TIME COMPLEXITY: BIG O

> Servicio crítico 1: Ingreso de apuestas

```
/*SERVICIO CRITICO 1 -> O = O(1)*/
private void addApuesta(String[] datosApuesta){
    apuestasRecibidas.add(datosApuesta);
}
```

```
public void add(E contenido){//agrega al final
    if(!inicializarLista(contenido)){
        ultimoNodo.setNext(contenido);
        ultimoNodo = ultimoNodo.getNext();
    }
    size++;
}
```

```
private boolean inicializarLista(E contenido){
    if(size == 0){
        primerNodo = ultimoNodo = new Nodo<>(contenido);
        return true;
    }
    return false;
}
```

```
public void setNext(E contenido){
    siguiente = new Nodo(contenido);
}
```

>	Complejidad Objetivo -> O(1)	
>	Complejidad alcanzada: O(1)	
>	Descripción de la complejidad	
	O(n) =	asignación
	O(n) =	((comparacion) + (setNext)+asignacion)
	O(n) =	((comparacion + asignacion + return) + (asignacion) + asignacion)
	O(n) =	((1+1+1)+(1)+1
	O(n) =	5
	o(n) =	**O(1)**

> Servicio crítico 2: Verificación de apuestas

```
public void estudiarApuestasRecibidas(ListaEnlazada<String[]> apuestasRecibidas){
    while(!apuestasRecibidas.isEmpty()){//puesto que la inicializo para evitar que sea null...
        String[] apuesta = apuestasRecibidas.removeFirstElement().getContent();
        ListaEnlazada<Error> listadoErrores = new ListaEnlazada<>();//puesto que aunque no se hagan las demas revisiones, de todos modos no será ace

        if(apuesta.length > 2){//sino, quiere decir que solo tiene el monto y el nombre del postor
            int posicionActual = 0;//será útil para recorrer cada apuesta xD
            String[] listadoJinetes = new String [apuesta.length-2];
            System.arraycopy(apuesta, 2, listadoJinetes, 0, listadoJinetes.length);

            String jinetesEspecificados = "";
            int[] repeticiones = new int[10];

            while(posicionActual < listadoJinetes.length){
                if(listadoJinetes.length > 1){
                    if(jinetesEspecificados.contains(listadoJinetes[posicionActual])){
                        repeticiones[((listadoJinetes[posicionActual].equals(participantes[0])?0:(listadoJinetes[posicionActual].equals(participantes[1])?1
                            :(listadoJinetes[posicionActual].equals(participantes[2])?2:(listadoJinetes[posicionActual].equals(participantes[3])?3:
                            (listadoJinetes[posicionActual].equals(participantes[4])?4:(listadoJinetes[posicionActual].equals(participantes[5])?5
                            :(listadoJinetes[posicionActual].equals(participantes[6])?6:(listadoJinetes[posicionActual].equals(participantes[7])?7
                            :(listadoJinetes[posicionActual].equals(participantes[8])?8:9)))))))))] ++;//y así sigues con cada comparación...
                    }else{
                        jinetesEspecificados+= " " + listadoJinetes[posicionActual];//esto no lo vayas a quitar, sino no habrá con qué comparar xD
                    }
                }

                if(posicionActual == (listadoJinetes.length-1)){
                    revisarUltimosPosiblesErrores(apuesta[0], apuesta[1], listadoJinetes, repeticiones, listadoErrores);
                }
                posicionActual++;
            }
        }else{
            listadoErrores.add(new Error("0 posiciones apostadas", "No especificaste jinete alguno", "Debes indicar un orden de posiciones por el cual apostar"));
```

>	Complejidad Objetivo -> O(n)	
>	Complejidad alcanzada: O(n)	
>	Descripción de la complejidad	
	O(n) =	n_ciclos + n asignaciones + n creaciones +
		n_comparaciones + n asignaciones +
		n_copias + n asignaciones + n asignaciones +
		n*(10 ciclos + 10 comparaciones + 10(asignaciones + 10 comparaciones) +
		10 comparaaciones + 10 incrementos) + n else + n creaciones +
		n agregaciones
	O(n) =	n + n + n + n + n + n + n + n + n+ n(10 + 10 + 10*(11)+ 10 + 10) + n + n + n
	O(n) =	8n + n(20 + 110 + 20) + 3n
	O(n) =	11n + 150n
	O(n) =	161n
	o(n) =	**O(n)**

```
        apuestasRechazadas.add(new ApuestaErronea(apuesta[0], apuesta[1], listadoErrores));
      }
    }
}//Servicio crítico 2 [LISTO] -> O(n) quitándole las ctes xD uwu
```

