

# TECNICAS DE PROGRAMACION AVANZADAS

**DIGITAL BLOCK 3** 



VICTOR PEREZ PEREZ 21/02/2021

# Contenido

Ejercicio 1	2
Código	2
Pseudocódigo	3
Complejidad	3
Ejercicio 2	4
Código	
Pseudocódigo	6
Complejidad	7
Ejercicio 3	
Código	8
Pseudocódigo	10
Compleiidad	11

# Ejercicio 1

#### Código

```
package DigitalBlock3;
public class Ejercicio1 {
         public static void main(String[] args) {
                    * Creamos un array <u>llamado</u> 'miLista'
                   int[] miLista = {0,1,2,3,4,5,6,7,8,9};
                    * <u>Llamamos</u> a <u>la funcion</u> pares, a <u>la cual le pasamos la lista</u>, el <u>inicio(</u>0), el final(array-1)
                    * y <u>una</u> variable <u>para almacenar las numeros</u> pares (<u>iniciada en</u> 0)
                   System.out.println("Tenemos" + pares(miLista, 0, miLista.length-1, 0));
         }
         public static int pares(int[]array, int inicio, int fin, int par) {
                   /** Caso base **/
                   if (inicio == fin) {
                             * Si la posicion en la que estamos es par, incrementamos el valor par (de 0 a
1)
                             * En el caso contrario devolveriamos 0, ya que el elemento seria impar.
                             if(array[inicio] %2 == 0) {
                                      par++;
                             return par;
                   /** Parte recursiva **/
                   } else {
                             * <u>Dividimos</u> el array <u>en dos mitades iguales</u>
                             int mitad = (fin+inicio)/2;
                             * Creamos dos variables, 'x' e 'y'. Estas seran las encargadas de almacenar el
valor de 'par',
                             * <u>para posteriormete hacer la suma de los numeros</u> pares <u>que tenemos en</u>
cada una de las partes de la lista.
                             int x = pares(array, inicio, mitad, par);
                             int y = pares(array, mitad+1, fin, par);
                              * Sumamos las cantidades de numeros pares de cada una de las partes en la
que hemos dividido la lista
                             return x + y;
                   }
         }
}
```

## Pseudocódigo

```
Función pares(array[1....n]: enteros; inicio, fin, pares: enteros)

Si 'inicio' es igual que 'fin'

Si array[inicio]%2 es igual a 0

par++

devolver par

Si no

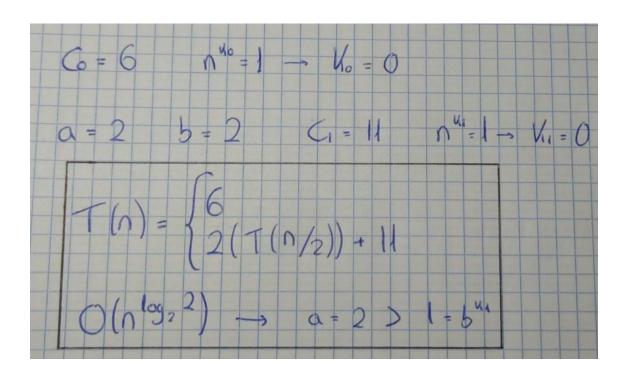
mitad = (fin + inicio)/2

x = pares(array[1....n]: enteros; inicio, mitad, pares: enteros)

y = pares(array[1....n]: enteros; mitad+1, fin, pares: enteros)

devolver x + y
```

