Практическая №6

1.

```
        fun main () {
    val numbers = intArrayOf(10,20,30,40,50)
    for (number in numbers) {
        println(numbers)
    }
    numbers.forEach { number ->
        println(number)}

}
```

2.

```
fun main () {
    val numbers = intArrayOf(1,2,3,4,5)
    var sum = 0
    for (number in numbers) {
        sum += number
    }
    println("Сумма элементов массива:$sum")
    val sum2 = numbers.sum()
    println("Сумма элементов массива (sum()):$sum2")
}
```

3.

```
import kotlin.random.Random

fun main() {
    val numbers = List(10) { Random.nextInt(1, 101) }.toIntArray()
    println("Массив: ${numbers.contentToString()}")

    val max_value = numbers.maxOrNull() ?: 0 // ?: 0 обрабатывает случай
пустого массива
    val min_value = numbers.minOrNull() ?: 0 // ?: 0 обрабатывает случай
пустого массива

    println("Максимальное значение: $max_value")
    println("Минимальное значение: $min_value")
}
```

4.

```
import kotlin.random.Random

fun main() {
    val numbers = List(10) { Random.nextInt(1, 101) }.toIntArray()
    println("Неотсортированный массив: ${numbers.contentToString()}")

    numbers.sort() // встроенная функция сортировки

    println("Отсортированный массив: ${numbers.contentToString()}")
}
```

5.

```
fun main() {
    val myArray = arrayOf(1, 2, 2, 3, 4, 4, 5, 1)

    // Способ 1: использование toSet() - наиболее эффективный
    val uniqueElements = myArray.toSet().toList()
    println("Уникальные элементы (способ 1): $uniqueElements")
```

```
    // Способ 2: итерация (менее эффективный)
    val uniqueElements2 = mutableListOf<Int>()
    myArray.forEach {
        if (!uniqueElements2.contains(it)) {
            uniqueElements2.add(it)
        }
    }
    println("Уникальные элементы (способ 2): $uniqueElements2")
}
```

6.

```
fun splitEvenOdd(numbers: IntArray): Pair<IntArray, IntArray> {
    val evenNumbers = numbers.filter { it % 2 == 0 }.toIntArray()
    val oddNumbers = numbers.filter { it % 2 != 0 }.toIntArray()
    return Pair(evenNumbers, oddNumbers)
}

fun main() {
    val numbers = intArrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    val (even, odd) = splitEvenOdd(numbers)
    println("Четные числа: ${even.contentToString()}")
    println("Нечетные числа: ${odd.contentToString()}")
}
```

7.

```
fun reverseArray(numbers: IntArray): IntArray {
    return numbers.reversedArray()
}

fun main() {
    val numbers = intArrayOf(1, 2, 3, 4, 5)
    val reversedNumbers = reverseArray(numbers)
    println("Исходный массив: ${numbers.contentToString()}")
    println("Реверсированный массив: ${reversedNumbers.contentToString()}")
}
```

8.

```
fun findElement(array: IntArray, element: Int): Int {
    return array.indexOf(element)
}

fun main() {
    val numbers = intArrayOf(1, 5, 2, 8, 5, 3)
    val index = findElement(numbers, 5)
    if (index != -1) {
        println("Элемент 5 найден под индексом: $index")
    } else {
        println("Элемент 5 не найден")
    }
}
```

9.

```
fun copyArray(originalArray: IntArray): IntArray {
    return originalArray.copyOf()
}

fun main() {
    val originalArray = intArrayOf(1, 2, 3, 4, 5)
    val newArray = copyArray(originalArray)

    println("Оригинальный массив: ${originalArray.contentToString()}")
    println("Скопированный массив: ${newArray.contentToString()}")
```

```
    //Изменение скопированного массива не влияет на оригинальный
    newArray[0] = 10
    println("Оригинальный массив после изменения копии:
${originalArray.contentToString()}")
    println("Скопированный массив после изменения:
${newArray.contentToString()}")
}
```

10.

```
fun sumEvenNumbers(numbers: IntArray): Int {
    return numbers.filter { it % 2 == 0 }.sumOf { it }
}

fun main() {
    val numbers = intArrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    val sum = sumEvenNumbers(numbers)
    println("Сумма четных чисел: $sum")
}
```

11.

```
fun findIntersection(array1: IntArray, array2: IntArray): Set<Int> {
    return array1.toSet().intersect(array2.toSet())
}

fun main() {
    val array1 = intArrayOf(1, 2, 3, 4, 5)
    val array2 = intArrayOf(3, 5, 6, 7, 8)
    val intersection = findIntersection(array1, array2)
    println("Пересечение массивов: $intersection") // Вывод: [3, 5]
}
```

