TOPIC 1: History and Features of Java

Easy MCQs (10)

1. Java was originally developed by:
   A) Microsoft
   B) Sun Microsystems
   C) IBM
   D) Oracle
   Answer: B) Sun Microsystems

2. Java's original name was:
   A) Oak
   B) Pine
   C) JavaBean
   D) J++
   Answer: A) Oak

3. Which feature allows Java programs to run on any platform?
   A) Multithreading
   B) Platform Independence
   C) Garbage Collection
   D) Exception Handling
   Answer: B) Platform Independence

4. Java is:
   A) Compiled only
   B) Interpreted only
   C) Both compiled and interpreted
   D) Neither
   Answer: C) Both compiled and interpreted

5. The 'Write Once, Run Anywhere' slogan refers to:
   A) Source code portability
   B) Bytecode portability via JVM
   C) IDE compatibility
   D) Cross-database support
   Answer: B) Bytecode portability via JVM

6. Which of the following is NOT a Java edition?
A) Java SE
B) Java EE
C) Java ME
D) Java WE
Answer: D) Java WE

7. Java was first released in:
A) 1991
B) 1995
C) 2000
D) 1989
Answer: B) 1995

8. Who is known as the “Father of Java”?
A) James Gosling
B) Bill Joy
C) Patrick Naughton
D) Mike Sheridan
Answer: A) James Gosling

9. Java does NOT support:
A) Pointers
B) Multithreading
C) Inheritance
D) Polymorphism
Answer: A) Pointers

10. Which company acquired Sun Microsystems in 2010?
A) Google
B) Microsoft
C) Oracle
D) IBM
Answer: C) Oracle

Medium MCQs (10)

1. Which Java feature helps prevent memory leaks automatically?
A) Finalizers

B) Garbage Collection
C) Destructors
D) Smart pointers
Answer: B) Garbage Collection

2. Java's security model primarily relies on:
A) Firewall integration
B) Bytecode verifier and SecurityManager
C) Encrypted source code
D) OS-level sandboxing only
Answer: B) Bytecode verifier and SecurityManager

3. The Java Community Process (JCP) is responsible for:
A) Selling Java licenses
B) Developing Java specifications via JSRs
C) Hosting Java conferences
D) Debugging JVM crashes
Answer: B) Developing Java specifications via JSRs

4. Which version introduced the `var` keyword for local variable type inference?
A) Java 8
B) Java 10
C) Java 11
D) Java 17
Answer: B) Java 10

5. Java's "Robustness" is achieved through:
A) Strong memory management and exception handling
B) High-speed compilation
C) Native code integration
D) GUI builders
Answer: A) Strong memory management and exception handling

6. Which of the following was a key motivation for Java's creation?
A) To replace C++ in system programming
B) To enable smart appliances and embedded systems
C) To compete with Python
D) To build Windows-only apps
Answer: B) To enable smart appliances and embedded systems

7. The Java Virtual Machine (JVM) is:
   A) Part of the operating system
   B) A specification implemented by vendors
   C) Only available on Windows
   D) Written exclusively in C++
   Answer: B) A specification implemented by vendors

8. Which Java feature supports distributed computing?
   A) RMI (Remote Method Invocation)
   B) JDBC
   C) AWT
   D) Servlets
   Answer: A) RMI (Remote Method Invocation)

9. Java's "Architecture Neutral" feature means:
   A) Code runs on any CPU architecture via JVM
   B) Programs don't use CPU registers
   C) Java ignores hardware
   D) Only cloud deployment is allowed
   Answer: A) Code runs on any CPU architecture via JVM

10. Which of these is a design goal of Java?
    A) Simplicity
    B) Backward compatibility with C
    C) Manual memory management
    D) Platform-specific optimization
    Answer: A) Simplicity

Hard MCQs (10)

1. In early Java (JDK 1.0), the GUI toolkit was:
   A) Swing
   B) JavaFX
   C) AWT
   D) SWT
   Answer: C) AWT

2. The "Green Project" at Sun Microsystems led to the creation of:

A) Solaris OS
B) Java
C) SPARC processors
D) Network File System
Answer: B) Java

3. Which Java version first included the HotSpot JVM?
A) JDK 1.1
B) JDK 1.2
C) JDK 1.3
D) JDK 1.4
Answer: C) JDK 1.3

4. The Java Native Interface (JNI) was introduced to:
A) Replace JVM
B) Allow Java to call native code (C/C++)
C) Improve garbage collection
D) Enable Python interoperability
Answer: B) Allow Java to call native code (C/C++)

5. Which statement about Java's evolution is FALSE?
A) Java 5 introduced generics
B) Java 8 introduced lambda expressions
C) Java 9 introduced modules (JPMS)
D) Java 1 removed applets
Answer: D) Java 1 removed applets *(Applets were deprecated much later)*

6. The "Duke" mascot was created by:
A) James Gosling
B) Joe Palrang
C) Kathy Sierra
D) Joshua Bloch
Answer: B) Joe Palrang

7. Java's initial target application domain was:
A) Web browsers
B) Set-top boxes and interactive TV
C) Mobile phones
D) Scientific computing

Answer: B) Set-top boxes and interactive TV

8. Which Java specification defines the language syntax and semantics?
A) JVM Spec
B) Java Language Specification (JLS)
C) JRE Guide
D) JDK Manual
Answer: B) Java Language Specification (JLS)

9. The transition from “Oak” to “Java” occurred because:
A) Oak was trademarked
B) Developers liked coffee
C) Sun wanted a cooler name
D) Legal issues with Oak Technology Inc.
Answer: D) Legal issues with Oak Technology Inc.

10. Which Java version marked the shift to a 6-month release cadence?
A) Java 8
B) Java 9
C) Java 10
D) Java 11
Answer: C) Java 10

Coding Problem

Easy Coding (10)

Q1. Write a Java program that prints “Welcome to Java!”

```
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

Q2. Write a program that prints your name using command-line arguments.

```
public class PrintName {
    public static void main(String[] args) {
        if (args.length > 0)
            System.out.println("Hello, " + args[0]);
```

```
        else
            System.out.println("Hello, Guest");
    }
}
```

Q3. Create a program that displays the Java version at runtime.

```
public class JavaVersion {
    public static void main(String[] args) {
        System.out.println("Java Version: " + System.getProperty("java.version"));
    }
}
```

Q4. Write a program that prints the current operating system.

```
public class OSInfo {
    public static void main(String[] args) {
        System.out.println("OS: " + System.getProperty("os.name"));
    }
}
```

Q5. Create a “Hello, World” program that compiles and runs on any platform.

