# Scala Interview Handbook — Batch 6

Generated: 2025-09-13 02:45:31Z (UTC)

## Scala Theory & Cheatsheet

```
SCALA THEORY & CHEATSHEET (Quick Ref)
-------------------------------------
- Syntax: vals vs vars; methods; case classes; pattern matching.
- Collections: immutable List/Vector/Map/Set; mutable variants.
- Functional: map/flatMap/filter/fold; Options/Eithers; for-comprehensions.
- Performance: prefer immutable + structural sharing; use arrays for tight loops.
- Testing: ScalaTest AnyFunSuite; assertions; property-based (optional).
```

## 051. TransposeMatrix

## Problem Overview & Strategy

TransposeMatrix — Detailed Explanation Approach: Bounds pointers (top/bottom/left/right) for spiral; transpose+reverse for rotation; classic i-k-j loops for multiplication. Correctness: Each layer/element is moved/visited exactly once. Complexity: O(mn) time.

## Scala Solution

```
package problems

object Problem051TransposeMatrix {
  def transpose(M:Array[Array[Int]]): Array[Array[Int]] = {
    val m=M.length; val n=M(0).length
    val T = Array.ofDim[Int](n,m)
    var i=0; while (i<m) { var j=0; while (j<n) { T(j)(i)=M(i)(j); j+=1 }; i+=1 }
    T
  }
}
```

## ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem051TransposeMatrix

class Problem051TransposeMatrixSpec extends AnyFunSuite {
  test("transpose") {
    val M = Array(Array(1,2,3), Array(4,5,6))
    val T = Problem051TransposeMatrix.transpose(M)
    assert(T(0).toList==List(1,4) && T(1).toList==List(2,5) && T(2).toList==List(3,6))
  }
}
```

## 052. SpiralMatrix

## Problem Overview & Strategy

SpiralMatrix — Detailed Explanation Approach: Bounds pointers (top/bottom/left/right) for spiral; transpose+reverse for rotation; classic i-k-j loops for multiplication. Correctness: Each layer/element is moved/visited exactly once. Complexity: O(mn) time.

## Scala Solution

```
package problems
import scala.collection.mutable.ArrayBuffer

object Problem052SpiralMatrix {
  def spiral(m:Array[Array[Int]]): List[Int] = {
    val res = ArrayBuffer[Int]()
    if (m.isEmpty) return res.toList
    var top=0; var bot=m.length-1; var left=0; var right=m(0).length-1
    while (top<=bot && left<=right) {
      var j=left; while(j<=right){ res += m(top)(j); j+=1 }; top+=1
      var i=top; while(i<=bot){ res += m(i)(right); i+=1 }; right-=1
      if (top<=bot) { j=right; while(j>=left){ res += m(bot)(j); j-=1 }; bot-=1 }
      if (left<=right) { i=bot; while(i>=top){ res += m(i)(left); i-=1 }; left+=1 }
    }
    res.toList
  }
}
```

## ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem052SpiralMatrix

class Problem052SpiralMatrixSpec extends AnyFunSuite {
  test("spiral 3x3") {
    val m = Array(Array(1,2,3),Array(4,5,6),Array(7,8,9))
    assert(Problem052SpiralMatrix.spiral(m) == List(1,2,3,6,9,8,7,4,5))
  }
}
```

## 053. FindMissingNumber

## Problem Overview & Strategy

FindMissingNumber — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

## Scala Solution

```
package problems

object Problem053FindMissingNumber {
  def missing(a:Array[Int]): Int = {
    val n=a.length
    val expected = n*(n+1)/2
    expected - a.sum
  }
}
```

## ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem053FindMissingNumber

class Problem053FindMissingNumberSpec extends AnyFunSuite {
  test("missing") {
    assert(Problem053FindMissingNumber.missing(Array(0,1,3)) === 2)
  }
}
```

## 054. FindDuplicate

### Problem Overview & Strategy

FindDuplicate — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

### Scala Solution

```scala
package problems

object Problem054FindDuplicate {
  def findDuplicate(a:Array[Int]): Int = {
    var slow=a(0); var fast=a(0)
    do { slow=a(slow); fast=a(a(fast)) } while (slow!=fast)
    slow=a(0)
    while (slow!=fast) { slow=a(slow); fast=a(fast) }
    slow
  }
}
```

### ScalaTest

```scala
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem054FindDuplicate

class Problem054FindDuplicateSpec extends AnyFunSuite {
  test("dup") {
    assert(Problem054FindDuplicate.findDuplicate(Array(1,3,4,2,2)) === 2)
  }
}
```

## 055. MedianTwoSortedArraysSmall

### Problem Overview & Strategy

MedianTwoSortedArraysSmall — Detailed Explanation Approach: Use appropriate sorting—Merge/Quick/Heap—or selection (min-heap/quickselect). Correctness: Follows standard algorithms with proven invariants. Complexity: typically O(n log n) sort; quickselect average O(n).

### Scala Solution

