Características básicas de Kotlin

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información

Plan

Introducción

¿Qué es Kotlin?

- Lenguaje de programación de propósito general.
- Con tipos estáticos y con inferencia de tipo.
- Multiplataforma: Android, JVM, iOS, macOS, Windows, Linux, JavaScript (Kotlin/JS), WebAssembly, LLVM, entre otros.
- Multiparadigma: procedimientos, orientado a objetos y funcional.
- Kotlin está diseñado para trabajar con Java y la JVM.
- Tiene una sintaxis concisa.
- Kotlin compila principalmente para la JVM, pero también compila a JavaScript y código nativo.
- Diseñado por JetBrains.
- Website: `https://kotlinlang.org/`.

Configurando Kotlin

- Se recomienda usar el sistema operación GNU/Linux.
- Se usará el compilador que tiene como objetivo la JVM (Kotlin/JVM).
- Se recomienda trabajar con un editor (Emacs, Vi, VSCode, etc) y la línea de comandos, en lugar de un IDE (IntelliJ IDEA, Eclipse, etc)
- La versión de Kotlin a usar es 1.8 o superior.
- Se debe instalar el Java SDK versión 11 o superior.

Un primer programa en Kotlin

```
1 fun main() {
2   println("Hola Mundo!")
3 }
```

Listado 1: Hola Mundo en Hola.kt.

Compilando y ejecutando un primer programa en Kotlin

```
>kotlinc Hola.kt -include-runtime -d Hola.jar
```

Figura 1: Compilando el programa Hola.kt

```
>java -jar Hola.jar
```

Figura 2: Ejecutando el archivo Hola.jar

```
>Hola Mundo!
```

Figura 3: Salida obtenida de ejecutar Hola.jar

Compilando y ejecutando programas en Kotlin

```
>kotlinc <archivo .kt> -include-runtime -d <archivo .jar>
```

Figura 4: Compilando un programa Kontlin

```
>java -jar <archivo .jar>
```

Figura 5: Ejecutando un archivo .jar con la JVM

Esquema de compilación y ejecución de Kotlin

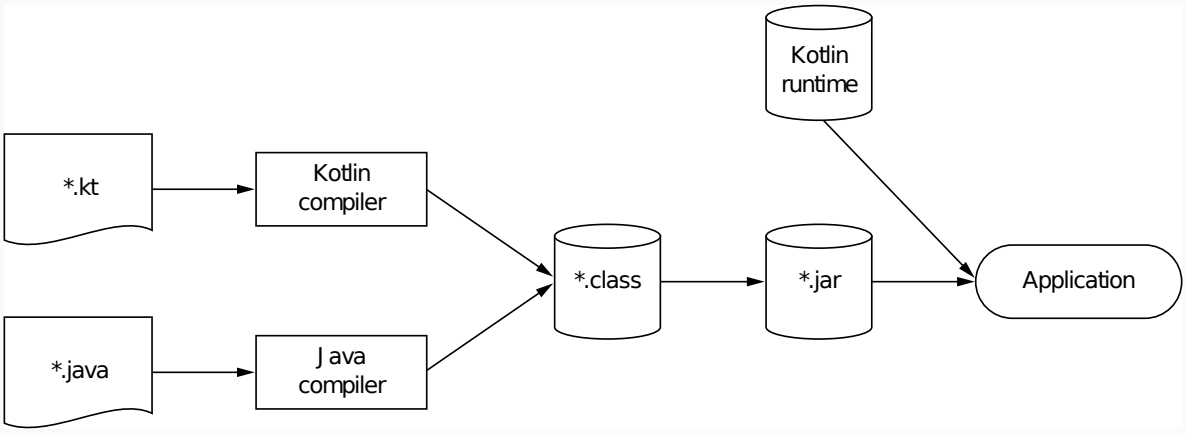

Figura 6: Esquema del proceso de construcción de una aplicación en Kotlin. Fuente [2]

Otra forma de compilar y ejecutar en Kotlin

```
>kotlinc Hola.kt
```

Figura 7: Compilando el programa Hola.kt para generar archivos .class

```
>kotlin HolaKt
```