```
// Same as Q1  demonstrates WORA principle
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Q6. Write a program that takes two numbers as CLI args and prints their sum.

```
public class AddCLI {
    public static void main(String[] args) {
        if (args.length >= 2) {
            int a = Integer.parseInt(args[0]);
            int b = Integer.parseInt(args[1]);
            System.out.println(a + b);
        }
    }
}
```

Q7. Print “Java is portable!” only if running on Windows.

```
public class ConditionalPrint {
    public static void main(String[] args) {
        String os = System.getProperty("os.name").toLowerCase();
        if (os.contains("win")) {
            System.out.println("Java is portable!");
        }
    }
}
```

Q8. Write a program that outputs the number of CPU cores available.

```
public class CPUInfo {
    public static void main(String[] args) {
        System.out.println("Cores: " + Runtime.getRuntime().availableProcessors());
    }
}
```

Q9. Create a program that prints “Compiled on: ” + compile time (use comment trick).

```
public class CompileTime {
    public static void main(String[] args) {
        System.out.println("Compiled on: " + "2024-06-15"); // manually updated
    }
}
```

Q10. Write a minimal Java program that does nothing but runs successfully.

```
public class DoNothing {
    public static void main(String[] args) {
        // Valid empty program
    }
}
```

Medium Coding (10)

Q11. Write a program that lists all system properties related to Java.

```
import java.util.Properties;
public class JavaProps {
    public static void main(String[] args) {
        Properties props = System.getProperties();
        props.stringPropertyNames().stream()
```

```
            .filter(key -> key.startsWith("java"))
            .sorted()
            .forEach(key -> System.out.println(key + " = " + props.getProperty(key)));
    }
}
```

Q12. Create a program that checks if garbage collection is supported.

```
public class GCCheck {
    public static void main(String[] args) {
        // GC is always supported in standard JVMs
        System.out.println("Garbage Collection: Supported");
        // Trigger GC (suggestion)
        System.gc();
    }
}
```

Q13. Write a program that demonstrates multithreading (print from two threads).

```
public class MultiThreadDemo {
    public static void main(String[] args) {
        Thread t1 = new Thread(() -> System.out.println("Thread 1"));
        Thread t2 = new Thread(() -> System.out.println("Thread 2"));
        t1.start();
        t2.start();
    }
}
```

Q14. Create a program that reads environment variables and prints JAVA_HOME.

```
public class EnvVar {
    public static void main(String[] args) {
        String javaHome = System.getenv("JAVA_HOME");
        System.out.println("JAVA_HOME = " + (javaHome != null ? javaHome : "Not set"));
    }
}
```

Q15. Write a program that shows memory usage (free/total/max).

```
public class MemoryUsage {
    public static void main(String[] args) {
        Runtime rt = Runtime.getRuntime();
        long total = rt.totalMemory();
```

```
        long free = rt.freeMemory();
        long max = rt.maxMemory();
        System.out.printf("Total: %d, Free: %d, Max: %d%n", total, free, max);
    }
}
```

Q16. Demonstrate exception handling with a division-by-zero error.

```
public class SafeDivide {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Handled: " + e.getMessage());
        }
    }
}
```

Q17. Write a program that loads a class dynamically using `Class.forName()`.

```
public class DynamicLoad {
    public static void main(String[] args) {
        try {
            Class<?> cls = Class.forName("java.lang.String");
            System.out.println("Loaded: " + cls.getName());
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Q18. Create a program that prints the classpath used to run it.

```
public class ShowClasspath {
    public static void main(String[] args) {
        System.out.println("Classpath: " + System.getProperty("java.class.path"));
    }
}
```

Q19. Write a program that uses `assert` to validate a condition (enable with -ea).

```
public class AssertionDemo {
    public static void main(String[] args) {
```

```
        int x = 5;
        assert x > 0 : "x must be positive";
        System.out.println("Assertion passed");
    }
}
```

Q20. Demonstrate automatic memory management by creating and discarding objects.

```
public class AutoMemory {
    public static void main(String[] args) {
        for (int i = 0; i < 1000; i++) {
            new Object(); // eligible for GC immediately
        }
        System.out.println("Objects created and discarded");
    }
}
```

Hard Coding (10)

Q21. Write a program that detects if it's running on HotSpot JVM.

```
public class JVMType {
    public static void main(String[] args) {
        String vmName = System.getProperty("java.vm.name");
        if (vmName != null && vmName.contains("HotSpot")) {
            System.out.println("Running on HotSpot JVM");
        } else {
            System.out.println("JVM: " + vmName);
        }
    }
}
```

Q22. Create a program that lists all garbage collector names available.

```
import java.lang.management.ManagementFactory;
import java.lang.management.GarbageCollectorMXBean;

public class GCDetector {
    public static void main(String[] args) {
        for (GarbageCollectorMXBean gc :
ManagementFactory.getGarbageCollectorMXBeans()) {
```

```
            System.out.println("GC: " + gc.getName());
        }
    }
}
```

Q23. Write a program that measures time taken to run a method using `System.nanoTime()`.

```
public class TimingDemo {
    public static void main(String[] args) {
        long start = System.nanoTime();
        busyWork();
        long end = System.nanoTime();
        System.out.println("Time: " + (end - start) + " ns");
    }

    static void busyWork() {
        for (int i = 0; i < 1_000_000; i++) {
            Math.sqrt(i);
        }
    }
}
```

Q24. Demonstrate platform independence by writing a file that works on all OS.

```
import java.io.File;
public class PortableFile {
    public static void main(String[] args) {
        // Uses platform-independent separator
        File f = new File(System.getProperty("user.home"), "test_java.txt");
        System.out.println("File path: " + f.getAbsolutePath());
    }
}
```

Q25. Write a program that checks if assertions are enabled at runtime.

```
public class AssertionCheck {
    public static void main(String[] args) {
        boolean enabled = false;
        assert enabled = true;
        if (enabled) {
```

```
            System.out.println("Assertions enabled");
        } else {
            System.out.println("Assertions disabled");
        }
    }
}
```

Q26. Create a program that uses reflection to print all methods of `String` class.

```
import java.lang.reflect.Method;
public class ReflectionDemo {
    public static void main(String[] args) {
        Method[] methods = String.class.getDeclaredMethods();
        for (Method m : methods) {
            System.out.println(m.getName());
        }
    }
}
```

Q27. Write a program that simulates a memory leak (for educational purposes).

