Documentación técnica.

Ingreso de las apuestas antes del inicio de la carrera (servicio crítico):

```
public void agregar(T valor){

    if(primerNodo == null){
        primerNodo = new NodoApuesta<>(valor, anterior: null, siguiente: null);
        ultimoNodo = primerNodo;
    } else {

        if(ultimoNodo == primerNodo){
            ultimoNodo = new NodoApuesta<>(valor, primerNodo, siguiente: null);
            primerNodo.setSiguiente(ultimoNodo);
        } else {
            NodoApuesta<T> temporal = ultimoNodo;
            ultimoNodo = new NodoApuesta<>(valor, temporal, siguiente: null);
            temporal.setSiguiente(ultimoNodo);
        }
    }

}
```

Como se puede ver en el algoritmo se utiliza un valor genérico para poder hacer genérica nuestra lista. Además este método contiene solamente sentencias if's que si contamos el total de operaciones totales podremos darnos cuenta que **es un O(1) cada que se ejecuta este método**, ya que si no se ejecuta una condición se ejecuta otra.

Eliminación de una apuesta:

```
public void eliminar(NodoApuesta<T> nodo){

    if(primerNodo == ultimoNodo){
        primerNodo = null;
        ultimoNodo = null;

    } else if (nodo == ultimoNodo){
        ultimoNodo = ultimoNodo.getAnterior();
        ultimoNodo.setSiguiente(null);

    } else if(nodo == primerNodo){
        primerNodo = primerNodo.getSiguiente();
        if(primerNodo.getAnterior() != null){
            primerNodo.setAnterior(null);
        }
    } else {
        nodo.getAnterior().setSiguiente(nodo.getSiguiente());
        nodo.getSiguiente().setAnterior(nodo.getAnterior());
    }
    System.out.println("nodo eliminado = " + nodo);
}
```

Si se observa detenidamente se puede dar cuenta que cuenta con la misma idea que con la agregación de las apuestas. Este método solamente cuenta con sentencias if. Así que podemos decir que por cada sentencia if **nuestro big-O es un O(1).**

Métodos insertar (inicio y entre):

```
public void insertarInicio(NodoApuesta<T> nodoInsertar){
    if(this.ultimoNodo == nodoInsertar){
        nodoInsertar.getAnterior().setSiguiente(null);
        this.ultimoNodo = nodoInsertar.getAnterior();
    } else {
        nodoInsertar.getAnterior().setSiguiente(nodoInsertar.getSiguiente());
        nodoInsertar.getSiguiente().setAnterior(nodoInsertar.getAnterior());
    }
    this.primerNodo.setAnterior(nodoInsertar);
    nodoInsertar.setSiguiente(this.primerNodo);
    nodoInsertar.setAnterior(null);
    this.primerNodo = nodoInsertar;
}


public void insertarEntre(NodoApuesta<T> primerNodo, NodoApuesta<T> segundoNodo, NodoApuesta<T> nodoInsertar){
    if(this.ultimoNodo == nodoInsertar){
        nodoInsertar.getAnterior().setSiguiente(null);
        this.ultimoNodo = nodoInsertar.getAnterior();
    } else {
        nodoInsertar.getAnterior().setSiguiente(nodoInsertar.getSiguiente());
        nodoInsertar.getSiguiente().setAnterior(nodoInsertar.getAnterior());
    }

    primerNodo.setSiguiente(nodoInsertar);
    nodoInsertar.setAnterior(primerNodo);
    segundoNodo.setAnterior(nodoInsertar);
    nodoInsertar.setSiguiente(segundoNodo);

}
```

Estos métodos también cuentan con una lógica muy simple, sentencia if-else por lo que **el cálculo de estos algoritmos es un O(1) por cada if.**

Cálculo de resultados al finalizar la carrera (Servicio crítico):

```java
public class Calculador {


    public static void calcularPunteo(NodoApuesta<Apuesta> nodoApuesta, Resultado resultado){

        if(nodoApuesta != null){
            nodoApuesta.getValor().calcularPuntos(resultado);
            if(nodoApuesta.getSiguiente() != null){
                calcularPunteo(nodoApuesta.getSiguiente(), resultado);
            }
        }
    }
}
```

```java
    public void calcularPuntos(Resultado resultado){
        for(int i = 0; i < resultado.getResultado().length; i++){
            if(this.getResultado().getResultado()[i] == resultado.getResultado()[i]){
                this.setPuntos(10-i);
            }
        }
    }
```

Para este servicio crítico se decidió usar un método recursivo que es “calcularPunteo”. Este al ser recursivo en el peor de los casos podemos hacer un cálculo de O(n) donde n son todos los nodos que se tienen en la lista. Además de esto dentro de este método se llama al método “calcularPuntos” de nuestra clase Apuesta. Como se puede ver en este método contamos con un ciclo for que es el encargado de recorrer el arreglo de resultados de la apuesta para hacer el cálculo del punteo del apostador. Sin embargo este no es un O(n) ya que en el peor de los este ciclo solo será recorrido 10 veces que es el tamaño del arreglo de resultados de nuestra apuesta **por lo que nuestro big-O sigue siendo un O(n).**