Figura 8: Ejecutando el archivo HolaKt.class

Algunas herramientas para la compilación de código Kotlin

- Ant
- Gradle
- **Make**

Características básicas

Comentarios de código

```
1    // Ejemplo de comentario, esto no se ejecuta
```

Listado 2: Comentario de una línea

```
1    /*
2     * Ejemplo de comentario, esto no se ejecuta
3     */
```

Listado 3: Comentario de varias líneas

```
1    /**
2     * Ejemplo de comentario, esto no se ejecuta
3     */
```

Listado 4: Comentario de documentación

Operadores básicos y símbolos especiales

- **Operadores matemáticos:** $+$, -, *, $/$, %
- **Asignación:** $=$
- **Operadores de asignación aumentada:** $+=$, -=, *=, $/=$, %=
- **Operadores de incremento y decremento:** $++$, $--$
- **Operadores lógicos** $\wedge, \vee$ **y** $\neg$**:** &&, ||, !
- **Operadores de igualdad:** $==$, !=
- **Operadores de referencia:** $===$, !==
- **Operadores de comparación:** $<$, $>$, $<=$, $>=$
- **Aserción de que una operación es no nula:** !!
- **Operador Elvis que toma el valor de la derecha si el valor de la izquierda es nulo:** ?:
- **Separa un nombre de una declaración de tipo:** :
- **Declara que el tipo acepta valores nulos:** ?
- **Separa varias declaraciones en la misma línea:** ;

Constantes

```kotlin
val nombre: String = "Maria"
val edad: Int = 22
val peso = 54.2
val pi: Double = 3.14159
edad = 23 // error: val cannot be reassigned
```

Listado 5: Ejemplos de asignación de constantes que no cambian su valor una vez que son inicializadas.

```kotlin
const val miConstante: Int = 10
```

Listado 6: Ejemplo de una constante en tiempo de compilación. Se declara fuera de una función y debe ser inicializada.

Variables

```kotlin
var creditosAprobados: Int = 10
creditosAprobados = 15
var indice = 4.45
```

Listado 7: Ejemplo de asignación de variables.

Operadores sobre una variable

```
1    var cont = 1
2    cont += 1 // El valor de cont es 2
3    cont -= 2 // El valor de cont es 0
4    println(cont++) // El valor de cont es 1 y se nuestra 1
5    println(++cont) // El valor de cont es 2 y se muestra 2
6    cont *= 3 // El valor de cont es 2*3 = 6
7    cont /= 2 // El valor de cont es 6/2 = 3
8    cont %= 2 // El valor de cont es 3%2 = 1
```

Listado 8: Ejemplo de operadores de incremento, decremento y otros sobre una misma variable.

Tipos de datos básicos de Kotlin

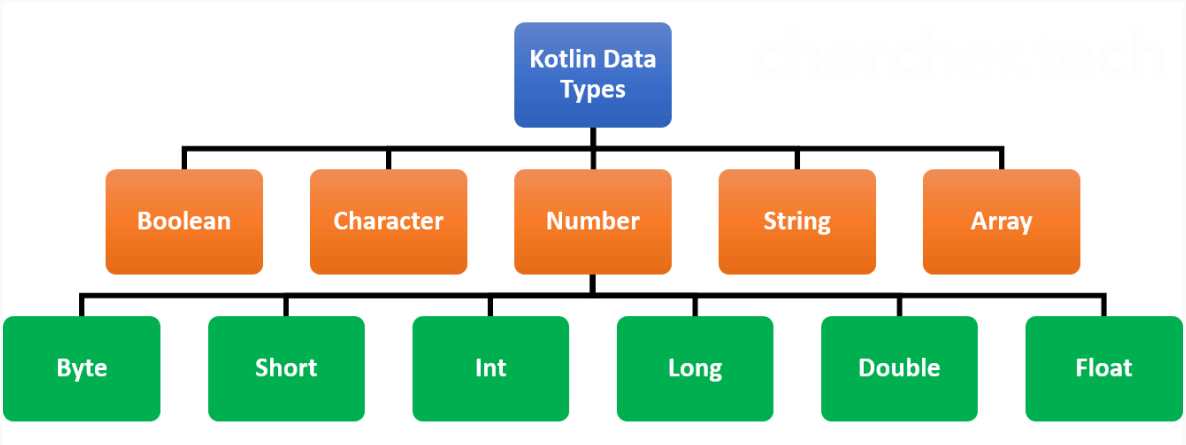

Figura 9: Jerarquía de los tipos de datos básicos de Kotlin. Fuente [1]