```
import java.util.ArrayList;
import java.util.List;
public class SimulateLeak {
    public static void main(String[] args) throws InterruptedException {
        List<byte[]> leak = new ArrayList<>();
        while (true) {
            leak.add(new byte[1024 * 1024]); // 1MB each
            Thread.sleep(100);
        }
    }
}
```

Q28. Detect if the program is running in a container (e.g., Docker) via cgroup.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
public class ContainerDetect {
    public static void main(String[] args) {
        try {
            String cgroup = new String(Files.readAllBytes(Paths.get("/proc/1/cgroup")));
```

```
        if (cgroup.contains("docker") || cgroup.contains("kubepods")) {
            System.out.println("Running in container");
        } else {
            System.out.println("Bare metal or VM");
        }
    } catch (IOException e) {
        System.out.println("Not Linux or no access");
    }
  }
}
```

Q29. Write a program that uses JNI placeholder (commented, no native code).

```
public class JNIDemo {
    // Native method declaration (requires .c implementation and System.loadLibrary)
    // public native void nativeMethod();

    public static void main(String[] args) {
        System.out.println("JNI method declared but not implemented in this demo.");
    }
}
```

Q30. Create a self-documenting program that prints its own source file name.

```
public class SelfAware {
    public static void main(String[] args) {
        // Java doesn't provide __FILE__, but we can infer from stack trace
        String className = new
Object(){}.getClass().getEnclosingClass().getSimpleName();
        System.out.println("Source file likely: " + className + ".java");
    }
}
```

<u>TOPIC 2: Java Virtual Machine (JVM), JRE, and JDK</u>

Easy MCQs (10)

1. Which component is required to run a compiled Java program?
   A) JDK
   B) JRE
   C) IDE

D) Text Editor
Answer: B) JRE

2. JVM stands for:
A) Java Verified Machine
B) Java Virtual Machine
C) Java Variable Manager
D) Java Version Manager
Answer: B) Java Virtual Machine

3. Which of the following includes the compiler (`javac`)?
A) JRE
B) JVM
C) JDK
D) JVM + JRE
Answer: C) JDK

4. The JVM is:
A) Platform-dependent
B) Platform-independent
C) Written in Python
D) Part of the operating system kernel
Answer: A) Platform-dependent

5. Bytecode is executed by:
A) Operating System
B) JVM
C) CPU directly
D) Web browser
Answer: B) JVM

6. JRE = ?
A) JVM + Libraries
B) JDK + Compiler
C) JVM only
D) IDE + Debugger
Answer: A) JVM + Libraries

7. Which tool is used to compile Java source code?

A) `java`
B) `javac`
C) `javadoc`
D) `jar`
Answer: B) `javac`

8. The `java` command launches:
A) Compiler
B) JVM
C) Debugger
D) Profiler
Answer: B) JVM

9. JDK is needed for:
A) Only running Java programs
B) Developing and running Java programs
C) Only debugging
D) Only packaging JARs
Answer: B) Developing and running Java programs

10. Which is NOT part of JRE?
A) JVM
B) Class libraries
C) `javac`
D) `rt.jar`
Answer: C) `javac`

Medium MCQs (10)

1. The JVM performs:
A) Only interpretation
B) Only compilation
C) Just-In-Time (JIT) compilation
D) Ahead-of-Time (AOT) compilation only
Answer: C) Just-In-Time (JIT) compilation

2. Which memory area stores object instances?
A) Stack
B) Method Area

C) Heap
D) PC Register
Answer: C) Heap

3. The Method Area in JVM stores:
A) Local variables
B) Object data
C) Class metadata, static variables, and method code
D) Thread execution state
Answer: C) Class metadata, static variables, and method code

4. What does the Bytecode Verifier do?
A) Optimizes loops
B) Checks bytecode for safety before execution
C) Converts bytecode to machine code
D) Manages garbage collection
Answer: B) Checks bytecode for safety before execution

5. Which JVM option prints version info?
A) `-version`
B) `--version`
C) `-showversion`
D) All of the above
Answer: D) All of the above

6. The JIT compiler improves performance by:
A) Compiling bytecode to native code at runtime
B) Interpreting every instruction slowly
C) Disabling garbage collection
D) Reducing heap size
Answer: A) Compiling bytecode to native code at runtime

7. Which is true about JVM implementations?
A) Only Oracle provides JVM
B) OpenJDK, IBM J9, and Azul Zulu are JVM implementations
C) JVM is same as JRE
D) JVM cannot be replaced
Answer: B) OpenJDK, IBM J9, and Azul Zulu are JVM implementations

8. The `java.lang` package is loaded by:
   A) Bootstrap ClassLoader
   B) Extension ClassLoader
   C) Application ClassLoader
   D) Custom ClassLoader
   Answer: A) Bootstrap ClassLoader

9. Garbage Collection in JVM primarily reclaims memory from:
   A) Stack
   B) Heap
   C) Method Area
   D) Native Memory
   Answer: B) Heap

10. Which command lists all JVM system properties?
    A) `java -props`
    B) `java -Dlist`
    C) `java -XX:+PrintFlagsFinal`
    D) Not directly possible; must use code
    Answer: D) Not directly possible; must use code

Hard MCQs (10)

1. In JVM architecture, the Execution Engine includes:
   A) Interpreter, JIT Compiler, and Garbage Collector
   B) Interpreter and JIT Compiler only
   C) ClassLoader and Memory Manager
   D) Only the Interpreter
   Answer: B) Interpreter and JIT Compiler only
   (Note: GC is part of Memory Management, not Execution Engine)

2. Which garbage collector was the default in Java 8?
   A) G1
   B) Parallel GC
   C) CMS
   D) ZGC
   Answer: B) Parallel GC

3. The Metaspace (introduced in Java 8) replaced:

A) Heap
B) Stack
C) Permanent Generation (PermGen)
D) Code Cache
Answer: C) Permanent Generation (PermGen)

4. What is the purpose of the `-Xmx` JVM flag?
A) Set initial heap size
B) Set maximum heap size
C) Enable debugging
D) Disable JIT
Answer: B) Set maximum heap size

5. Which class loader loads classes from the `-classpath`?
A) Bootstrap
B) Extension
C) System (Application) ClassLoader
D) Custom
Answer: C) System (Application) ClassLoader

6. The JVM specification is maintained by:
A) Oracle only
B) Java Community Process (JCP)
C) ISO
D) W3C
Answer: B) Java Community Process (JCP)

7. What happens if the heap memory is exhausted and GC cannot reclaim enough?
A) JVM restarts
B) `OutOfMemoryError` is thrown
C) Program pauses indefinitely
D) OS kills the process silently
Answer: B) `OutOfMemoryError` is thrown

8. Which of the following is stored in the JVM Stack?
A) Object instances
B) Static variables
C) Local variables and method call frames
D) String literals

Answer: C) Local variables and method call frames

9. The `-XX:+UseG1GC` flag enables:
   A) Parallel Garbage Collector
   B) Concurrent Mark Sweep
   C) Garbage-First (G1) Collector
   D) Z Garbage Collector
   Answer: C) Garbage-First (G1) Collector

10. Native Method Stack is used for:
   A) Java method calls
   B) Storing primitive arrays
   C) Executing native (C/C++) methods via JNI
   D) Loading JAR files
   Answer: C) Executing native (C/C++) methods via JNI

Coding Problems

Easy Coding (10)

Q1. Write a program that prints the maximum memory available to JVM.

```
public class MaxMemory {
    public static void main(String[] args) {
        long max = Runtime.getRuntime().maxMemory();
        System.out.println("Max Memory: " + max + " bytes");
    }
}
```

