UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
INGENIERÍA EN CIENCIAS Y SISTEMAS

# PRIMERA PRÁCTICA: DOCUMENTACIÓN


**No. carné:** 202030799 **Nombre estudiante:** Manuel Antonio Rojas Paxtor


**Docente:** Oliver Ernesto Sierra Pac
**Materia:** Estructura de datos
**Fecha:** 07-03-2022

Quetzaltenango, Guatemala

## Ingreso de apuestas

### Cálculo de complejidad del algoritmo

```
   /**
     * Método para ingresar una nueva apuesta. Tiene un tiene un time
complexity de O(1).
    * @param value la apuesta que se va a ingresar a la lista.
    */

   public void push(Apuesta value){                                    //1
       if (first == null){                                             //1
           last = new Nodo<>(value);                                   //1
           first = last;                                               //1
       }else{
           Nodo<Apuesta> newValue = new Nodo<>(value, null, last);     //1
           last.setNext(newValue);                                     //1
           last = newValue;                                            //1
       }
       this.size++;                                                    //1
   }
```

O (n) = 1+1+3+1 = 6
**Complejidad**: O(1)

### Explicación y argumentación del algoritmo

Para el ingreso de las apuestas se optó por una lista enlazada, ya que son más eficientes cuando se trata de ingresar un nuevo dato sin conocer cuántos realmente se ingresarán: así como podrían ser unos cuantos datos, podrían ser miles; y este tipo de estructura de datos es eficiente en ese sentido.
En método se puede ver como se considera como 1 paso desde la declaración del parámetro del método, luego la comparación que debe hacerse, y en el peor de los casos (que sería el else) se tendrían 3 pasos más. Se agrega un último paso por el incremento del contador, dando como resultado una complejidad de 6 pasos.

## Verificación de apuesta (Individual)

### Cálculo de complejidad del algoritmo

```java
   /**
     * Método que valida si la apuesta es válida o no. Posee un time
complexity de
    * O(n)
    *
    * @return retorna false si existe algún caballo repetido dentro de
la apuesta,
    *         de lo contrario es true.
    */

   public boolean isValid() {
       for (int i = 0; i < horses.length - 1; i++) {     //2n+1
           if (horses[i].equals(horses[i + 1])) {        //(n-1)*1
               return false;                            //(n-1)*1
           }
       }
       return true;
   }
```

O(n) = (2n+1)+(n-1)+1 = 2n+n+1-1 = 3n

Se sabe que n = número de caballos en el arreglo, el cual es siempre 10.
Por lo tanto O(n) = 3(n) = 30

**Complejidad**: O(1)

### Explicación y argumentación del algoritmo

En este algoritmo se ve como se recorre el arreglo de los caballos en busca de algún elemento repetido. El truco para que este recorrido sea lineal y no cuadrático está en que el arreglo se encuentra previamente ordenado. De esta manera se sabe que mientras se va avanzando los que elementos del arreglo que ya se pasaron no se encuentran repetidos. El arreglo de caballos se ordena inmediatamente después de crear la apuesta para eficientar el proceso de datos.

## Cálculo de resultados (por apuesta)

### Cálculo de complejidad del algoritmo

```
   /**
     * Método que calcula el punteo obtenido por el apostador en función de
los
    * caballos a los que les apostó. Tiene un time complexity de O(n)
    *
     * @param ganadores arreglo de los caballos con los cuales se comparará
para
    *                  calcular los resultados.
    */
   public void calcularPunteo(Horse[] ganadores) {                      //1
       int punteo = 10;                                                 //1

       for (int i = 0; i < ganadores.length; i++) {                     //2n+2
           if (ganadores[i].getLugar() == horses[i].getLugar()) {       //n
               apostador.setPuntaje(punteo);                            //n
           }
           punteo--;                                                    //n
       }

   }
```

O(n) = 1+1+2n+2+3n = 5n+4
Se sabe que n = número de caballos en el arreglo, el cual es siempre 10.
Por lo tanto O(n) = 5(10)+4 = 54
**Complejidad**: O(1)

### Explicación y argumentación del algoritmo

El método para darle el puntaje a cada apostador de nuevo tiene un time complexity de O(1), esto debido a que el arreglo de caballos a los cuales se les apostó ya se encuentra ordenado, facilitando las validaciones. El arreglo del cual se pide como atributo también se asume que se encuentra ordenado por lo que es solo de ir comparando ambos arreglos de caballos, los cuales se sabe que son 10 siempre.