Verificación de la Apuesta (Servicio Crítico):

```
private void verificarApuestas(NodoApuesta<Apuesta> nodoApuesta){
    if(nodoApuesta != null){
        verificarResultados(nodoApuesta.getValor().getResultado(), nodoApuesta);
        if(nodoApuesta.getSiguiente() != null){
            verificarApuestas(nodoApuesta.getSiguiente());
        }
    }
}

private void verificarResultados(Resultado resultado, NodoApuesta<Apuesta> nodo){

    if(tieneRepetido(resultado.getResultado(), inicio: 0) || tieneRepetido(resultado.getResultado(), inicio: 1)
    || tieneRepetido(resultado.getResultado(), inicio: 2) || tieneRepetido(resultado.getResultado(), inicio: 3)
    || tieneRepetido(resultado.getResultado(), inicio: 4) || tieneRepetido(resultado.getResultado(), inicio: 5)
    || tieneRepetido(resultado.getResultado(), inicio: 6) || tieneRepetido(resultado.getResultado(), inicio: 7)
    || tieneRepetido(resultado.getResultado(), inicio: 8) || tieneRepetido(resultado.getResultado(), inicio: 9)){
        lista.eliminar(nodo);
        this.rechazadas.agregar(nodo.getValor());
    }
}
```

```
public boolean tieneRepetido(int[] resultados, int inicio){

    for(int i = inicio+1; i < resultados.length; i++){
        if(resultados[inicio] == resultados[i] || resultados[inicio] > 10 || resultados[inicio] < 1){
            return true;
        }
    }

    return false;
}
```

Para la verificación de las apuestas se decidió hacer uso de tres métodos. El primero llamado “verificarApuestas” tiene un big-O de O(n) ya que este es recursivo llamandose a sí mismo para cada uno de los nodos de nuestra lista. El segundo método “veriicarResultado” contiene un if con varias operaciones o (||) para hacer la verificación en el último método llamado “tiene repetido”. Este método a simple vista parece que tiene on big-O de O(n) ya que tiene un ciclo for. Sin embargo el mismo caso anterior se aplica aquí ya que en el peor de los casos nuestro ciclo será recorrido solamente 10 veces. Ahora en el caso de la sentencia if del segundo método este llama 10 veces al método “tieneRepetido” por lo que en el peor de los casos el big-O de este método sería un O(100) y nada más. Esto juntando el O(n) del método recursivo de arriba **podríamos decir que tendríamos un O(n)** ya que en casos muy grandes las constantes pasan a segundo plano y lo importante es la cantidad n de veces que se repite un método o una sentencia en el caso de ser un for.

Ordenamiento de resultados de las apuestas (servicio crítico):

```java
public void ordenar(){
    if(this.orden == Orden.NUMERICO){
        System.out.println("orden = " + orden);
        this.ordenarNum();
    } else {
        System.out.println("entra acá");
        this.ordenarAlf();
    }
}
```

Para la solución de este servicio se hizo uso de dos métodos para los distintos ordenamientos que se podían implementar pero antes de pasar a la explicación de los métodos quisiera explicar el tipo de ordenamiento y la razón del porqué. El método utilizado fue el ordenamiento por inserción la forma en que este funciona es ir comparando los nodos o valores del arreglo por sus anteriores buscando si hay menor o mayor valor en uno de ellos para hacer la inserción entre 2 nodos dependiendo del caso. La razón por la que se utilizó este tipo de ordenamiento fue simplemente por comodidad. Al haber implementado una lista manual y no un arrayList o un arreglo estático que nos permitiera el acceso rápido a los elementos por medio de un método get, en el caso del arrayList y la obtención de un valor de un arreglo estático, se nos hacía complicada la realización del método de burbúja o la burbúja bidimensional. Ambos métodos de ordenamiento cumplían con ser $O(n^2)$ en el peor de los casos igual que el método por inserción pero la forma de implementarlos hubiera sido complicada por cómo se manejaba la lista.

```java
private void ordenarNum(){
    NodoApuesta<Apuesta> nodoInsertar = lista.getPrimerNodo().getSiguiente();

    while(nodoInsertar != null){
        NodoApuesta<Apuesta> primerNodo = nodoInsertar.getAnterior();

        while(primerNodo != null){

            if(primerNodo.getValor().getPuntos() >= nodoInsertar.getValor().getPuntos()){
                if(primerNodo.getSiguiente().getValor().getPuntos() < nodoInsertar.getValor().getPuntos()){
                    this.lista.insertarEntre(primerNodo, primerNodo.getSiguiente(), nodoInsertar);
                }
                break;
            } else {
                if(this.lista.getPrimerNodo() == primerNodo){
                    this.lista.insertarInicio(nodoInsertar);
                }
            }

            primerNodo = primerNodo.getAnterior();
        }
        nodoInsertar = nodoInsertar.getSiguiente();
    }

}
```