## Complejidad



# Ejercicio 2

#### Código

```
package DigitalBlock3;
import java.util.Scanner;
public class Ejercicio2 {
        public static void main(String[] args) {
                 /** Creamos variable Scanner para escribir **/
                 Scanner sc = new Scanner(System.in);
                 /** Creamos tres arrays, para poder demostrar que el emtodo funciona
correctamente,
                          dos listas iguales y otra diferente a las anteriores
                  **/
                 int[] miLista1 = {1,2,3,4,5,6,7,8,9,10};
                 int[] miLista2 = {1,2,3,4,5,6,7,8,9,10};
                 int[] miLista3 = {1,3,5,7,9,2,4,6,8,10};
                 System. out. println("Lista1 --> 1,2,3,4,5,6,7,8,9,10");
                 System. out. println("Lista2 --> 1,2,3,4,5,6,7,8,9,10");
                 System. out. println("Lista3 --> 1,3,5,7,9,2,4,6,8,10");
                 System.out.println("");
                 System.out.println("RECURSIVIDAD DIVISION (DyV) (Lista1 - Lista2)");
                 if(comparacionesDyV(0, miLista1.length-1, miLista1, miLista2) == true)
System.out.println("Listas ordenadas");
                 else System.out.println("Listas desordenadas");
                 System.out.println("");
                 System.out.println("RECURSIVIDAD DIVISION (DyV) (Lista1 - Lista3)");
                 if(comparacionesDyV(0, miLista1.length-1, miLista1, miLista3) == true)
System.out.println("Listas ordenadas");
                 else System.out.println("Listas desordenadas");
                 System.out.println("");
                 System.out.println("RECURSIVIDAD SUSTRACCION (Lista1 - Lista2)");
                 if(comparacionesSus(miLista1.length-1, miLista1, miLista2) == true)
System.out.println("Listas ordenadas");
                 else System.out.println("Listas desordenadas");
                 System.out.println("");
                 System.out.println("RECURSIVIDAD SUSTRACCION (Lista1 - Lista3)");
                 if(comparacionesSus(miLista1.length-1, miLista1, miLista3) == true)
System.out.println("Listas ordenadas");
                 else System.out.println("Listas desordenadas");
                 System.out.println("");
        }
```

```
public static boolean comparacionesSus(int fin, int miLista1[], int miLista2[]) {
                  * <u>Si una vez recorrida la lista por completo</u>, no <u>hemos obtenido ningun</u> false,
devolveremos true.
                  * Si algunos de los elementos no coinciden, devolvemos false
                  * Mientras que la lista no se haya recorrida entera y los elementos comparados hasta
el momento coincidan,
                  * <u>llamaremos</u> a <u>la funcion restandole uno al</u> fin <u>de la lista</u>
                 if(fin < 0) {
                           return true;
                 }else if(miLista1[fin] != miLista2[fin]) {
                           return false;
                 }else{
                           return comparacionesSus(fin-1, miLista1, miLista2);
                 }
        }
        public static boolean comparacionesDyV(int inicio, int fin, int miLista1[], int miLista2[]) {
                 /** Caso base **/
                 if (inicio == fin) {
                           if(miLista1[inicio] == miLista2[inicio]) {
                                    return true; /** Si los elementos son iguales devolvemos true **/
                           }else {
                                    return false; /** Si los elementos son diferentes devolvemos false
**/
                          }
                           /** Parte recursiva **/
                 }else {
                             Creamos una variable para dividir la lista en dos partes iguales
                            * Creamos dos variable booleanas para saber si los elementos de las listas
coinciden (true) o no (false)
                           int mitad = (inicio+fin)/2;
                           boolean x = comparacionesDyV(inicio, mitad, miLista1, miLista2);
                           boolean y = comparacionesDyV(mitad+1, fin, miLista1, miLista2);
                          /**
                           * Si en la primera mitad encontramos alguno diferente, devolveriamos false
                           if(x != true) {
                                    return false;
                           }
                          /**
                           * Si en la segunda mitad encontramos alguno diferente, devolveriamos false
                           if(y != true) {
                                    return false;
                           }
```

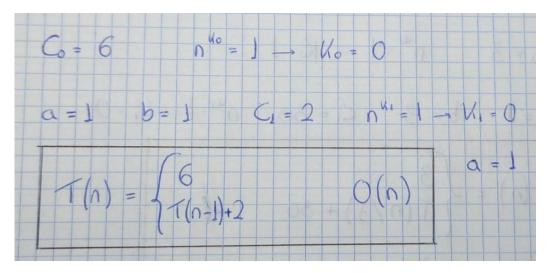
```
return true; /** <u>Si todo esta bien (listas iguales)</u>, <u>devolvemos</u> true **/
}
}
```

#### Pseudocódigo

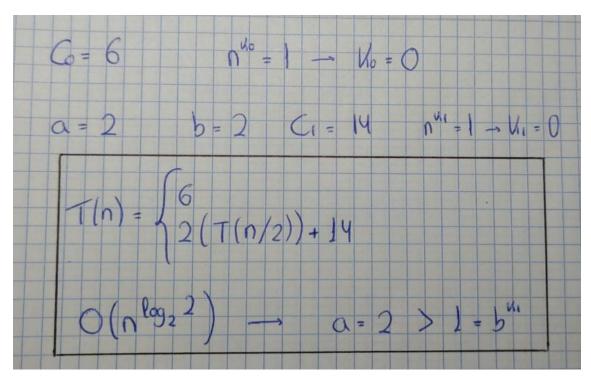
```
Funcion comparacionesSus (fin: entero; lista1[1....n], lista2[1....n]: enteros)
    Si 'fin' es menor que 0
             devolver true
    Si no si lista1[fin] es diferente de lista2[fin]
             devolver false
    Si no
             devolver Funcion comparacionesSus (fin: entero; lista1[1....n], lista2[1....n]:
enteros) --> (fin = fin - 1)
Funcion comparacionesDyV(inicio, fin: enteros; lista1[1....n], lista2[1....n]: enteros)
    Si 'inicio' es igual a 'fin'
             Si lista1[inicio] es igual que lista2[inicio]
                     devolver true
             Si no
                     devolver false
    Si no
             mitad = (inicio + fin) / 2
             x = Funcion comparacionesDyV(inicio, fin: enteros; lista1[1....n], lista2[1....n]:
enteros) --> (fin = mitad)
             y = Funcion comparacionesDyV(inicio, fin: enteros; lista1[1....n], lista2[1....n]:
enteros) --> (inicio = mitad + 1)
             Si x no es true
                     devolver flase
             Si y no es true
                     devolver false
             devolver true
```