Tipo de datos Strings

```
val carrera = "Computacion"
val saludo = "Hola Mundo\n"
val multiple = """
Tengo varias
lineas
"""
```

Listado 9: Varias formas de inicializar un String.

```
var nombre:String = "Pepe"
println(nombre.uppercase()) // Imprime un nuevo String PEPE
println(nombre) // El nombre no se modifica, imprime Pepe
nombre[0] = 'T' // Error, no se puede modificar el String
```

Listado 10: Los Strings son inmutables.

```
val nombre = "Juana"
val apellido = "De Arco"
val nombreCompleto = nombre + " " + apellido
println(nombreCompleto)  // Juana De Arco
```

Listado 11: Ejemplo de concatenación de Strings.

String templates

```
val base = 3
val altura = 2.5
println("Area = ${base}*${altura} = ${base*altura}") //Area
    = 3*2.5 = 7.5
```

Listado 12: Ejemplo de un string que contienen *string templates*.

```
val zorro:String = "Zorro"
println("La ${zorro[0]} del $zorro") //La Z del Zorro
println("La $zorro.length es ${zorro.length}") // La Zorro.
    length es 5
```

Listado 13: Ejemplo de los corchetes en *string templates* y como un string es una cadena de caracters.

Tipos de datos para números enteros

Tipo de dato	Bits	Valor mínimo	Valor máximo
Byte	8	-128	127
Short	16	-32768	32767
Int	32	-2147483648	2147483647
Long	64	-9223372036854775808	9223372036854775807

```
1  var bMin:Byte = Byte.MIN_VALUE
2  var bMax:Byte = Byte.MAX_VALUE
3  println("Byte: el menor ${bMin} y el mayor ${bMax}")
4  var sMin:Short = Short.MIN_VALUE
5  var sMax:Short = Short.MAX_VALUE
6  println("Short: el menor ${sMin} y el mayor ${sMax}")
7  var iMin:Int = Int.MIN_VALUE
8  var iMax:Int = Int.MAX_VALUE
9  println("Int: el menor ${iMin} y el mayor ${iMax}")
10 var lMin:Long = Long.MIN_VALUE
11 var lMax:Long = Long.MAX_VALUE
12 println("Long: el menor ${lMin} y el mayor ${lMax}")
```

Listado 14: Ejemplo de variables con los límites de los tipos enteros.

Tipos de datos para números reales

Tipo de dato	Bits	Valor mínimo	Valor máximo
Float	32	1.40129846432481707e-45	3.40282346638528860e+38
Double	64	4.94065645841246544e-324	1.79769313486231570e+308

```
1  var fMin:Float = Float.MIN_VALUE
2  var fMax:Float = Float.MAX_VALUE
3  println("El menor valor tipo float: ${fMin}")
4  println("El mayor valor tipo float: ${fMax}")
5
6  var dMin:Double = Double.MIN_VALUE
7  var dMax:Double = Double.MAX_VALUE
8  println("El menor valor tipo double: ${dMin}")
9  println("El mayor valor tipo double: ${dMax}")
```

Listado 15: Ejemplo de variables con los límites de los tipos reales.

Tipo de dato Number

```
1   val real: Double = 1.2
2   val entero: Int = 10
3   val flotante : Float = 2.3f
4   val corto: Short = 100
5   val miByte: Byte = 12
6   var numero: Number = 0
7   numero = real
8   numero = entero
9   numero = flotante
10  numero = corto
11  numero = miByte // 12
12  println(numero)
13
```

Listado 16: Una variable de tipo Number puede tomar cualquier tipo numérico

Tipo de datos booleanos

Tipo de dato	Bits	Rango
Boolean	1	**true** o **false**

```
1   var r1:Boolean = 1 > 2
2   println("Es 1 > 2?: ${r1}")  // Es 1 > 2?: false
```

Listado 17: Ejemplo de una variable booleana.

Tipo de datos de caracteres

Tipo de dato	Bits	Valor mínimo	Valor máximo
Char	8	-128	127

```
1    var zeta:Char = 'Z'
2    println("Le marcaron la ${zeta} del Zorro") // Le
     marcaron la Z del Zorro
```

Listado 18: Ejemplo de una variable caracter.