Q2. Create a program that prints total and free memory.

```
public class MemoryInfo {
    public static void main(String[] args) {
        Runtime rt = Runtime.getRuntime();
        System.out.println("Total: " + rt.totalMemory());
        System.out.println("Free: " + rt.freeMemory());
    }
}
```

Q3. Write a program that lists all input arguments and exits.

```
public class EchoArgs {
    public static void main(String[] args) {
```

```
        for (String arg : args) {
            System.out.println(arg);
        }
    }
}
```

Q4. Print the current working directory using Java.

```
public class CurrentDir {
    public static void main(String[] args) {
        System.out.println("CWD: " + System.getProperty("user.dir"));
    }
}
```

Q5. Write a program that triggers a minor GC suggestion.

```
public class SuggestGC {
    public static void main(String[] args) {
        System.gc(); // Suggests GC (not guaranteed)
        System.out.println("GC suggested");
    }
}
```

Q6. Print the number of available processors.

```
public class CPUCount {
    public static void main(String[] args) {
        System.out.println("CPUs: " + Runtime.getRuntime().availableProcessors());
    }
}
```

Q7. Write a program that exits with status code 1.

```
public class ExitWithCode {
    public static void main(String[] args) {
        System.exit(1);
    }
}
```

Q8. Print the temporary directory path.

```
public class TempDir {
    public static void main(String[] args) {
        System.out.println("Temp Dir: " + System.getProperty("java.io.tmpdir"));
```

```
    }
}
```

Q9. Create a program that prints its own PID (Java 9+).

```
public class PrintPID {
    public static void main(String[] args) {
        long pid = ProcessHandle.current().pid();
        System.out.println("PID: " + pid);
    }
}
```

Q10. Write a program that prints the classpath.

```
public class ShowClassPath {
    public static void main(String[] args) {
        System.out.println("Classpath: " + System.getProperty("java.class.path"));
    }
}
```

Medium Coding (10)

Q11. Monitor memory usage before and after creating 10,000 strings.

```
public class MemoryMonitor {
    static void printMemory(String label) {
        Runtime rt = Runtime.getRuntime();
        System.out.printf("%s: Free=%d, Total=%d%n", label, rt.freeMemory(),
rt.totalMemory());
    }

    public static void main(String[] args) {
        printMemory("Before");
        String[] arr = new String[10000];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = "String-" + i;
        }
        printMemory("After");
    }
}
```

Q12. List all garbage collector names using ManagementFactory.

```
import java.lang.management.GarbageCollectorMXBean;
import java.lang.management.ManagementFactory;
public class ListGCs {
    public static void main(String[] args) {
        for (GarbageCollectorMXBean gc :
ManagementFactory.getGarbageCollectorMXBeans()) {
            System.out.println("GC: " + gc.getName());
        }
    }
}
```

Q13. Write a program that forces an `OutOfMemoryError` (educational).

```
import java.util.ArrayList;
import java.util.List;
public class ForceOOM {
    public static void main(String[] args) {
        List<byte[]> list = new ArrayList<>();
        try {
            while (true) {
                list.add(new byte[1024 * 1024]); // 1MB chunks
            }
        } catch (OutOfMemoryError e) {
            System.out.println("Caught OOM: " + e.getMessage());
        }
    }
}
```

Q14. Get JVM uptime in milliseconds.

```
import java.lang.management.ManagementFactory;
public class JVMTiming {
    public static void main(String[] args) {
        long uptime = ManagementFactory.getRuntimeMXBean().getUptime();
        System.out.println("JVM Uptime: " + uptime + " ms");
    }
}
```

Q15. Print all input arguments as a single string.

```
public class JoinArgs {
```

```java
    public static void main(String[] args) {
        System.out.println(String.join(" ", args));
    }
}
```

Q16. Detects if running in 64-bit JVM.

```java
public class BitnessCheck {
    public static void main(String[] args) {
        String arch = System.getProperty("sun.arch.data.model");
        if (arch == null) arch = System.getProperty("os.arch");
        System.out.println("Architecture: " + arch + "-bit");
    }
}
```

Q17. Write a program that lists all system properties starting with "java.vm".

```java
import java.util.Properties;
public class VMProps {
    public static void main(String[] args) {
        Properties p = System.getProperties();
        p.stringPropertyNames().stream()
         .filter(k -> k.startsWith("java.vm"))
         .sorted()
         .forEach(k -> System.out.println(k + " = " + p.getProperty(k)));
    }
}
```

Q18. Create a thread that runs for 2 seconds and prints “Alive”.

```java
public class ShortThread {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new Thread(() -> {
            try { Thread.sleep(2000); } catch (InterruptedException e) {}
            System.out.println("Alive");
        });
        t.start();
        t.join();
    }
}
```

Q19. Print the name of the default charset.

```
import java.nio.charset.Charset;
public class DefaultCharset {
    public static void main(String[] args) {
        System.out.println("Default Charset: " + Charset.defaultCharset());
    }
}
```

Q20. Write a program that uses `Runtime.exec()` to run `java -version`.

```
import java.io.*;
public class RunJavaVersion {
    public static void main(String[] args) throws Exception {
        Process p = Runtime.getRuntime().exec("java -version");
        BufferedReader err = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
        String line;
        while ((line = err.readLine()) != null) {
            System.out.println(line);
        }
    }
}
```

Hard Coding (10)

Q21. Monitor GC events using a notification listener (Java 7+).

```
import java.lang.management.*;
import javax.management.*;
public class GCCallback {
    public static void main(String[] args) throws Exception {
        for (GarbageCollectorMXBean gcBean :
ManagementFactory.getGarbageCollectorMXBeans()) {
            NotificationEmitter emitter = (NotificationEmitter) gcBean;
            emitter.addNotificationListener((notification, handback) -> {
                if
(notification.getType().equals(GarbageCollectionNotificationInfo.GARBAGE_COLLECTI
ON_NOTIFICATION)) {
                    GarbageCollectionNotificationInfo info =
                        GarbageCollectionNotificationInfo.from(notification.getCompositeData());
                    System.out.println("GC: " + info.getGcName() +
```

```
                                ", Duration: " + info.getGcInfo().getDuration() + "ms");
                }
            }, null, null);
        }
        // Trigger GC
        for (int i = 0; i < 100000; i++) new Object();
        System.gc();
        Thread.sleep(1000);
    }
}
```

Q22. Estimate object size using `Instrumentation` (requires agent).
> *Note: Full agent setup is complex; here's the class-side code.*

```
import java.lang.instrument.Instrumentation;
public class ObjectSizeFetcher {
    private static Instrumentation instrumentation;

    public static void premain(String args, Instrumentation inst) {
        instrumentation = inst;
    }

    public static long getObjectSize(Object obj) {
        if (instrumentation == null) {
            throw new IllegalStateException("Instrumentation not initialized. Use
-javaagent");
        }
        return instrumentation.getObjectSize(obj);
    }

    public static void main(String[] args) {
        try {
            System.out.println("Size of new Object(): " + getObjectSize(new Object()));
        } catch (Exception e) {
            System.out.println("Run with -javaagent to enable size measurement.");
        }
    }
}
```

