfectamente

Ciclo for con cantidad limite de apostadores en arreglo ***//n pasos***

Condcional if verificando existencia ***//1 paso***

Condicional if con datos internos para validación de orden con verificación de punteo***// 5 pasos***

Condicional if para verificar si ya está ordenado el arreglo ***//2 pasos***

Reinicio de variable para condicional principal en for ***// 1 paso***

Cantidad de pasos= 3+n(n(1+5+2+1)= 3+9n^2 = **O(n^2) complejidad R*//***

Método burbuja en orden alfabético:

Crear variables necesarias boolean y arreglo **//3 pasos**

Ciclo for con condicional distinto de variable ordenar**//n pasos**, esto pudo haber sido un while perfectamente

Ciclo for con cantidad limite de apostadores en arreglo **//n pasos**

Condcional if verificando existencia **//1 paso**

Condicional if con datos internos para validación de orden usando el nombre con un compare to **// 5 pasos**

Condicional if para verificar si ya está ordenado el arreglo **//2 pasos**

Reinicio de variable para condicional principal en for **// 1 paso**

Cantidad de pasos= $3+n(n(1+5+2+1)= 3+9n^2$ = **O(n^2) complejidad R//**

Complejidad métodos implementados (reducido) :

→Verificar repitencia en datos ingresados como ganadores

Ciclo for constante de 0 a 10 y condicionales if internas//10 pasos, pero este pasa 10 veces 10(10), y en el peor de los casos si ejecuta el if, 10 pasos más = **110 pasos**

Cantidad de pasos= 10(10)+10=110 = **O(1) complejidad R//**

→Guardar Resultados:

Ciclo for con limite de apostadores en el arreglo **//n pasos**

Condicional if con dato interno para escritura **//2 pasos**

Ciclo for con limite de apostadores en el arreglo **//n pasos**

Condicional if con dato interno para escritura **//2 pasos**

Cantidad de pasos= n(2)+n(2)=2n+2n=4n = **O(n) complejidad R//**

→Cerrar apuestas y verificar error de léxico

Ciclo while para leer el text área **// n pasos**

Condicional if y creación de variable con separación por comas **// 2 pasos**

Condicional if verifcar cantidad 12 de campos ingresados **//1 paso**

Conversión de datos string a double y entero para ver si no hay error en léxico **//10 pasos**

Crear objeto apostador temporal **// 1 paso**

Incremento contador**//1paso**

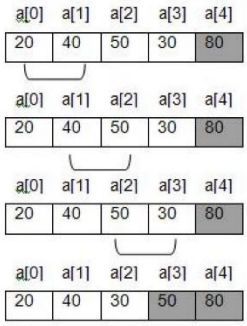
Agregar dato a archivo de apuestas rechazadas **//1 paso**

Cantidad de pasos= n(2+1+10+1+1+1)=26n = **O(n) complejidad R//**

Método de ordenamiento argumentación:

Utilicé el método de burbuja para la ordenación ya que en el peor de los casos el primer ciclo ejecutaría una condicional la cantidad de n veces y el segundo ciclo lo mismo en su peor de los casos va a ejecutar hasta el tamaño del arreglo, eso sí, el primero puede disminuir si ya casi que al principio tenemos los datos sin ordenar y los demás ordenados, al igual que el segundo ciclo si detecta que ya quedó ordenado antes de tiempo finaliza, por lo tanto en su peor escenario sería n^2 pero en el mejor de los casos menor a eso.

Adjuntando imagen para representación de mi argumentación:



Donde ordenamos 20 y 40 en posición 0 y 1, entonces miramos que están en orden, comprobamos 40 y 50 en posición 1 y 2, miramos que todo bien, entonces comprobamos 50 y 30 en la posición 2 y 3, vemos que no están en orden, intercambiamos posiciones con una variable temporal, 50 y 80 son los mayores encontrados se quedan ahí, lo que haría que ya pasara a comprobar lo demás, si ve que ya está en orden, solito lo detecta.

Por lo tanto, usé este método por su facilitación en realizarlo por la computadora y eficaz para arreglos de un gran tamaño por el hecho que se disminuye cuando ya encontró algo de importancia, y así cumpliendo con la complejidad que se solicitaba.