Como se explicó con anterioridad este método implementa el ordenamiento por inserción por lo que ya se sabe que en el peor de los casos su big-O es de $O(n^2)$. Pero ¿Por qué es un n^2? Esa pregunta puede ser respondida ya que si analizamos el método podremos ver que se tiene un ciclo while dentro de otro ciclo while. El primero va a ir desde el segundo nodo de nuestra lista hasta el final de la lista. Entonces en el peor de los casos ese ciclo tiene una complejidad de O(n-1) donde n es el valor total de nodos que contiene nuestra lista y 1 es porque partimos del segundo nodo como indica el algoritmo. El segundo while va a ir desde el valor N hasta el principio de la lista, en reversa. El calculo de su big-O es el de O(n-1) ya que parte desde el valor N ingresado en el primer while hasta el primer nodo. El 1 es ya que el while no comienza desde N sino desde el nodo anterior a N. Dentro del while anidado se encuentra un par de sentencias if que llaman a los métodos “insertarEntre” e “insertarInicio” de los que ya se explicó su big-O anteriormente. Sin embargo estos big-O quedan ignorados ya que lo que importa es saber que **el big-O total es de $O(n^2)$**. Claro esto aplica para el método que se mostrará a continuación.

```
private void ordenarAlf(){
    Collator comparador = Collator.getInstance();
    comparador.setStrength(Collator.PRIMARY);

    NodoApuesta<Apuesta> nodoInsertar = lista.getPrimerNodo().getSiguiente();

    while(nodoInsertar != null){
        NodoApuesta<Apuesta> primerNodo = nodoInsertar.getAnterior();

        while(primerNodo != null){
            if(comparador.compare(nodoInsertar.getValor().getNombreApostador(), primerNodo.getValor().getNombreApostador()) < 0){
                if(primerNodo == this.lista.getPrimerNodo()){
                    this.lista.insertarInicio(nodoInsertar);
                } else {
                    if(comparador.compare(nodoInsertar.getValor().getNombreApostador(), primerNodo.getAnterior().getValor().getNombreApostador()) >= 0){
                        this.lista.insertarEntre(primerNodo.getAnterior(), primerNodo, nodoInsertar);
                        break;
                    }
                }
            }

            primerNodo = primerNodo.getAnterior();
        }
        nodoInsertar = nodoInsertar.getSiguiente();
    }
}
```

Generación de Archivo csv:

```
public void generarArchivo(String columnas) throws IOException {
    FileDialog guardar = null;
    guardar = new FileDialog(guardar, title: "Guardar como", FileDialog.SAVE);
    guardar.setVisible(true);
    guardar.dispose();
    if(guardar.getFile() != null && guardar.getDirectory() != null){
        FileWriter writer;
        if(!Objects.equals(FilenameUtils.getExtension(guardar.getFile()), b: "csv")){
            writer= new FileWriter( fileName: guardar.getDirectory()+guardar.getFile()+".csv");
        } else {
            writer = new FileWriter( fileName: guardar.getDirectory()+guardar.getFile());
        }
        String resultado = columnas+"\n";
        resultado += escribirArchivo(lista.getPrimerNodo());
        writer.write(resultado);
        writer.close();
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Archivo guardado con éxito en la ruta: " +
                "\n"+guardar.getDirectory(), title: "Información", JOptionPane.INFORMATION_MESSAGE);
    }
}

private String escribirArchivo(NodoApuesta<Apuesta> nodo) {
    String resultado = "";
    NodoApuesta<Apuesta> temporal = nodo;
    while (temporal != null){
        resultado+=temporal.getValor().getNombreApostador()+","+temporal.getValor().getMonto()
                +","+temporal.getValor().getPuntos()+","+temporal.getValor().getResultado().getResultado()[0]
                +"," +temporal.getValor().getResultado().getResultado()[1]+","+temporal.getValor().getResultado().getResultado()[2]
                +","+temporal.getValor().getResultado().getResultado()[3]+"," +temporal.getValor().getResultado().getResultado()[4]
                +","+temporal.getValor().getResultado().getResultado()[5]+","+temporal.getValor().getResultado().getResultado()[6]
                +"," +temporal.getValor().getResultado().getResultado()[7]+","+temporal.getValor().getResultado().getResultado()[8]
                +","+temporal.getValor().getResultado().getResultado()[9]+"\n";
        temporal = temporal.getSiguiente();
    }
    return resultado;
}
```

Este algoritmo hace uso de 2 métodos. El primero que es generarArchivo es el encargado de pedir al usuario el ingreso de la ruta así como el nombre del archivo. Tiene un par de sentencias if por lo que se podría decir que tiene una complejidad O(1). Sin embargo hace uso del método “escribirArchivo” que como se puede apreciar usa un ciclo while. Al tener esto podemos darnos cuenta que en el peor de los casos el big-O de el método “escribirArchivo” es de O(n) en el que n es el valor de todos los nodos de las apuestas que se envían a escribir el archivo. El O(n) del segundo método hace que **el método principal tenga una complejidad en el peor de los casos (y en el único ya que sí o sí debe escribir todos y cada uno de los nodos) es de O(n).**