Conversión de tipos

```
1  var entero: Integer = 10
2  var real: Double = 10.4
3  entero = real // error: type mismatch: inferred type is
       Double but Int was expected
4  real = entero // error: type mismatch: inferred type is Int
       but Double was expected
5  real = entero.toDouble() // real es 10.0
6  var entrada = "30"
7  var edad: Int = entrada.toInt() // edad es 30
8  entrada = real.toString() // entrada es "10.0"
```

Listado 19: Ejemplo de conversión de tipos entre variables y constantes.

Pares y Triples

```
1       val punto2D: Pair<Int, Int> = Pair(2, 3)
2       val punto3D: Triple<Int, Int, Int> = Triple(1, 2, 3)
3       val peso = 56.8
4       val altura = 1.75
5       val nombre = "Karla"
6       val datosPaciente = Triple(nombre, altura, peso)
7       val coordenada = Pair(3.34, 6.78)
```

Listado 20: Ejemplos de pares y triples.

Tipo de dato Any

Se tiene que Any es un tipo de dato del cual descienden todos los demás tipos datos. Una variable de tipo Any puede tomar cualquier tipo de datos que no sea nulo.

```
1       val punto2D: Pair<Int, Int> = Pair(2, 3)
2       val punto3D: Triple<Int, Int, Int> = Triple(1, 2, 3)
3       val peso: Double = 56.8
4       val nombre: String = "Julia"
5       val edad: Int = 10
6       var objeto: Any = ""
7       objeto = punto2D
8       objeto = punto3D
9       objeto = peso
10      objeto = nombre
11      objeto = edad
12      println(objeto) // 10
```

Listado 21: Ejemplos del tipo Any

Tipo de dato Unit

Un tipo Unit se usa para indicar que una función no retorna un elemento de algún tipo.

```
1 fun dosAlCuadrado(): Unit {
2         val resultado = 2 * 2
3         println(resultado)
4 }
```

Listado 22: La función `dosAlCuadrado` no retorna ningún elemento

Control de Flujo

La expresión `if`

```
1    if (90 < 10) {
2      println("Es mayor 90 que 10")
3    }
```

Listado 23: Ejemplo de la expresión `if`

La expresión `if - else`

```
1    val edadPedro = 19
2    val edadKris = 23
3    if (edadPedro  < edadKris) {
4      println("Kris es mayor que Pedro")
5    } else {
6      println("Pedro es mayor que Kris")
7    }
```

Listado 24: Ejemplo de la expresión `if - else`

La expresión `if - else if - else`

```
1  val edadPedro = 19
2  val descripcion = if ( edadPedro <= 10) {
3    "Ni~o"
4  } else if (edadPedro  > 10 && edadPedro < 20) {
5    "Adolescente"
6  } else {
7    "Adulto"
8  }
9  println(descripcion) // Adolescente
```

Listado 25: Ejemplo de la expresión `if - else if - else`

Bucle `while`

```
1  var acumulador = 0
2  val incremento = 17
3
4  while (acumulador < 1000) {
5    acumulador += incremento
6  }
7  println(acumulador) // 1003
```

Listado 26: Ejemplo de la expresión `while`

Bucle `do - while`

```
1   var acumulador = 0
2   val incremento = 17
3
4   do {
5     acumulador += incremento
6   } while (acumulador < 1000)
7   println(acumulador) // 1003
```

Listado 27: Ejemplo de la expresión `do - while`

Saliendo de un bucle con `break`

```
1   var acumulador = 0
2   val incremento = 17
3
4   while(true) {
5     if (acumulador > 1000) {
6       break
7     }
8     acumulador += incremento
9   }
10  println(acumulador) // 1003
```

Listado 28: Ejemplo del comando `break`

Rangos

```
1 val rango10 = 0..10 // (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
2 println(rango10) // 0..10
```

Listado 29: Ejemplo del comando `..`

```
1 val rango9 = 0 until 10 // (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
2 println(rango9) // 0..9
```

Listado 30: Ejemplo del comando `until`

```
1 val decre = 10 downTo 0 // (10, 9, 8, 7 ,6, 5, 4, 3, 2, 1)
2 println(decre) // 10 downTo 0 step 1
```