# Complejidad

#### Sustracción



#### División



# Ejercicio 3

#### Código

```
package DigitalBlock3;
import java.util.Scanner;
public class Ejercicio3 {
        public static void main(String[] args) {
                 /** Creamos variable Scanner para escribir **/
                 Scanner sc = new Scanner(System.in);
                 /** Creamos un array de enteros **/
                 int[] miarray= {1,2,3,4,5,6,7,8,9};
                  /** Pedimos al usuario que nos indique el numero que desea buscar **/
                 System.out.println("Introduzca un numero para buscar: ");
                 int x = sc.nextInt();
                  /** <u>Llamamos</u> a <u>la funcion comprobar</u>, <u>para</u> saber <u>si</u> el <u>numero se encuentra en</u> el
array o no **/
                 System. out. println("El numero se encuentra en la posicion: " + (comprobar(miarray, x)-
1) + " del array");
        }
         * Funcion para saber si el numero que introducimos se encuentra en el array o no
         * Si no se encuentra, devolvemos 0
         * Si se encuentra, llamamos a la funcion para que nos indique la posicion en la que se
encuentra
        public static int comprobar(int array[], int x) {
                 if(x > array.length) {
                           return 0;
                 }else {
                           return buster(array,x, 0, array.length);
                 }
        }
         * Metodo que busca el numero indicado en el array y nos devuelve la posicion en la que se
encuentra
        public static int buster (int array[],int x, int inicio, int fin) {
                 /** Caso base **/
                 if(inicio==fin) {
                           return array[inicio];
                  /** Parte recursiva **/
                 }else {
```

```
* Creamos una variable rango, para saber en cuantas partes tenemos que
dividir nuestra lista (evitando errores)
                            int rango = fin - inicio;
                            int tercio1 = inicio + (rango/3);
                            int tercio2 = inicio + (2*rango/3);
                             /**
                             * <u>De esta manera nuestra lista se queda dividida en tres partes donde</u>
diferenciamos:
                                      el inicio (0/3)
                                      el tercio1
                                                         (1/3)
                                      el tercio2
                                                         (2/3)
                                      el final (3/3)
                             * Buscamos si nuestro numero coincide con alguna de las partes anteriores.
                             * Si esto pasara, devolvemos ese valor
                             * [<u>inicio</u>, ...., tercio1, ...., tercio2, ...., fin]
                            if(x==tercio1) {
                                      return tercio1;
                            }else if (x==tercio2) {
                                      return tercio2;
                            }else if(x==fin) {
                                      return fin;
                            }else if (x==inicio) {
                                      return inicio;
                             /**
                             * <u>Si nuestro numero se encuentra entre</u> el <u>inicio</u> y el primer <u>tercio</u>, <u>llamamos</u>
a <u>la funcion pasandole la lista correspondiente</u>
                            else if(inicio < x && x < tercio1) {
                                      return buster(array, x, inicio, inicio+tercio1-1);
                            * Si nuestro numero se encuentra entre el primer tercio y el segundo tercio,
<u>llamamos</u> a <u>la funcion pasandole la lista correspondiente</u>
                            }else if(x > tercio1 && tercio2 > x) {
                                      return buster(array, x, inicio+tercio1+1, fin-tercio1-1);
                             * Si nuestro numero se encuentra entre el segundo tercio y el final, llamamos
a <u>la funcion pasandole la lista correspondiente</u>
                            }else {
                                      return buster(array, x, fin-tercio1+1, fin);
                            }
                   }
         }
   }
```

#### Pseudocódigo

```
Funcion comprobar (array[1....n]: entero; x: entero)
    Si 'x' mayor que 'n'
             Devolver 0
    Si no
             Devolver Funcion buster(array[1....n]: entero; x, inicio, fin: entero)
Funcion buster(array[1....n]: entero; x, inicio, fin: entero)
    Si 'inicio' es igual que 'fin'
             devolver array[inicio]
    Si no
             rango = fin - inicio
             tercio1 = inicio + (rango/3)
             tercio2 = inicio + (2*rango/3)
             Si 'x' igual que 'tercio1'
                      devolver tercio1
             Si no si 'x' igual que 'tercio2'
                      devolver tercio2
             Si no si 'x' igual que 'inicio'
                      devolver inicio
             Si no si 'x' igual que 'fin'
                      devolver fin
             Si no si 'inicio' menor que 'x' y 'x' menor que 'tercio1'
                      devolver Funcion buster(array[1....n]: entero; x, inicio, fin: entero) -->
(fin = inicio + tercio1 - 1)
             Si no si 'x' mayor que 'tercio1' y 'x' menor que 'tercio2'
                      devolver Funcion buster(array[1....n]: entero; x, inicio, fin: entero) -->
(inicio = inicio + tercio1 + 1)(fin = fin - tercio1 - 1)
             Si no
                      devolver Funcion buster(array[1....n]: entero; x, inicio, fin: entero) -->
(inicio = fin - tercio1 + 1)
```

# Complejidad

