

**Institute of Engineering and Technology (IET)**

**JK Lakshmipat University**

**Kotlin -Assignment**  
*Set 2*

**Submitted BY**

Siddhi Nyati (2022btech101)

**FACULTY GUIDE**

Dr. Pranab Roy

**March 2025**

**Question 1**

import java.util.\*  
  
fun insertionSort(arr: IntArray) {  
 for (i in 1 *until* arr.size) {  
 val key = arr[i]  
 var j = i - 1  
 while (j >= 0 && arr[j] > key) {  
 arr[j + 1] = arr[j]  
 j--  
 }  
 arr[j + 1] = key  
 }  
}  
  
fun mergeArraysDescending(arr1: IntArray, arr2: IntArray): IntArray {  
 val merged = IntArray(arr1.size + arr2.size)  
 var i = arr1.*lastIndex* var j = arr2.*lastIndex* var k = 0  
  
 while (i >= 0 && j >= 0) {  
 merged[k++] = if (arr1[i] > arr2[j]) arr1[i--] else arr2[j--]  
 }  
  
 while (i >= 0) merged[k++] = arr1[i--]  
 while (j >= 0) merged[k++] = arr2[j--]  
  
 return merged  
}  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 *print*("Enter size of first array: ")  
 val n1 = scanner.nextInt()  
 val arr1 = IntArray(n1)  
 *println*("Enter elements of first array:")  
 for (i in 0 *until* n1) arr1[i] = scanner.nextInt()  
  
 *print*("Enter size of second array: ")  
 val n2 = scanner.nextInt()  
 val arr2 = IntArray(n2)  
 *println*("Enter elements of second array:")  
 for (i in 0 *until* n2) arr2[i] = scanner.nextInt()  
  
 *insertionSort*(arr1)  
 *insertionSort*(arr2)  
  
 val mergedArray = *mergeArraysDescending*(arr1, arr2)  
  
 *println*("Merged Array in Descending Order: ${mergedArray.*joinToString*(" ")}")  
}

**Output**Enter size of first array: 4

Enter elements of first array:

5 1 8 3

Enter size of second array: 3

Enter elements of second array:

7 2 6

Merged Array in Descending Order: 8 7 6 5 3 2 1

**Question 2**

import java.util.\*  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 *print*("Enter the number of elements: ")  
 val n = scanner.nextInt()  
  
 val numbers = IntArray(n)  
 *println*("Enter numbers between 0-100:")  
 for (i in 0 *until* n) numbers[i] = scanner.nextInt()  
  
 val frequency = IntArray(10)  
  
 for (num in numbers) {  
 if (num in 1..100) {  
 val index = (num - 1) / 10  
 frequency[index]++  
 }  
 }  
  
 for (i in 0 *until* 10) {  
 val lower = i \* 10 + 1  
 val upper = lower + 9  
 *println*("$lower-$upper: ${frequency[i]}")  
 }  
}

**Output**

Enter the number of elements: 12

Enter numbers between 0-100:

5 17 29 35 42 50 66 71 88 90 99 100

1-10: 1

11-20: 1

21-30: 1

31-40: 1

41-50: 2

51-60: 0

61-70: 1

71-80: 1

81-90: 2

91-100: 2

**Question 3**

import java.util.\*  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 fun inputMatrix(rows: Int, cols: Int): Array<IntArray> {  
 val matrix = Array(rows) **{** IntArray(cols) **}** for (i in 0 *until* rows) for (j in 0 *until* cols) matrix[i][j] = scanner.nextInt()  
 return matrix  
 }  
  
 fun isSymmetric(matrix: Array<IntArray>): Boolean {  
 for (i in matrix.*indices*) for (j in matrix[i].*indices*) if (matrix[i][j] != matrix[j][i]) return false  
 return true  
 }  
  
 fun isSkewSymmetric(matrix: Array<IntArray>): Boolean {  
 for (i in matrix.*indices*) for (j in matrix[i].*indices*) if (matrix[i][j] != -matrix[j][i]) return false  
 return true  
 }  
  