Listado 31: Ejemplo del comando `downTo`

Bucle `for`

```
1   for (<CONSTANTE> in <RANGO>) {
2     <INSTRUCCIONES DEL BUCLE>
3 }
```

Listado 32: Estructura del bucle `for`

Ejemplos de bucle `for`

```
val n = 10; var acumulador = 0
for (i in 0..n) {
  acumulador += i
}
println(acumulador) // 55
```

Listado 33: Ejemplo de bucle `for` con incremento de 1

```
val n = 10; var acumulador = 0
for (i in 0..n step 5) {
  acumulador += i
}
println(acumulador) // 15
```

Listado 34: Ejemplo de bucle `for` con incremento de 5

```
val n = 10; var acumulador = 0
for (i in n downTo 0 step 2) {
  acumulador += i
}
println(acumulador) // 30
```

Listado 35: Ejemplo de bucle `for` con incremento de 2

Bucle `Repeat`

```
val n = 10
val incremento = 10
var acumulador = 0

repeat (n) {
  acumulador += incremento
}
println(acumulador) // 100
```

Listado 36: Ejemplo de bucle `repeat`

Salto de un bucle con `continue`

```
1  var acumulador = 0
2  val n = 100
3
4  for (i in 0..n) {
5    if (i % 2 == 0) {
6      continue
7    }
8    acumulador += i
9  }
10 println(acumulador) // 2500
```

Listado 37: Ejemplo del comando `continue`

Expresión `when`

```
1  var animal = "Perro"
2
3  when (animal) {
4    "Gato" -> println("Es un gato")
5    "Perro" -> println("Es un perro")
6    else -> println("No es ni perro ni gato")
7  }
```

Listado 38: Ejemplo de la expresión `when`

Funciones

Funciones básicas

```
1  import java.time.LocalDateTime
2
3  fun horaActual() {
4      val actual = LocalDateTime.now()
5      println("La hora actual es: $actual")
6  }
7
8  fun saludo(nombre: String, apellido: String) {
9      println("Hola $nombre $apellido")
10 }
11
12 fun main() {
13     val name: String = "Sasha"
14     val surname: String  = "Ramirez"
15     saludo(name, surname) // Hola Sasha Ramirez
16     horaActual() // La hora actual es: 2021-05-15T17
       :28:58.133
17 }
```

Listado 39: Ejemplo de dos procedimientos.

Funciones con retorno de valores

```kotlin
fun circuloAreaPerimetro(radio: Double): Pair<Double,Double>
{
    return Pair(2*Math.PI*radio, 2*Math.PI*radio)
}

fun areaRectangulo(base: Int, altura:Int): Int = base*altura

fun main() {
    val radio = 2.3
    val h = 2
    val b = 3
    val (area, perimetro) = circuloAreaPerimetro(radio)
    val rArea = areaRectangulo(b, h)
    println("Circunferencia area: $area")  // Circunferencia
     area: 14.451326206513047
    println("Circunferencia perimetro: $perimetro")  //
    Circunferencia perimetro: 14.451326206513047
    println("Area rectangulo: $rArea") // Area rectangulo: 6
}
```

Listado 40: Ejemplo de dos funciones.

Funciones como variables y argumentos

```kotlin
fun sumaEntero(a: Int, b: Int): Int {
    return a + b
}

fun aplicarMostrar(f :(Int, Int) -> Int, a: Int, b: Int){
    val r = f(a, b)
    println("Resultado: $r")
}

fun main() {
    var suma = ::sumaEntero
    val n = 5
    val m = 4
    println("Resultado: ${suma(n, m)}") // Resultado: 9
    aplicarMostrar(suma, n, m) // Resultado: 9
    aplicarMostrar(::sumaEntero, n, m) // Resultado: 9
}
```

Listado 41: La función `sumaEntero` se asigna como variable y se usa como argumento.

Valores nulos

Tipos nulos

```
1    var entrada :String? = null
2    println(entrada) // null
3    println("Indique su numbre:")
4    entrada = readLine()
5    println("Hola $entrada") // Hola Rosa
```

Listado 42: La variable entrada se inicializa como nula. Como su tipo de datos permite valores nulos entonces puede recibir data de la entrada estándar.