Q23. Detect JVM vendor and version programmatically.

```
public class JVMDetails {
    public static void main(String[] args) {
        System.out.println("Vendor: " + System.getProperty("java.vendor"));
        System.out.println("VM Name: " + System.getProperty("java.vm.name"));
        System.out.println("VM Version: " + System.getProperty("java.vm.version"));
        System.out.println("Spec Version: " +
System.getProperty("java.specification.version"));
    }
}
```

Q24. Simulate stack overflow via infinite recursion.

```
public class StackOverflowDemo {
    static void recurse() {
        recurse(); // No base case
    }

    public static void main(String[] args) {
        try {
            recurse();
        } catch (StackOverflowError e) {
            System.out.println("Stack overflow detected.");
        }
    }
}
```

Q25. Write a custom class loader that loads a class from byte array.

```
public class CustomClassLoader extends ClassLoader {
    public Class<?> defineClassFromBytes(String name, byte[] bytes) {
        return defineClass(name, bytes, 0, bytes.length);
    }

    public static void main(String[] args) throws Exception {
        // Normally, you'd read .class file bytes here
        System.out.println("CustomClassLoader ready for byte[] input.");
    }
}
```

Q26. Measure time spent in GC using MXBeans.

```
import java.lang.management.GarbageCollectorMXBean;
import java.lang.management.ManagementFactory;
public class GCTimeTracker {
    public static void main(String[] args) throws InterruptedException {
        long initialGCTime = getTotalGCTime();
        // Allocate memory
        for (int i = 0; i < 500000; i++) new Object();
        System.gc();
        Thread.sleep(100);
        long finalGCTime = getTotalGCTime();
        System.out.println("GC Time Spent: " + (finalGCTime - initialGCTime) + " ms");
    }

    static long getTotalGCTime() {
        return ManagementFactory.getGarbageCollectorMXBeans().stream()
                .mapToLong(GarbageCollectorMXBean::getCollectionTime)
                .sum();
    }
}
```

Q27. Print all thread names currently running in JVM.

```
import java.util.Set;
public class ThreadLister {
    public static void main(String[] args) {
        Set<Thread> threads = Thread.getAllStackTraces().keySet();
        threads.forEach(t -> System.out.println("Thread: " + t.getName()));
    }
}
```

Q28. Detect if JVM is running in server mode.

```
public class ServerModeCheck {
    public static void main(String[] args) {
        String vmInfo = System.getProperty("java.vm.info");
        if (vmInfo != null && vmInfo.contains("server")) {
            System.out.println("JVM is in server mode");
        } else {
            System.out.println("JVM mode: " + (vmInfo != null ? vmInfo : "Unknown"));
        }
    }
```

```
}
```

Q29. Create a memory leak via static collection (common real-world bug).

```
import java.util.ArrayList;
import java.util.List;
public class StaticLeak {
    private static List<Object> LEAKY_LIST = new ArrayList<>();

    public static void main(String[] args) throws InterruptedException {
        while (true) {
            LEAKY_LIST.add(new byte[1024 * 1024]); // 1MB
            Thread.sleep(100);
        }
    }
}
```

Q30. Use JMX to connect to own JVM and fetch heap usage.

```
import java.lang.management.ManagementFactory;
import java.lang.management.MemoryMXBean;
import java.lang.management.MemoryUsage;
public class SelfJMX {
    public static void main(String[] args) {
        MemoryMXBean memBean = ManagementFactory.getMemoryMXBean();
        MemoryUsage heap = memBean.getHeapMemoryUsage();
        System.out.println("Heap Used: " + heap.getUsed() + " / " + heap.getMax());
    }
}
```

TOPIC 3: Setting up Java Environment & IDEs

Easy MCQs (10)

1. Which command checks if Java is installed on Windows?
   A) `java --version`
   B) `javac -version`
   C) Both A and B
   D) `java version`
   Answer: C) Both A and B

2. On macOS, where is JDK typically installed?
   A) `/usr/bin/java`
   B) `/Library/Java/JavaVirtualMachines/`
   C) `C:\Program Files\Java`
   D) `/opt/java`
   Answer: B) `/Library/Java/JavaVirtualMachines/`

3. Which file stores system-wide environment variables on Linux?
   A) `.bashrc`
   B) `/etc/environment`
   C) `~/.profile`
   D) All of the above
   Answer: D) All of the above

4. In VS Code, which extension is essential for Java development?
   A) Python
   B) Java Extension Pack
   C) C/C++
   D) Debugger for Chrome
   Answer: B) Java Extension Pack

5. What does `JAVA_HOME` point to?
   A) JRE directory
   B) JDK installation root
   C) JVM executable
   D) User’s home directory
   Answer: B) JDK installation root

6. Which IDE is developed by Apache?
   A) Eclipse
   B) IntelliJ IDEA
   C) NetBeans
   D) VS Code
   Answer: C) NetBeans

7. On Windows, environment variables are set via:
   A) Control Panel → System → Advanced → Environment Variables
   B) Registry Editor

C) Command Prompt only
D) Not configurable
Answer: A) Control Panel → System → Advanced → Environment Variables

8. Which command compiles a Java file named `Hello.java`?
A) `java Hello.java`
B) `javac Hello.java`
C) `compile Hello`
D) `run Hello`
Answer: B) `javac Hello.java`

9. In Eclipse, a “workspace” is:
A) A single Java file
B) A folder containing multiple projects
C) The IDE executable
D) A temporary cache
Answer: B) A folder containing multiple projects

10. Which key combination runs a Java program in most IDEs?
A) Ctrl + S
B) Ctrl + F5
C) Ctrl + Shift + F10 (IntelliJ) or green (Eclipse/VS Code)
D) Alt + R
Answer: C) Ctrl + Shift + F10 (IntelliJ) or green (Eclipse/VS Code)

Medium MCQs (10)

1. If `java -version` works but `javac` is not recognized, what is likely missing?
A) JRE only installed
B) JDK not installed or PATH not set
C) Corrupted JVM
D) IDE not installed
Answer: B) JDK not installed or PATH not set

2. On Linux, which command permanently adds JDK to PATH in `.bashrc`?
A) `export PATH=$PATH:/path/to/jdk/bin`
B) `set PATH=/path/to/jdk`
C) `java_home=/path/to/jdk`
D) `update-java-alternatives`

Answer: A) `export PATH=$PATH:/path/to/jdk/bin`

3. In VS Code, where do you configure `JAVA_HOME` for the Java extension?
   A) `settings.json`
   B) `launch.json`
   C) `tasks.json`
   D) All of the above
   Answer: A) `settings.json`