 fun addMatrices(a: Array<IntArray>, b: Array<IntArray>): Array<IntArray>? {  
 if (a.size != b.size || a[0].size != b[0].size) return null  
 return Array(a.size) **{** i **->** IntArray(a[0].size) **{** j **->** a[i][j] + b[i][j] **} }** }  
  
 fun multiplyMatrices(a: Array<IntArray>, b: Array<IntArray>): Array<IntArray>? {  
 if (a[0].size != b.size) return null  
 val result = Array(a.size) **{** IntArray(b[0].size) **}** for (i in result.*indices*) for (j in result[i].*indices*) for (k in a[i].*indices*) result[i][j] += a[i][k] \* b[k][j]  
 return result  
 }  
  
 fun transpose(matrix: Array<IntArray>): Array<IntArray> {  
 return Array(matrix[0].size) **{** i **->** IntArray(matrix.size) **{** j **->** matrix[j][i] **} }** }  
  
 fun createCustomMatrix(a: Array<IntArray>, b: Array<IntArray>): Array<IntArray> {  
 val n = a.size  
 val newMatrix = Array(n) **{** IntArray(n) **}** for (i in 0 *until* n) for (j in 0 *until* n) newMatrix[i][j] = when {  
 i > j -> a[i][j]  
 i < j -> b[i][j]  
 else -> 0  
 }  
 return newMatrix  
 }  
  
 *print*("Enter matrix dimensions (rows cols): ")  
 val rows = scanner.nextInt()  
 val cols = scanner.nextInt()  
  
 *println*("Enter first matrix:")  
 val matrixA = inputMatrix(rows, cols)  
  
 *println*("Enter second matrix:")  
 val matrixB = inputMatrix(rows, cols)  
  
 *println*("Symmetric: ${isSymmetric(matrixA)}")  
 *println*("Skew Symmetric: ${isSkewSymmetric(matrixA)}")  
  
 addMatrices(matrixA, matrixB)?.*let* **{** *println*("Addition:\n" + **it**.*joinToString*("\n") **{** row **->** row.*joinToString*(" ") **}**) **}** ?: *println*("Addition not possible")  
  
 multiplyMatrices(matrixA, matrixB)?.*let* **{** *println*("Multiplication:\n" + **it**.*joinToString*("\n") **{** row **->** row.*joinToString*(" ") **}**) **}** ?: *println*("Multiplication not possible")  
  
 val transposed = transpose(matrixA)  
 *println*("Transpose of first matrix:\n" + transposed.*joinToString*("\n") **{** row **->** row.*joinToString*(" ") **}**)  
  
 if (rows == cols) {  
 val customMatrix = createCustomMatrix(matrixA, matrixB)  
 *println*("Custom Matrix:\n" + customMatrix.*joinToString*("\n") **{** row **->** row.*joinToString*(" ") **}**)  
 } else *println*("Custom Matrix requires square matrices.")  
}

**Output:**

Enter matrix dimensions (rows cols): 3 3

Enter first matrix:

1 2 3

2 4 5

3 5 6

Enter second matrix:

7 8 9

8 10 11

9 11 12

Symmetric: true

Skew Symmetric: false

Addition:

8 10 12

10 14 16

12 16 18

Multiplication:

50 61 67

91 111 122

115 140 154

Transpose of first matrix:

1 2 3

2 4 5

3 5 6

Custom Matrix:

0 8 9

2 0 11

3 5 0

**Question 4**

import java.util.\*  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 fun inputSet(size: Int): Set<Int> {  
 val set = *mutableSetOf*<Int>()  
 while (set.size < size) set.add(scanner.nextInt())  
 return set  
 }  
  
 fun powerSet(set: Set<Int>): List<Set<Int>> {  
 val list = set.*toList*()  
 val result = *mutableListOf*<Set<Int>>()  
 val total = 1 shl list.size  
 for (i in 0 *until* total) result.add(list.*indices*.*filter* **{** i and (1 shl **it**) != 0 **}**.*map* **{** list[**it**] **}**.*toSet*())  
 return result  
 }  
  
