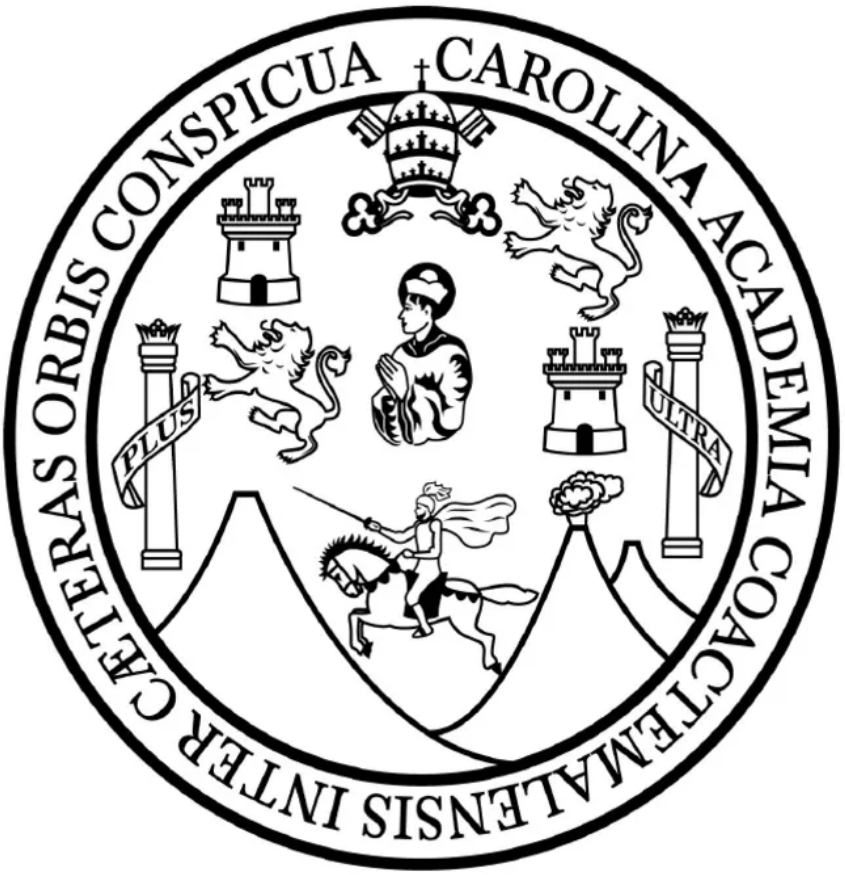
Universidad de San Carlos de Guatemala

Centro Universitario de Occidente
División De Ciencias de la Ingeniería



Manual Tecnico Practica01

Rudy Adolfo Pacheco Pacheco

Ing. Oliver Sierra

Calculo Complejidad servicios criticos

1. Ingreso de apuestas o(1)

```java
public void agregarApuesta(apuesta apuesta, apuesta[] apuestas) {
    apuestas[apuestas.length-1] = apuesta;
}
```

al ser solo una inserción el algoritmo para ingresar una apuesta se basa en solo agregar una línea del texto de entrada o de el ingreso por interfaz , teniendo asi un coste **o(1)**

2. Verificación de apuestas (n)

```java
public apuesta[] validarApuestas(apuesta[] apuestas) {
    long tiempoInicial = System.nanoTime();
    this.apuestas = apuestas;
    size = apuestas.length;
    for (int i = 0; i < apuestas.length; i++) {//n
        int[] posiciones = new int[10];
        for (int j = 0; j < 10; j++) {// 1

                if (posiciones[apuestas[i].getLista()[j] - 1] == apuestas[i].getLista()[j]) {
                    System.out.println("Numero repetido" + (apuestas[i].getLista()[j]));
                    rechazadas += "Numero repetido " + (apuestas[i].getLista()[j]) + "en apuesta " + (i + 1);
                    apuestas[i] = null;
                    size--;
                    break;
                } else {
                    posiciones[apuestas[i].getLista()[j] - 1] = apuestas[i].getLista()[j];
                    System.out.println("no repetido");
                }
        }
        tiempoT += (System.nanoTime() - tiempoInicial);
    }
    tiempoPromedioVerificacion = tiempoT / apuestas.length;
    return apuestas;

}
```

este algoritmo al tener al inicio un for que depende de la cantidad de apuestas totales , es o(n) el segundo for (for anidado) que está dentro de este únicamente se va a ejecutar 10 veces por lo que se vuelve un o(1) manteniendo así el coste de **o(n)**

3. Cálculo de resultados o(n)

```
public apuesta[] calcular(int[] orden, apuesta[] apuestas) {
      long tiempoI = System.nanoTime();
    for (int i = 0; i < apuestas.length; i++) {//n
        int ganado = 0;
        int[] temp = apuestas[i].getLista();
        apuesta apuestaT = apuestas[i];
        for (int j = 0; j < 10; j++) {// 1
            if (orden[j] == temp[j]) {
                ganado = ganado + (10 - j);
            }
        }
        apuestaT.setGanancia(ganado);
        System.out.println("Ganado " + ganado);
        tiempoT += (System.nanoTime() - tiempoI);
    }
    tiempoPromedio= tiempoT/ apuestas.length;
    return apuestas;

}
```

este algoritmo al igual que el anterior el primer for depende de la cantidad de apuestas existentes (apuestas totales) y el segundo nuevamente se va a ejecutar únicamente 10 veces manteniendo así el coste **o(n)**

4. Ordenamiento de resultados o(n^2)

```
public apuesta[] ordenarPorPuntos(apuesta[] apuestas) {
    this.tiempoPuntos = 0;
    this.tiempoPromedioPuntos = 0;
    apuesta aux;
    int posAux;
    for (int i = 0; i < apuestas.length; i++) {                                    //n
        long tiempoI = System.nanoTime();
        posAux = i;
        aux = apuestas[i];
        while ((posAux > 0) && (apuestas[posAux - 1].getGanancia() < aux.getGanancia())) {  //n
            apuestas[posAux] = apuestas[posAux - 1];
            posAux--;
        }
        apuestas[posAux] = aux;
        tiempoPuntos += (System.nanoTime() - tiempoI);
    }
    this.tiempoPromedioPuntos = tiempoPuntos / apuestas.length;
    return apuestas;
}
```

este algoritmo se trata del algoritmo insertionSort , al ser un algoritmo de ordenamiento ya tiene una complejidad de o(n^2) se prefirió este algoritmo por sobre otros ya que en el mejor de los casos el coste puede ser menor.