4. Which tool manages multiple JDK versions on macOS/Linux?
   A) Maven
   B) Gradle
   C) SDKMAN! or jenv
   D) Docker
   Answer: C) SDKMAN! or jenv

5. What is the purpose of the `.classpath` file in Eclipse?
   A) Stores project metadata
   B) Defines build path and dependencies
   C) Contains source code
   D) Logs runtime errors
   Answer: B) Defines build path and dependencies

6. In NetBeans, “Clean and Build” does what?
   A) Deletes source files
   B) Compiles all files and creates JAR
   C) Only runs the program
   D) Updates JDK
   Answer: B) Compiles all files and creates JAR

7. Which file in VS Code defines debug configurations?
   A) `settings.json`
   B) `launch.json`
   C) `pom.xml`
   D) `.project`
   Answer: B) `launch.json`

8. On Windows, if `JAVA_HOME` is set but `javac` still fails, what is missing?
   A) `%JAVA_HOME%\bin` not in PATH

B) JDK not installed
C) Antivirus blocking
D) User permissions
Answer: A) `%JAVA_HOME%\bin` not in PATH

9. Which command lists all installed Java versions on macOS?
A) `java -list`
B) `/usr/libexec/java_home -V`
C) `ls /Library/Java`
D) `java --versions`
Answer: B) `/usr/libexec/java_home -V`

10. In Eclipse, “Problems” view shows:
A) Runtime exceptions
B) Compilation errors and warnings
C) Memory leaks
D) Network issues
Answer: B) Compilation errors and warnings

Hard MCQs (10)

1. Which environment variable does Maven use to locate Java?
A) `JAVA_HOME`
B) `JRE_HOME`
C) `PATH` only
D) `M2_HOME`
Answer: A) `JAVA_HOME`

2. In VS Code, if the Java debugger fails to attach, what is a likely cause?
A) Missing `launch.json` with correct `mainClass`
B) Wrong file extension
C) Using JRE instead of JDK
D) All of the above
Answer: D) All of the above

3. On Linux, which command sets default Java version system-wide?
A) `update-alternatives --config java`
B) `java-select`
C) `set-java-version`

D) `alternatives --java`
Answer: A) `update-alternatives --config java`

4. What does the Eclipse `.metadata` folder contain?
A) Source code
B) Workspace settings, plugin data, and state
C) Compiled `.class` files
D) JAR libraries
Answer: B) Workspace settings, plugin data, and state

5. Which file does NetBeans use to store project properties?
A) `build.xml`
B) `project.properties`
C) `nbproject/project.properties`
D) `.netbeans`
Answer: C) `nbproject/project.properties`

6. In a multi-module Maven project in VS Code, what enables Java support?
A) Only `pom.xml`
B) Java Extension Pack + Maven plugin
C) Manual classpath setup
D) Eclipse compatibility mode
Answer: B) Java Extension Pack + Maven plugin

7. If `echo %JAVA_HOME%` shows correct path on Windows but `javac` fails, what is the issue?
A) `%JAVA_HOME%\bin` not in PATH
B) JDK corrupted
C) Command Prompt not restarted
D) Both A and C
Answer: D) Both A and C

8. Which IDE uses OSGi framework internally?
A) NetBeans
B) IntelliJ IDEA
C) Eclipse
D) VS Code
Answer: C) Eclipse

Powered By BridgeLabz

9. In VS Code, the “Java Projects” view appears only if:
   A) A `src` folder exists with `.java` files
   B) `JAVA_HOME` is set
   C) The folder is opened as a workspace
   D) All of the above
   Answer: D) All of the above

10. What is the minimal folder structure for a Java project in VS Code?
   A) `project/Hello.java`
   B) `project/src/Hello.java`
   C) `project/com/example/Hello.java`
   D) Any structure; VS Code infers from files
   Answer: B) `project/src/Hello.java` *(recommended for standard setup)*

Coding Problems

Easy Coding (10)

Q1. Write a program that prints “Java setup is working!” if it runs.

```
public class SetupCheck {
    public static void main(String[] args) {
        System.out.println("Java setup is working!");
    }
}
```

Q2. Create a program that prints the current directory (works on all OS).

```
public class CurrentDirectory {
    public static void main(String[] args) {
        System.out.println("Current Dir: " + System.getProperty("user.dir"));
    }
}
```

Q3. Write a program that outputs the Java executable path (approximate).

```
public class JavaPath {
    public static void main(String[] args) {
        System.out.println("Java Home: " + System.getProperty("java.home"));
    }
}
```

Q4. Print all command-line arguments (useful for script testing).

```
public class ArgsPrinter {
   public static void main(String[] args) {
      for (int i = 0; i < args.length; i++) {
         System.out.println("Arg[" + i + "] = " + args[i]);
      }
   }
}
```

Q5. Write a program that creates a file `setup_test.txt` in the current directory.

```
import java.io.FileWriter;
import java.io.IOException;
public class CreateTestFile {
   public static void main(String[] args) {
      try (FileWriter fw = new FileWriter("setup_test.txt")) {
         fw.write("Java environment is functional.\n");
         System.out.println("File created: setup_test.txt");
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Q6. Print the OS name and version.

```
public class OSInfo {
   public static void main(String[] args) {
      System.out.println("OS: " + System.getProperty("os.name") + " " +
System.getProperty("os.version"));
   }
}
```

Q7. Write a program that exits with code 0 (success).

```
public class SuccessExit {
   public static void main(String[] args) {
      System.exit(0);
   }
}
```

Q8. Print the user’s home directory.

```
public class HomeDir {
    public static void main(String[] args) {
        System.out.println("Home: " + System.getProperty("user.home"));
    }
}
```

Q9. Create a program that prints “Compiled and running!” to confirm toolchain.

```
public class ToolchainTest {
    public static void main(String[] args) {
        System.out.println("Compiled and running!");
    }
}
```

Q10. Write a program that lists all files in the current directory.

```
import java.io.File;
public class ListFiles {
    public static void main(String[] args) {
        File dir = new File(".");
        for (File f : dir.listFiles()) {
            System.out.println(f.getName());
        }
    }
}
```

Medium Coding (10)

Q11. Validate that `JAVA_HOME` is set and points to a valid JDK.

```
import java.io.File;
public class JavaHomeValidator {
    public static void main(String[] args) {
        String javaHome = System.getenv("JAVA_HOME");
        if (javaHome == null || javaHome.isEmpty()) {
            System.out.println("JAVA_HOME not set");
            return;
        }
        File bin = new File(javaHome, "bin");
        File javac = new File(bin, "javac" +
(System.getProperty("os.name").startsWith("Windows") ? ".exe" : ""));
```

```java
        if (javac.exists()) {
            System.out.println("Valid JDK at: " + javaHome);
        } else {
            System.out.println("JAVA_HOME set but no javac found");
        }
    }
}
```

Q12. Simulate a build script that compiles and runs a class.