 *print*("Enter size of set A: ")  
 val sizeA = scanner.nextInt()  
 *println*("Enter elements of set A:")  
 val setA = inputSet(sizeA)  
  
 *print*("Enter size of set B: ")  
 val sizeB = scanner.nextInt()  
 *println*("Enter elements of set B:")  
 val setB = inputSet(sizeB)  
  
 val unionSet = setA *union* setB  
 val intersectionSet = setA *intersect* setB  
 val differenceSet = setA *subtract* setB  
 val powerSetA = powerSet(setA)  
  
 *println*("Union: $unionSet")  
 *println*("Intersection: ${if (intersectionSet.isEmpty()) "Disjoint" else intersectionSet}")  
 *println*("Set A - Set B: $differenceSet")  
 *println*("Power Set of A: $powerSetA")  
}

**Output**

Enter size of set A: 4

Enter elements of set A:

1 2 3 4

Enter size of set B: 3

Enter elements of set B:

3 4 5

Union: [1, 2, 3, 4, 5]

Intersection: [3, 4]

Set A - Set B: [1, 2]

Power Set of A: [[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3], [4], [1, 4], [2, 4], [1, 2, 4], [3, 4], [1, 3, 4], [2, 3, 4], [1, 2, 3, 4]]

**Question 5**

fun findIncreasingSequences(arr: IntArray): List<List<Int>> {  
 val sequences = *mutableListOf*<List<Int>>()  
 var tempList = *mutableListOf*(arr[0])  
  
 for (i in 1 *until* arr.size) {  
 if (arr[i] > arr[i - 1]) {  
 tempList.add(arr[i])  
 } else {  
 if (tempList.size > 1) sequences.add(tempList.*toList*())  
 tempList = *mutableListOf*(arr[i])  
 }  
 }  
 if (tempList.size > 1) sequences.add(tempList.*toList*())  
 return sequences  
}  
  
fun findMaxNumbers(arr: IntArray): Pair<Int, Int> {  
 var max1 = Int.MIN\_VALUE  
 var max2 = Int.MIN\_VALUE  
 for (num in arr) {  
 if (num > max1) {  
 max2 = max1  
 max1 = num  
 } else if (num > max2 && num != max1) {  
 max2 = num  
 }  
 }  
 return Pair(max1, max2)  
}  
  
fun rearrangeOddEven(arr: IntArray): IntArray {  
 val odds = arr.*filter* **{ it** % 2 != 0 **}**.*toMutableList*()  
 val evens = arr.*filter* **{ it** % 2 == 0 **}**.*toMutableList*()  
 val result = *mutableListOf*<Int>()  
  
 while (odds.*isNotEmpty*() || evens.*isNotEmpty*()) {  
 if (evens.*isNotEmpty*()) result.add(evens.removeAt(0))  
 if (odds.*isNotEmpty*()) result.add(odds.removeAt(0))  
 }  
 return result.*toIntArray*()  
}  
  
fun main() {  
 val arr = *intArrayOf*(1, 3, 5, 4, 7, 10, 2, 3, 5, 8, 12)  
 *println*("Increasing Sequences: ${*findIncreasingSequences*(arr)}")  
  
 val (max1, max2) = *findMaxNumbers*(arr)  
 *println*("First Maximum: $max1, Second Maximum: $max2")  
  
 *println*("Rearranged Array: ${*rearrangeOddEven*(arr).*joinToString*(", ")}")  
}

**Output**

Increasing Sequences: [[1, 3, 5], [4, 7, 10], [2, 3, 5, 8, 12]]

First Maximum: 12, Second Maximum: 10

Rearranged Array: 4, 1, 10, 3, 2, 5, 8, 7, 12, 3, 5