12.

```
fun swapElements(array: IntArray, index1: Int, index2: Int) {
    if (index1 in array.indices && index2 in array.indices) {
        val temp = array[index1]
        array[index1] = array[index2]
        array[index2] = temp
    } else {
        println("Один или оба индекса находятся за пределами массива.")
    }
}

fun main() {
    val array = intArrayOf(1, 2, 3, 4, 5)
    swapElements(array, 0, 2) // Меняем местами элементы с индексами 0 и 2
    println(array.contentToString()) // Вывод: [3, 2, 1, 4, 5]
}
```

13.

```
import kotlin.random.Random

fun main() {
    val randomNumbers = IntArray(20) { Random.nextInt(1, 101) } // 101,
потому что nextInt(100) дает числа от 1 до 99
    println(randomNumbers.contentToString())
}
```

14.

```
fun findDivisibleByThree(numbers: IntArray): List<Int> {
    return numbers.filter { it % 3 == 0 }
}
```

```
fun main() {
    val numbers = intArrayOf(1, 3, 6, 9, 12, 15, 2, 4, 5, 7, 8, 10)
    val divisibleByThree = findDivisibleByThree(numbers)
    println("Числа, делящиеся на 3: $divisibleByThree")
}
```

15.

```
fun isPalindrome(array: IntArray): Boolean {
    val reversedArray = array.reversedArray()
    return array.contentEquals(reversedArray)
}

fun main() {
    val array1 = intArrayOf(1, 2, 3, 2, 1)
    val array2 = intArrayOf(1, 2, 3, 4, 5)
    println("Array1 is palindrome: ${isPalindrome(array1)}") // true
    println("Array2 is palindrome: ${isPalindrome(array2)}") // false
}
```

16.

```
fun concatenateArrays(array1: IntArray, array2: IntArray): IntArray {
    return array1 + array2
}

fun main() {
    val array1 = intArrayOf(1, 2, 3)
    val array2 = intArrayOf(4, 5, 6)
    val concatenatedArray = concatenateArrays(array1, array2)
    println(concatenatedArray.contentToString()) // Вывод: [1, 2, 3, 4, 5, 6]
}
```

17.

```
fun sumAndProduct(array: IntArray): Pair<Int, Int> {
    var sum = 0
    var product = 1
    for (element in array) {
        sum += element
        product *= element
    }
    return Pair(sum, product)
}

fun main() {
    val array = intArrayOf(1, 2, 3, 4, 5)
    val (sum, product) = sumAndProduct(array)
    println("Сумма: $sum, Произведение: $product") // Вывод: Сумма: 15,
Произведение: 120
}
```

18.

```
fun groupNumbers(numbers: IntArray, groupSize: Int): List<List<Int>> {
    return numbers.toList().chunked(groupSize)
}

fun main() {
    val numbers = intArrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
    val groupedNumbers = groupNumbers(numbers, 5)
    println(groupedNumbers) // Вывод: [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10],
[11, 12, 13]]
}
```

19.

```
fun mergeSortedArraysInefficient(arr1: IntArray, arr2: IntArray): IntArray {
    return (arr1 + arr2).sortedArray()
}

fun main() {
    val arr1 = intArrayOf(1, 3, 5, 7)
    val arr2 = intArrayOf(2, 4, 6, 8)
    val mergedArray = mergeSortedArraysInefficient(arr1, arr2)
    println(mergedArray.contentToString()) // Вывод: [1, 2, 3, 4, 5, 6, 7, 8]
}
```

20.

```
fun arithmeticProgression(firstElement: Int, difference: Int,
numberOfElements: Int): IntArray {
    val array = IntArray(numberOfElements)
    var currentElement = firstElement
    for (i in 0 until numberOfElements) {
        array[i] = currentElement
        currentElement += difference
    }
    return array
}

fun main() {
    val progression = arithmeticProgression(2, 3, 5)
    println(progression.contentToString()) // Вывод: [2, 5, 8, 11, 14]
}
```

21.

```
fun removeElementFunctional(array: IntArray, elementToRemove: Int): IntArray
{
    return array.filterNot { it == elementToRemove }.toIntArray()
}

fun main() {
    val array = intArrayOf(1, 2, 3, 4, 2, 5)
    val newArray = removeElementFunctional(array, 2)
    println(newArray.contentToString()) // Вывод: [1, 3, 4, 5]
}
```