```java
import java.io.File;
import java.io.IOException;
public class SimulatedBuild {
    public static void main(String[] args) throws IOException, InterruptedException {
        // Assume Hello.java exists in current dir
        Process compile = Runtime.getRuntime().exec("javac Hello.java");
        compile.waitFor();
        if (compile.exitValue() == 0) {
            Process run = Runtime.getRuntime().exec("java Hello");
            run.waitFor();
        }
    }
}
```

Q13. Detect if running inside an IDE (heuristic: check for common system properties).

```java
public class IDEDetector {
    public static void main(String[] args) {
        String[] ideProps = {"idea.launcher.bin.path", "eclipse.launcher", "netbeans.home"};
        boolean inIDE = false;
        for (String prop : ideProps) {
            if (System.getProperty(prop) != null) {
                inIDE = true;
                break;
            }
        }
        System.out.println("Running in IDE: " + inIDE);
    }
}
```

Q14. Write a program that creates a standard Java project structure.

```
import java.io.File;
public class ProjectScaffold {
    public static void main(String[] args) {
        new File("src").mkdirs();
        new File("bin").mkdirs();
        System.out.println("Created: src/, bin/");
    }
}
```

Q15. Print the full path of the running JAR (or class directory).

```
public class CodeLocation {
    public static void main(String[] args) {
        String location = CodeLocation.class.getProtectionDomain()
            .getCodeSource().getLocation().getPath();
        System.out.println("Code location: " + location);
    }
}
```

Q16. Check if a file `Hello.java` exists and is readable.

```
import java.io.File;
public class SourceChecker {
    public static void main(String[] args) {
        File f = new File("Hello.java");
        if (f.exists() && f.canRead()) {
            System.out.println("Source file ready");
        } else {
            System.out.println("Missing or unreadable: Hello.java");
        }
    }
}
```

Q17. Write a program that outputs environment variables related to Java.

```
import java.util.Map;
public class JavaEnvVars {
    public static void main(String[] args) {
        Map<String, String> env = System.getenv();
        env.keySet().stream()
           .filter(k -> k.toLowerCase().contains("java"))
```

```
        .sorted()
        .forEach(k -> System.out.println(k + "=" + env.get(k)));
    }
}
```

Q18. Create a timestamped log file for build diagnostics.

```
import java.io.FileWriter;
import java.time.LocalDateTime;
public class BuildLogger {
    public static void main(String[] args) throws Exception {
        String timestamp = LocalDateTime.now().toString().replace(":", "-");
        try (FileWriter fw = new FileWriter("build_" + timestamp + ".log")) {
            fw.write("Build started at: " + timestamp + "\n");
            fw.write("Java Version: " + System.getProperty("java.version") + "\n");
        }
        System.out.println("Log created");
    }
}
```

Q19. Simulate a cross-platform command executor.

```
public class PlatformCommand {
    public static void main(String[] args) throws Exception {
        String os = System.getProperty("os.name").toLowerCase();
        String[] cmd = os.contains("win") ? new String[]{"cmd", "/c", "dir"} : new
String[]{"ls"};
        Process p = Runtime.getRuntime().exec(cmd);
        p.getInputStream().transferTo(System.out);
    }
}
```

Q20. Write a program that checks write permission in current directory.

```
import java.io.File;
import java.io.IOException;
public class WritePermissionCheck {
    public static void main(String[] args) {
        File test = new File("perm_test.tmp");
        try {
            if (test.createNewFile()) {
                System.out.println("Write permission: OK");
```

```
            test.delete();
        }
    } catch (IOException e) {
        System.out.println("Write permission: DENIED");
    }
  }
}
```

Hard Coding (10)

Q21. Auto-detect JDK path on current system (Windows/macOS/Linux).

```
import java.io.File;
public class JDKDetector {
  static String detectJDK() {
    // Try JAVA_HOME first
    String javaHome = System.getenv("JAVA_HOME");
    if (javaHome != null && isValidJDK(javaHome)) return javaHome;

    String os = System.getProperty("os.name").toLowerCase();
    if (os.contains("win")) {
      File f = new File("C:\\Program Files\\Java");
      if (f.exists()) {
        for (File d : f.listFiles()) {
          if (d.getName().startsWith("jdk") && isValidJDK(d.getPath())) {
            return d.getPath();
          }
        }
      }
    } else if (os.contains("mac")) {
      File f = new File("/Library/Java/JavaVirtualMachines");
      if (f.exists()) {
        for (File d : f.listFiles()) {
          File jdk = new File(d, "Contents/Home");
          if (isValidJDK(jdk.getPath())) return jdk.getPath();
        }
      }
    } else { // Linux
      String[] paths = {"/usr/lib/jvm", "/opt/java"};
      for (String p : paths) {
```

```
                File dir = new File(p);
                if (dir.exists()) {
                    for (File d : dir.listFiles()) {
                        if (d.getName().contains("jdk") && isValidJDK(d.getPath())) {
                            return d.getPath();
                        }
                    }
                }
            }
        }
        return null;
    }

    static boolean isValidJDK(String path) {
        File javac = new File(path + "/bin/javac" +
(System.getProperty("os.name").startsWith("Windows") ? ".exe" : ""));
        return javac.exists();
    }

    public static void main(String[] args) {
        String jdk = detectJDK();
        System.out.println("Detected JDK: " + (jdk != null ? jdk : "Not found"));
    }
}
```

Q22. Generate a `.vscode/settings.json` for Java project programmatically.

```
import java.io.FileWriter;
import java.io.IOException;
public class VSCodeConfigGenerator {
    public static void main(String[] args) throws IOException {
        String config = "{\n" +
            "  \"java.home\": \"" + System.getProperty("java.home").replace("\\", "\\\\") +
"\",\n" +
            "  \"java.configuration.updateBuildConfiguration\": \"interactive\"\n" +
            "}";
        new File(".vscode").mkdirs();
        try (FileWriter fw = new FileWriter(".vscode/settings.json")) {
            fw.write(config);
        }
```

```
        System.out.println("Generated .vscode/settings.json");
    }
}
```

Q23. Create an Eclipse `.project` and `.classpath` generator.