Operador aserción de no nulidad

```
1    var edadActual: Int? = 32
2    var edadDespuesDelCumple: Int = edadActual + 1  // error
     : operator call corresponds to a dot-qualified call '
     edadActual.plus(1)' which is not allowed on a nullable
     receiver 'edadActual'.
3    var edadDespuesDelCumple: Int = edadActual!! + 1
4    println(edadDespuesDelCumple) // 33
5    edadActual = null
6    println("Edad despues de 10 cumplea~nos es: ${edadActual
     !! + 10}") // java.lang.NullPointerException
```

Listado 43: Ejemplo de una asignación usando el operador de aserción de no nulidad.

Ejecución segura

```
1    var entrada :String? = null
2    println("Indique su numbre:")
3    entrada = readLine()
4    println("Hola $entrada") // Hola Rosa
5    println("El nombre tiene ${entrada.length} letras") //
     error: only safe (?.) or non-null asserted (!!.) calls
     are allowed on a nullable receiver of type String?
6    println("El nombre tiene ${entrada?.length} letras") //
     El nombre tiene 4 letras
```

Listado 44: La variable entrada se inicializa como nula. Como su tipo de datos permite valores nulos entonces puede recibir data de la entrada estándar.

El operador Elvis

```
1    var pruebaConNulo: Int? = 26
2    var pruebaSinNulo: Int = pruebaConNulo ?: 10
3    println(pruebaSinNulo) // 26
4    pruebaConNulo = null
5    pruebaSinNulo = pruebaConNulo ?: 10
6    println(pruebaSinNulo) // 10
```

Listado 45: Ejemplo del uso del operador Elvis.

Arreglos

Arreglos con diversos tipos con `arrayOf`

```
1     val impares = arrayOf(1, 3, 5, 7)
2     for (num in impares) {
3       println(num)
4     }
```

Listado 46: Ejemplo de arrayOf con elementos de un mismo tipo entero.

```
1     val variadito = arrayOf(1, true, "Carlota", 3.5)
2     for (v in variadito) {
3       println(v)
4     }
```

Listado 47: Ejemplo de arrayOf con elementos de diferentes tipos de datos.

Arreglos de cualquier tipo con `Array`

```
1 val enteros = Array<Int>(3, {0}) // 0, 0, 0
```

Listado 48: Ejemplo de un arreglo de enteros.

```
1 val numeros = Array<Number>(5, {1.1}) // 1,1, 1.1, 1.1, 1.1,
    1.1
2 val numeros = Array<Number>(5, {it -> it+1}) // 1, 2, 3, 4,
  5
```

Listado 49: Ejemplo de un arreglo de Números.

Arreglos de tipos primitivos

```
1  val aFloat = floatArrayOf(1.0f, 2.0f ,3.0f)
2  val aDouble = doubleArrayOf(1.0, 2.0, 3.0)
3  val aBoolean = booleanArrayOf(true, false, true)
4  val aInteger = intArrayOf(1, 2, 3)
5
6  val nFloat = FloatArray(3) // 0.0, 0.0, 0.0
7  val nDouble = DoubleArray(3) // 0, 0, 0
8  val nBoolean = BooleanArray(3) // false, false, false
9  val nEntero =  IntArray(3) // 0, 0, 0
10
11 val aFloat: FloatArray = floatArrayOf(1.0f, 2.0f ,3.0f)
12 val aDouble: DoubleArray = doubleArrayOf(1.0, 2.0, 3.0)
13 val aBoolean: BooleanArray = booleanArrayOf(true, false,
       true)
14 val aInteger: IntArray = intArrayOf(1, 2, 3)
```

Listado 50: Ejemplo de arreglos primitivos.

Ejemplo 1: problema de búsqueda

Problema de búsqueda

Planteamiento del problema de búsqueda

Dado un arreglo $T[1..n]$ ordenado de forma ascendente y un elemento x, se desea la posición k de x, en caso de que x se encuentre en el arreglo, o en que posición k debería ser insertado en caso de no pertenecer al arreglo.

Ejemplo: Dado el arreglo:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
-2	-1	0	2	3	5	6	8	9

- Si $x = 3$ entonces $k = 5$
- Si $x = 1$ entonces $k = 4$

Búsqueda lineal

```
1  fun busquedaLineal(secuencia: Array<Int>, x: Int): Int {
2    for ((index, value) in secuencia.withIndex()) {
3      if (value >= x) {
4        return index
5      }
6    }
7    return secuencia.size
8  }
```

Listado 51: Solución del problema de búsqueda con búsquueda lineal.