22.

```
fun findSecondMax(arr: IntArray): Int? {
    if (arr.size < 2) return null // Недостаточно элементов для второго максимума
    val sortedArr = arr.sortedArrayDescending()
    return sortedArr[1]
}
fun main() {
    val array = intArrayOf(1, 5, 2, 8, 3, 9, 4, 7, 6)
    val secondMax = findSecondMax(array)
```

```
    println("Второй по величине элемент: $secondMax") // Вывод: Второй по величине элемент: 8
}
```

23.

```
fun mergeArraysPlus(vararg arrays: IntArray): IntArray {
    var result = intArrayOf()
    for (array in arrays) {
        result += array
    }
    return result
}
fun main() {
    val array1 = intArrayOf(1, 2, 3)
    val array2 = intArrayOf(4, 5, 6)
    val array3 = intArrayOf(7, 8, 9)

    val mergedArray = mergeArraysPlus(array1, array2, array3)
    println("Объединенный массив: ${mergedArray.contentToString()}") // Вывод: Объединенный массив: [1, 2, 3, 4, 5, 6, 7, 8, 9]
}
```

24.

```
fun transposeMatrix(matrix: Array<IntArray>): Array<IntArray> {
    val rows = matrix.size
    val cols = matrix[0].size // Предполагаем, что все строки имеют одинаковую длину
    val transposedMatrix = Array(cols) { IntArray(rows) }

    for (i in 0 until rows) {
        for (j in 0 until cols) {
            transposedMatrix[j][i] = matrix[i][j]
```

```
        }
    }
    return transposedMatrix
}
fun main() {
    val matrix = arrayOf(
        intArrayOf(1, 2, 3),
        intArrayOf(4, 5, 6),
        intArrayOf(7, 8, 9)
    )
    val transposed = transposeMatrix(matrix)
    println("Исходная матрица:")
    printMatrix(matrix)
    println("\nТранспонированная матрица:")
    printMatrix(transposed)
}
fun printMatrix(matrix: Array<IntArray>) {
    for (row in matrix) {
        println(row.contentToString())
    }
}
```

25.

```
fun linearSearchAny(array: IntArray, target: Int): Boolean {
    return array.any { it == target }
}

fun main() {
    val array = intArrayOf(1, 5, 2, 8, 3, 9, 4, 7, 6)
    val target1 = 8
```

```
    val target2 = 10


    println("Элемент $target1 найден: ${linearSearchAny(array, target1)}") // Вывод: true
    println("Элемент $target2 найден: ${linearSearchAny(array, target2)}") // Вывод: false
}
fun main() {
    val kotlin = "🙂"
    println(kotlin)
}
```

26.

```
val numbersForAverage = intArrayOf(1, 2, 3, 4, 5)
    val average = numbersForAverage.average()
    println("Среднее арифметическое: $average")

println("▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨")
```

27.

```
fun maxSequence(array29: IntArray): Pair<Int, Int> {
        var maxCount = 1
        var currentCount = 1


        for (i in 1 until array29.size) {
            if (array29[i] == array29[i - 1]) {
                currentCount++
            } else {
                maxCount = maxOf(maxCount, currentCount)
                currentCount = 1
            }
```

```
        }
        maxCount = maxOf(maxCount, currentCount)

        return maxCount to array29.size // Возвращаем максимальную длину последовательности и размер массива для примера
    }
```

28.

```
// Для ввода массива можно использовать Scanner или другой метод в зависимости от среды выполнения.
// Здесь приведен пример с использованием статического массива.
    val userInputArray = intArrayOf(1, 2, 3, 4, 5)
    println("Введенный массив: ${userInputArray.joinToString(", ")}")


println("▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
▨▨▨▨▨▨▨▨▨▨▨▨")
```

29.

```
fun findMedian(array30: IntArray): Double {
        val sortedArray = array30.sorted()
        val size = sortedArray.size

        return if (size % 2 == 0) {
            (sortedArray[size / 2 - 1] + sortedArray[size / 2]) / 2.0
        } else {
            sortedArray[size / 2].toDouble()
        }
    }

// Пример использования
    val medianArray = intArrayOf(3, 1, 4, 2, 5)
```

```
    val medianValue = findMedian(medianArray)
    println("Медиана: $medianValue")

println("▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨")
```

30.

```
val groupSize = 10
    val totalNumbers = 100
    val randomNumbers = IntArray(totalNumbers) { (1..100).random() }

    for (i in randomNumbers.indices step groupSize) {
        val group = randomNumbers.slice(i until minOf(i + groupSize, randomNumbers.size))
        println("Группа ${i / groupSize + 1}: ${group.joinToString(", ")}")
    }
}
```