## Verificación de apuesta y cálculo de resultados (Total)

### Cálculo de complejidad del algoritmo

```
/**
 * Metodo para obtener las apuestas validas de todas las que se han ingresado.
 * Tiene un time complexity de O(n^2)
 * @param todas listado de las apuestas acumuladas.
 * @return Retorna una lista con las apuestas validas.
 */

public ListaEnlazada getValidApuestas(ListaEnlazada todas){                          //1
    ListaEnlazada validas = new ListaEnlazada();                                     //1
    while (todas.size() > 0) {                                                       //n
        Apuesta apuesta = todas.pop();                                               //1
        if (apuesta.isValid()) {                                                     //1*n
            apuesta.calcularPunteo(Hipodromo.COMPETIDORES);                          //1*n
            validas.push(apuesta);                                                   //1
        }else{
            System.out.println("Apuesta no valida de: " + apuesta.getApostador());   //1
        }
    }
    return validas;                                                                  //1
}
```

O(n) = 1+1+n+n+2(n)+1+1 = 4 + 4n
**Complejidad**: O(n)

### Explicación y argumentación del algoritmo

Para poder validar todas las apuestas ingresadas, el metodo hace dos llamdas a metodos propios de la clase apuesta: calcularPunteo() e isValid(). Dichos metodos tienen un time complexity de O(1) los cuales se encuentran dentro de un while, razón por la cual al final el cálculo final del time complexity método getValidApuestas() termina siendo un O(n).

## Ordenamiento de resultados

## Cálculo de complejidad del algoritmo

```
/**
 * Métod para ordenar la lista con las apuestas. Es del tipo insertion sort, tiene un time complexity de O(n2)
 * @param isAlfabetic valor que indica si se ordenará de manera alfabetica (true) o por punteo (false).
 */

public void insertionSort(boolean isAlfabetic) {                                                                    //1
    Nodo <Apuesta> anterior = first;                                                                                //1
    Nodo <Apuesta> actual;                                                                                          //1
    Apuesta temporal;                                                                                               //1

    while(anterior != null){                                                                                        //n
        actual = anterior;                                                                                          //n
        temporal = anterior.getValue();                                                                             //n

        while (actual != first                                                                                      //n(n)
                && (isAlfabetic
                ? temporal.getApostador().getName().compareTo(actual.getBefore().getValue().getApostador().getName()) < 0   //n(n)
                : temporal.getApostador().getPuntaje() > actual.getBefore().getValue().getApostador().getPuntaje())) {      //n(n)
            actual.setValue(actual.getBefore().getValue());                                                         //n(n)
            actual = actual.getBefore();                                                                            //n(n)
        }
        actual.setValue(temporal);                                                                                  //n
        anterior = anterior.getNext();                                                                              //n
    }
}
```

O(n) = 4+3n+5(n^2)+2n = 5n^2+5n+4
**Complejidad**: O(n^2)

## Explicación y argumentación del algoritmo

Para el ordenamiento de datos se optó por el ordenamiento por inserción ya que cuenta con un O(n) en el mejor de los casos y con un O(n^2) en el peor de los casos. El algoritmo resulta teniendo la constante 5 en el n^2 debido a las validaciones que se hacen para que el algoritmo sea más robusto a la hora de ordenar las apuestas. Pudiendo utilizar el mismo método para ordenar la lista de forma numérica (punteo) y de forma alfabética (por nombre).

## Proceso de texto de entrada

### Cálculo de complejidad del algoritmo

```java
/**
 * Metodo para validar los caballos del archivo de entrada.
 * @param posiciones posiciones obtenidas del archivo de entrada
 * @return listado de 10 caballos validos.
 */

private Horse[] validHorses(Double[] posiciones){                    //1
    Horse[] caballos = new Horse[10];                                //1
    for(int i = 0; i<caballos.length ; i++){                         //2n+2
        if(posiciones[i] > 0 && posiciones[i] < 11){                  //n
            int posicion = (int) posiciones[i].doubleValue();         //n
            Horse aux = Hipodromo.COMPETIDORES[posicion-1];           //n
            Horse agregar = new Horse(aux.getName(),aux.getNumero()); //n
            agregar.setLugar(i+1);                                    //n
            caballos[i] = agregar;                                    //n
        }else{
            return null;
        }
    }
    return caballos;                                                 //1
}
```

O(n) = 2+(2n+2)+6(n)+1=5+8n
Se sabe que n = número de caballos en el arreglo el cual es siempre 10, por lo tanto:
O(n) = 5+8(10) = 85
**Complejidad**: O(1)

### Explicación y argumentación del algoritmo

El procesamiento del archivo de entrada es realizado a través de jflex y cup. El único método que se utiliza durante el procesamiento del archivo de texto es este, el cual sirve para que sean validados únicamente las apuestas que hayan colocado números de caballos válidos. Al final, como se conoce que el número total de caballos, el time complexity es de O(1)