Búsqueda binaria

```
fun busquedaBinaria(secuencia: Array<Int>, x: Int): Int {
  if (secuencia[secuencia.size-1] < x) {
    return secuencia.size
  }
  return busquedaBinRec(secuencia, 0, secuencia.size-1, x)
}

fun busquedaBinRec(T: Array<Int>, i: Int, j: Int, x: Int):
    Int {
  if (i == j) {
    return i
  }
  val k = (i+j).div(2)
  if (x <= T[k]) {
    return busquedaBinRec(T, i, k, x)
  } else {
    return busquedaBinRec(T, k+1, j, x)
  }
}
```

Listado 52: Solución del problema de búsqueda con búsqueda binaria.

Prueba de la solución del problema de búsqueda

```
fun main(args: Array<String>) {
  val x = args[0].toInt()
  println("Elemento a buscar: $x")
  val et: Array<String> = args.sliceArray(1 until args.size)
  val sec: Array<Int> = et.map{it.toInt()}.toTypedArray()
  if (!estaEnOrdenAscendente(sec)) {
    println("Error, la sec no esta ordenada")
    return
  }
  var k = busquedaLineal(sec, x)
  println("Posicion busqueda lineal: $k")
  k = busquedaBinaria(sec, x)
  println("Posicion busqueda binaria: $k")
}
```

Listado 53: Programa de prueba del problema de búsqueda.

Ejemplo 2: problema de ordenamiento

Problema de ordenamiento

Planteamiento del problema de ordenamiento

Dada una secuencia de n números $a_1, a_2, a_3, \ldots a_n$, se quiere obtener una permutación $a'_1, a'_2, a'_3, \ldots a'_n$ de la entrada, tal que la secuencia resultante esté ordenada $a'_1 \leq a'_2 \leq a'_3 \leq \cdots \leq a'_n$

Algoritmo de Bubblesort

```
fun bubbleSort(A: Array<Int>) {
  for (i in 0 until A.size) {
    for (j in (A.size - 1) downTo (i+1)) {
      if (A[j] < A[j-1]) {
        var tmp = A[j]
        A[j] = A[j-1]
        A[j-1] = tmp
      }
    }
  }
}
```

Listado 54: Código del algoritmo de Bubblesort. Archivo: Ordenammiento.kt

Prueba del programa de ordenamiento

```
fun main(args: Array<String>) {
  val n: Int = args[0].toInt()
  val sec: Array<Int> = Array(n, {0})
  for (i in 0 until n) {
    sec[i] =  (0..n).random()
  }
  bubbleSort(sec)
  if (!estaEnOrdenAscendente(sec)) {
    println("Error, la secuencia no esta ordenada")
    return
  }
  println("Ordenamiento exitoso!")
}
```

Listado 55: Programa de prueba de Bubblesort. Archivo: Main.kt

Archivo Makefile para la compilación del programa de ordenamiento

```
KC= kotlinc
KFLAGS= -include-runtime
PROG= PruebaOrdenamiento.jar
SRC=  Ordenammiento.kt Main.kt

all:
  $(KC) $(SRC) $(KFLAGS) -d $(PROG)

.PHONY : clean

clean :
        rm -rf $(PROG)
```

Listado 56: Archivo Makefile del programa de ordenamiento

Archivo para la ejecución del problema de ordenamiento

```
#!/usr/bin/bash

java -jar PruebaOrdenamiento.jar $*
```

Listado 57: Archivo Bash para ejecutar el programa de ordenamiento. Archivo: runPruebaOrdenamiento.sh

Compilando y ejecutando el programa de ordenamiento

```
>make
```

Figura 10: Compilando el programa de ordenamiento

```
>./runPruebaOrdenamiento.sh <numero de elementos>
```

Figura 11: Ejecutando el programa de ordenamiento

```
>./runPruebaOrdenamiento.sh 10000
```

Figura 12: Ejemplo de la ejecución del programa de ordenamiento

Referencias

[1] Chercher Tech.
Data types in kotlin, 2021.
Accedido: 13.05.2021.

[2] D. Jemerov and S. Isakova.
Kotlin in Action.
Manning Publications Co, 2017.