Documentación Técnica.

Servicios Críticos

Complejidad para agregar una apuesta

```java
public void agregarApuestas(Apuesta[] apuestas, int numero, String linea) {
    Apuesta nuevaApuesta = new Apuesta();
    int[] lugares = new int[10];
    double pasos = 0;
    try {
        long startTime = System.currentTimeMillis();
        int comas[] = new int[11];
        int x = 0;
        for (int i = 0; i < linea.length() - 1; i++) {
            pasos++;
            int j = i + 1;
            if (",".equals(linea.substring(i, j))) {
                pasos++;
                comas[x] = i;
                x++;
            }
        }
        String nombre = linea.substring(1, comas[0] - 1);
        double monto = Double.parseDouble(linea.substring(comas[0] + 2, comas[1] - 1));
```

```java
        lugares[0] = Integer.parseInt(linea.substring(comas[1] + 2, comas[2] - 1));
        lugares[1] = Integer.parseInt(linea.substring(comas[2] + 2, comas[3] - 1));
        lugares[2] = Integer.parseInt(linea.substring(comas[3] + 2, comas[4] - 1));
        lugares[3] = Integer.parseInt(linea.substring(comas[4] + 2, comas[5] - 1));
        lugares[4] = Integer.parseInt(linea.substring(comas[5] + 2, comas[6] - 1));
        lugares[5] = Integer.parseInt(linea.substring(comas[6] + 2, comas[7] - 1));
        lugares[6] = Integer.parseInt(linea.substring(comas[7] + 2, comas[8] - 1));
        lugares[7] = Integer.parseInt(linea.substring(comas[8] + 2, comas[9] - 1));
        lugares[8] = Integer.parseInt(linea.substring(comas[9] + 2, comas[10] - 1));
        lugares[9] = Integer.parseInt(linea.substring(comas[10] + 2, linea.length() - 1));

        nuevaApuesta.setNombre(nombre);
        nuevaApuesta.setMonto(monto);
        nuevaApuesta.setOrden(lugares);
        nuevaApuesta.setValidacion(true);
        pasos += 10;
        long endTime = System.currentTimeMillis();
        reporte.setTiempoIngreso(((endTime - startTime) / 1000));
        reporte.setPasosIngreso(pasos);
    } catch (Exception e) {
        nuevaApuesta.setError("Datos Faltantes");
        nuevaApuesta.setValidacion(false);
    }
    apuestas[numero] = nuevaApuesta;
```

La complejidad es de O (1) ya que para agregar una apuesta el promedio de pasos es de 90 para todas las apuestas.

Complejidad para verificar una apuesta

```java
public double comprobarRepetidos(Apuesta apuesta) {
    double nuevos_pasos = 0;
    int[] orden = apuesta.getOrden();
    boolean repetido = false;
    for (int i = 1; i <= 10; i++) {
        int repeticiones = 0;
        nuevos_pasos++;
        for (int j = 0; j < 10; j++) {
            nuevos_pasos++;
            if (orden[j] == i) {
                nuevos_pasos++;
                repeticiones++;
            }
        }
        if (repeticiones > 1) {
            nuevos_pasos++;
            repetido = true;
        }
    }
    if (repetido == true) {
        nuevos_pasos++;
        apuesta.setValidacion(false);
        apuesta.setError("La apuesta tiene valores repetidos");
    }
    return nuevos_pasos;
```

La complejidad es de O(n). Ya que, aunque se utilizan dos ciclos for, estos siempre son de 100 pasos cada uno. Por lo que estes 100 pasos se realizaran n veces donde n es el numero de apuestas que se verificara. La complejidad seria entonces de un 100n