```
package problems

object Problem055MedianTwoSortedArraysSmall {
  def median(A:Array[Int], B:Array[Int]): Double = {
    val n = A.length + B.length
    var i=0; var j=0; var prev=0; var cur=0
    var k=0; while (k<=n/2) {
      prev=cur
      if (i<A.length && (j>=B.length || A(i) <= B(j))) { cur=A(i); i+=1 }
      else { cur=B(j); j+=1 }
      k+=1
    }
    if (n%2==1) cur.toDouble else (prev + cur) / 2.0
  }
}
```

### ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem055MedianTwoSortedArraysSmall

class Problem055MedianTwoSortedArraysSmallSpec extends AnyFunSuite {
  test("median") {
    assert(math.abs(Problem055MedianTwoSortedArraysSmall.median(Array(1,3), Array(2)) - 2.0) < 1e-9)
    assert(math.abs(Problem055MedianTwoSortedArraysSmall.median(Array(1,2), Array(3,4)) - 2.5) < 1e-9)
  }
}
```

## 056. SearchInsertPosition

### Problem Overview & Strategy

SearchInsertPosition — Detailed Explanation Approach: Classic binary search over a sorted array (or search-space). Correctness: Midpoint halving preserves invariant that target lies in [lo,hi]. Complexity: O(log n) time, O(1) space.

### Scala Solution

```scala
package problems

object Problem056SearchInsertPosition {
  def searchInsert(a:Array[Int], target:Int): Int = {
    var lo=0; var hi=a.length-1
    while (lo<=hi) {
      val mid=(lo+hi)>>>1
      if (a(mid)==target) return mid
      else if (a(mid)<target) lo=mid+1
      else hi=mid-1
    }
    lo
  }
}
```

### ScalaTest

```scala
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem056SearchInsertPosition

class Problem056SearchInsertPositionSpec extends AnyFunSuite {
  test("search insert") {
    assert(Problem056SearchInsertPosition.searchInsert(Array(1,2,4,5),3) === 2)
  }
}
```

# 057. IntervalMerge

## Problem Overview & Strategy

IntervalMerge — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

## Scala Solution

```
package problems

object Problem057IntervalMerge {
  def merge(intervals:Array[Array[Int]]): Array[Array[Int]] = {
    val arr = intervals.sortBy(_(0))
    val res = scala.collection.mutable.ArrayBuffer[Array[Int]]()
    for (in <- arr) {
      if (res.isEmpty || res.last(1) < in(0)) res += in.clone
      else res.last(1) = math.max(res.last(1), in(1))
    }
    res.toArray
  }
}
```

## ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem057IntervalMerge

class Problem057IntervalMergeSpec extends AnyFunSuite {
  test("merge intervals") {
    val r = Problem057IntervalMerge.merge(Array(Array(1,3),Array(2,6),Array(8,10),Array(15,18)))
    assert(r.length == 3)
  }
}
```

# 058. BestTimeToBuySellStock

## Problem Overview & Strategy

BestTimeToBuySellStock — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

## Scala Solution

```
package problems

object Problem058BestTimeToBuySellStock {
  def maxProfit(prices:Array[Int]): Int = {
    var minP = prices(0); var best = 0
    for (p <- prices) { if (p<minP) minP=p; best = math.max(best, p-minP) }
    best
  }
}
```

## ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem058BestTimeToBuySellStock

class Problem058BestTimeToBuySellStockSpec extends AnyFunSuite {
  test("stock") {
    assert(Problem058BestTimeToBuySellStock.maxProfit(Array(7,1,5,3,6,4)) === 5)
  }
}
```

## 059. ProductOfArrayExceptSelf

### Problem Overview & Strategy

ProductOfArrayExceptSelf — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

### Scala Solution

```
package problems

object Problem059ProductOfArrayExceptSelf {
  def productExceptSelf(a:Array[Int]): Array[Int] = {
    val n=a.length
    val res = Array.fill(n)(1)
    var pref=1
    var i=0; while (i<n) { res(i)=pref; pref*=a(i); i+=1 }
    var suf=1
    i=n-1; while (i>=0) { res(i)*=suf; suf*=a(i); i-=1 }
    res
  }
}
```

### ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem059ProductOfArrayExceptSelf

class Problem059ProductOfArrayExceptSelfSpec extends AnyFunSuite {
  test("product except self") {
    assert(Problem059ProductOfArrayExceptSelf.productExceptSelf(Array(1,2,3,4)).toList == List(24,12,8,6))
  }
}
```

## 060. MaxProfitKTransactions

## Problem Overview & Strategy

MaxProfitKTransactions — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

## Scala Solution

```scala
package problems


object Problem060MaxProfitKTransactions {
  def maxProfit(k:Int, prices:Array[Int]): Int = {
    val n=prices.length; if (n==0 || k==0) return 0
    if (k >= n/2) {
      var prof=0; var i=1; while (i<n) { if (prices(i)>prices(i-1)) prof+=prices(i)-prices(i-1); i+=1 }; return prof
    }
    val buy = Array.fill(k+1)(Int.MinValue/4)
    val sell = Array.fill(k+1)(0)
    for (p <- prices) {
      var t=1; while (t<=k) {
        buy(t) = math.max(buy(t), sell(t-1) - p)
        sell(t) = math.max(sell(t), buy(t) + p)
        t+=1
      }
    }
    sell(k)
  }
}
```

## ScalaTest

```scala
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem060MaxProfitKTransactions

class Problem060MaxProfitKTransactionsSpec extends AnyFunSuite {
  test("k transactions dp") { assert(Problem060MaxProfitKTransactions.maxProfit(2, Array(3,2,6,5,0,3)) == 7) }
}
```