> Servicio crítico 3: Cálculo de resultados

```
public void calcularResultadosApuestas(String[] posicionesFinales){//se recibirá el ar
  if(!this.apuestasAceptadas.isEmpty()){
    Nodo<Apuesta> nodoApuesta = apuestasAceptadas.getFirst();

    do{//puesto que se eliminará y add al final porque requiero que estos datos
      int apuestaActual = 0;
      Apuesta apuesta = nodoApuesta.getContent();
      String[] ordenApostado = apuesta.getOrdenApostado();

      while(apuestaActual  < posicionesFinales.length){
        if(ordenApostado[apuestaActual].equals(posicionesFinales[apuestaActual])){
          int montoGanado = this.maximoMonto-apuestaActual;
          apuesta.setResultados(apuestaActual, String.valueOf(montoGanado));
          apuesta.incrementarTotalGanado(montoGanado);
        }else{
          apuesta.setResultados(apuestaActual, "-");
        }
        apuestaActual++;
      }
    }while((nodoApuesta = nodoApuesta.getNext()) != null);
  }

  for (int revision = 0; revision < apuestasAceptadas.getSize(); revision++) {
    System.out.println((revision+1) + " "+ apuestasAceptadas);//solo es para ver qu
  }
}//Servicio crítico 3 [LISTO] -> O(n) uwu xD
//no requiero devolver algo, puesto que los datos se actualizan indirectamente
```

> Complejidad Objetivo -> O(n)

> Complejidad alcanzada: O(n)

> Descripción de la complejidad

O(n) =	1 comparacion + (1 obtencion) + n ciclos + n asignaciones + n(obtenciones) +
	n(obtenciones) + n(10 comparaciones + 10 comparaciones + 10 asignaciones +
	10 sets + 10 incrementos + 10 incrementos)
O(n) =	1 + (1) + n + n + n(1) + n(1) + n(10 + 10 + 10 + 10 +10 + 10)
O(n) =	2 + 2n + 2n + n(60)
O(n) =	64n + 2
O(n) =	**O(n)**

Nota: el for de abajo solo es para revisión xD, no cuenta como parte funcional del algoritmo

```
public E getContent(){//
  return contenido;
}
```

> Servicio crítico 4: Ordenamiento de resultados

```
private void orderByName(ListaEnlazada<Apuesta> listadoApuestas){
  Nodo<Apuesta> anterior;
  Nodo<Apuesta> nodoSiguiente;

  //Burbuja
  for (int rondaActual = 0; rondaActual < (listadoApuestas.getSize()-1); rondaActual++) {//puesto que solo se req
    anterior = listadoApuestas.getFirst();

    while((nodoSiguiente = anterior.getNext()) != null){//esto garantiza que el nodo "anterior" llegue a la posi
      if(anterior.getContent().getNombrePostor().compareTo(nodoSiguiente.getContent().getNombrePostor()) > 0){//
        Apuesta temporal = anterior.getContent();
        anterior.resetContent(nodoSiguiente.getContent());
        nodoSiguiente.resetContent(temporal);
      }
```

> Complejidad Objetivo -> O(n²)

> Complejidad alcanzada: **O(n²)**

> Descripción de la complejidad

O(n) =	1 declaracion + 1 declaracion + (n asignaciones + n comparaciones + n incrementos)
	+ n(1 comparacion + n-1 comparaciones + 1 obtencion + 1 reseteo + 1 reseteo + 1 asignacion)
O(n) =	2 + (3n) + n(4 + n)
O(n) =	2 + 30 + 4n + n²
O(n) =	**O(n²)**

```
            anterior = nodoSiguiente;
        }
    }
}//Servicio crítico 4.1 -> O(n) uwu menor de lo pedido xD xD
```

```
private void orderByGains(ListaEnlazada<Apuesta> listadoApuestas){
    Nodo<Apuesta> anterior;
    Nodo<Apuesta> nodoSiguiente;

    //Burbuja
    for (int rondaActual = 0; rondaActual < (listadoApuestas.getSize()-1); rondaActual++) {//puesto q
        anterior = listadoApuestas.getFirst();

        while((nodoSiguiente = anterior.getNext()) != null){//esto garantiza que el nodo "anterior"
            if(anterior.getContent().getTotalGanado() < nodoSiguiente.getContent().getTotalGanado()){
                Apuesta temporal = anterior.getContent();
                anterior.resetContent(nodoSiguiente.getContent());
                nodoSiguiente.resetContent(temporal);
            }

            anterior = nodoSiguiente;
        }
    }
}//Servicio crítico 4.2 -> O(n) uwu xD xD
```

› Complejidad Objetivo -> O(n²)

› Complejidad alcanzada: **O(n²)**

› Descripción de la complejidad

O(n) =	1 declaracion + 1 declaracion + (n asignaciones + n comparaciones + n incrementos)
	+ n(1 comparacion + n-1 comparaciones + 1 obtencion + 1 reseteo + 1 reseteo + 1 asignacion)
O(n) =	2 + (3n) + n(4 + n)
O(n) =	2 + 30 + 4n + n²
O(n) =	**O(n²)**