Complejidad para agregar punteos a una apuesta.

```
private void botonVerResultadosActionPerformed(java.awt.event.ActionEvent evt) {
    double pasos = 0;
    long startTime = System.currentTimeMillis();
    for (int i = 0; i < 10; i++) {
        pasos++;
        for (int j = 0; j < apuestas.length; j++) {
            pasos++;
            if (apuestas[j].isValidacion() == true) {
                pasos++;
                int[] orden = apuestas[j].getOrden();
                if (resultados[i] == orden[i]) {
                    pasos++;
                    if (i == 0) {
                        apuestas[j].aumentarPunteo(10);
                    } else if (i == 1) {
                        apuestas[j].aumentarPunteo(9);
                    } else if (i == 2) {
                        apuestas[j].aumentarPunteo(8);
                    } else if (i == 3) {
                        apuestas[j].aumentarPunteo(7);
                    } else if (i == 4) {
                        apuestas[j].aumentarPunteo(6);
                    } else if (i == 5) {
                        apuestas[j].aumentarPunteo(5);
```

```
                    } else if (i == 6) {
                        apuestas[j].aumentarPunteo(4);
                    } else if (i == 7) {
                        apuestas[j].aumentarPunteo(3);
                    } else if (i == 8) {
                        apuestas[j].aumentarPunteo(2);
                    } else if (i == 9) {
                        apuestas[j].aumentarPunteo(1);
                    }
                }
            }
        }
    long endTime = System.currentTimeMillis();
    reporte.setTiempoResultados(((endTime - startTime) / 1000));
    reporte.setPasosResultados(pasos);
    ordenarPorPunteo();
    botonVerResultados.setEnabled(false);
}
```

La complejidad es de O(n). Al igual que el servicio critico anterior, este utiliza dos ciclos for. Pero uno es constante ya que solo recorre los lugares de los

caballos que son 10, y el otro for recorre la cantidad de apuestas. Por lo que los pasos serian 10n.

Complejidad ordenamiento por punteo.

```
public void ordenarPorPunteo() {
    long startTime = System.currentTimeMillis();
    double pasos = 0;
    areaResultadosApuestas.setText(null);
    int posicion;
    Apuesta menor = new Apuesta();
    Apuesta auxiliar = new Apuesta();

    for (int i = 0; i < apuestas.length; i++) {
        pasos++;
        menor = apuestas[i];
        posicion = i;
        for (int j = i + 1; j < apuestas.length; j++) {
            pasos++;
            if (apuestas[j].getPunteo() < menor.getPunteo()) {
                menor = apuestas[j];
                posicion = j;
                pasos++;
            }
        }
        if (posicion != i) {
            pasos++;
            auxiliar = apuestas[i];
            apuestas[i] = apuestas[posicion];
            apuestas[posicion] = auxiliar;
```

Para este método se utiliza un ordenamiento por selección. El cual se elogio por ser menos complejo que el ordenamiento tipo burbuja y ser mas efectivo para ordenar cantidades. Tiene una complejidad de $O(n^2)$ ya que maneja dos ciclos for.

Complejidad ordenamiento por nombre.

```
public void ordenarPorNombre() {
    long startTime = System.currentTimeMillis();
    areaResultadosApuestas.setText(null);
    double pasos = 0;
    for (int i = 0; i < apuestas.length; i++) {
        pasos++;
        for (int j = 0; j < apuestas.length - 1; j++) {
            if (apuestas[j].getNombre().compareTo(apuestas[j + 1].getNombre()) > 0) {
                Apuesta auxiliar;
                auxiliar = apuestas[j];
                apuestas[j] = apuestas[j + 1];
                apuestas[j + 1] = auxiliar;
                pasos++;
            }
        }
    }
```

Para este método se utiliza el ordenamiento de tipo burbuja. El cual se considero como el mas efectivo para ordenar alfabéticamente. La complejidad de este algoritmo es de $O(n^2)$. Porque de igual manera utiliza dos ciclos for.

Otras funciones

Obtener las líneas del texto.

Para obtener las líneas del área de texto se utiliza una complejidad de O(n)

```
public String[] obtenerLineas(int numeroLineas, String texto) {
    String[] lineas = new String[numeroLineas];
    int lineasContadas = 0;
    int ultima = 0;
    for (int i = 0; i < texto.length() - 1; i++) {
        if (texto.substring(i, i + 1).equals("\n")) {
            lineas[lineasContadas] = texto.substring(ultima, i);
            lineasContadas++;
            ultima = i + 1;

        }
    }
    lineas[lineasContadas] = texto.substring(ultima, texto.length() - 1);
    return lineas;
}
```

Contar Líneas

Para contar las líneas del área de texto se utiliza una complejidad de O(n)

```
public int contarLineas() {
    int linea = 0;
    String texto = areaTexto1.getText();
    for (int i = 0; i < texto.length() - 1; i++) {
        if (texto.substring(i, i + 1).equals("\n")) {
            linea++;
        }
    }
    linea++;
    return linea;
}
```