```
import java.io.FileWriter;
import java.io.IOException;
public class EclipseProjectGenerator {
    public static void main(String[] args) throws IOException {
        // .project
        String project = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<projectDescription>\n" +
            "  <name>MyProject</name>\n" +
            "  <buildSpec>\n" +
            "    <buildCommand>\n" +
            "      <name>org.eclipse.jdt.core.javabuilder</name>\n" +
            "    </buildCommand>\n" +
            "  </buildSpec>\n" +
            "  <natures>\n" +
            "    <nature>org.eclipse.jdt.core.javanature</nature>\n" +
            "  </natures>\n" +
            "</projectDescription>";

        // .classpath
        String classpath = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<classpath>\n" +
            "  <classpathentry kind=\"con\"
path=\"org.eclipse.jdt.launching.JRE_CONTAINER\"/>\n" +
            "  <classpathentry kind=\"src\" path=\"src\"/>\n" +
            "  <classpathentry kind=\"output\" path=\"bin\"/>\n" +
            "</classpath>";

        try (FileWriter fw1 = new FileWriter(".project");
             FileWriter fw2 = new FileWriter(".classpath")) {
            fw1.write(project);
            fw2.write(classpath);
        }
        System.out.println("Eclipse project files generated.");
    }
```

```
}
```

Q24. Simulate NetBeans `build.xml` creation for Ant.

```
import java.io.FileWriter;
import java.io.IOException;
public class AntBuildGenerator {
    public static void main(String[] args) throws IOException {
        String buildXml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<project name=\"MyApp\" default=\"default\" basedir=\".\">\n" +
            "  <property name=\"src.dir\" value=\"src\"/>\n" +
            "  <property name=\"build.dir\" value=\"build\"/>\n" +
            "  <property name=\"classes.dir\" value=\"${build.dir}/classes\"/>\n" +
            "  <target name=\"init\">\n" +
            "    <mkdir dir=\"${classes.dir}\"/>\n" +
            "  </target>\n" +
            "  <target name=\"compile\" depends=\"init\">\n" +
            "    <javac srcdir=\"${src.dir}\" destdir=\"${classes.dir}\"/>\n" +
            "  </target>\n" +
            "</project>";

        try (FileWriter fw = new FileWriter("build.xml")) {
            fw.write(buildXml);
        }
        System.out.println("Ant build.xml generated.");
    }
}
```

Q25. Write a program that validates IDE project structure integrity.

```
import java.io.File;
public class ProjectValidator {
    public static void main(String[] args) {
        boolean valid = true;
        File src = new File("src");
        if (!src.exists() || !src.isDirectory()) {
            System.out.println("Missing src/ directory");
            valid = false;
        }
        File mainClass = new File("src/Hello.java");
        if (!mainClass.exists()) {
```

```
            System.out.println("Missing src/Hello.java");
            valid = false;
        }
        System.out.println("Project valid: " + valid);
    }
}
```

Q26. Detect if running in headless mode (common in servers/CI).

```
public class HeadlessCheck {
    public static void main(String[] args) {
        boolean headless = Boolean.getBoolean("java.awt.headless") ||
                    System.getProperty("java.awt.graphicsenv", "").contains("Headless");
        System.out.println("Headless mode: " + headless);
    }
}
```

Q27. Generate a Maven `pom.xml` for a simple Java app.

```
import java.io.FileWriter;
import java.io.IOException;
public class MavenPOMGenerator {
    public static void main(String[] args) throws IOException {
        String pom = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<project xmlns=\"http://maven.apache.org/POM/4.0.0\"\n" +
            "         xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"\n" +
            "         xsi:schemaLocation=\"http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd\">\n" +
            "  <modelVersion>4.0.0</modelVersion>\n" +
            "  <groupId>com.example</groupId>\n" +
            "  <artifactId>my-app</artifactId>\n" +
            "  <version>1.0</version>\n" +
            "  <properties>\n" +
            "    <maven.compiler.source>11</maven.compiler.source>\n" +
            "    <maven.compiler.target>11</maven.compiler.target>\n" +
            "  </properties>\n" +
            "</project>";

        try (FileWriter fw = new FileWriter("pom.xml")) {
            fw.write(pom);
        }
```

```
        System.out.println("pom.xml generated.");
    }
}
```

Q28. Create a cross-platform script generator (`.bat` and `.sh`).

```
import java.io.FileWriter;
import java.io.IOException;
public class ScriptGenerator {
    public static void main(String[] args) throws IOException {
        // Windows batch
        try (FileWriter bat = new FileWriter("run.bat")) {
            bat.write("@echo off\njava -cp . Hello\npause\n");
        }
        // Unix shell
        try (FileWriter sh = new FileWriter("run.sh")) {
            sh.write("#!/bin/bash\njava -cp . Hello\n");
        }
        new File("run.sh").setExecutable(true);
        System.out.println("Scripts generated: run.bat, run.sh");
    }
}
```

Q29. Monitor file changes in `src/` to simulate IDE auto-build.

```
import java.io.IOException;
import java.nio.file.*;
public class AutoBuildSimulator {
    public static void main(String[] args) throws IOException {
        WatchService watcher = FileSystems.getDefault().newWatchService();
        Paths.get("src").register(watcher, StandardWatchEventKinds.ENTRY_MODIFY);
        System.out.println("Watching src/ for changes...");
        while (true) {
            WatchKey key = watcher.take();
            for (WatchEvent<?> event : key.pollEvents()) {
                System.out.println("Change detected: " + event.context() + " → Rebuilding...");
            }
            key.reset();
        }
    }
}
```

Q30. Generate a complete VS Code launch configuration for debugging.

```
import java.io.FileWriter;
import java.io.IOException;
public class LaunchConfigGenerator {
    public static void main(String[] args) throws IOException {
        String launch = "{\n" +
            "  \"version\": \"0.2.0\",\n" +
            "  \"configurations\": [\n" +
            "    {\n" +
            "      \"type\": \"java\",\n" +
            "      \"name\": \"Debug Hello\",\n" +
            "      \"request\": \"launch\",\n" +
            "      \"mainClass\": \"Hello\",\n" +
            "      \"projectName\": \"my-project\"\n" +
            "    }\n" +
            "  ]\n" +
            "}";
        new File(".vscode").mkdirs();
        try (FileWriter fw = new FileWriter(".vscode/launch.json")) {
            fw.write(launch);
        }
        System.out.println("Debug config: .vscode/launch.json");
    }
}
```

TOPIC 4: Compiling, Interpreting, `main()` Method & Command Line Arguments

Easy MCQs (10)

1. Which command compiles a Java source file `App.java`?
   A) `java App.java`
   B) `javac App.java`
   C) `compile App`
   D) `run App`
   Answer: B) `javac App.java`

2. The entry point of a Java application must be:
   A) `static void start()`
   B) `public static void main(String[] args)`
   C) `void main()`
   D) `public void main(String[] args)`
   Answer: B) `public static void main(String[] args)`

3. What is the file extension of a compiled Java class?
   A) `.java`
   B) `.class`
   C) `.jar`
   D) `.exe`
   Answer: B) `.class`

4. Command-line arguments are passed to which parameter?
   A) `String args`
   B) `String[] args`
   C) `int argc, char argv`
   D) `List<String> args`
   Answer: B) `String[] args`

5. Which keyword makes the `main` method accessible to the JVM?
   A) `static`
   B) `public`
   C) `void`
   D) `final`
   Answer: B) `public`

6. What happens if you omit `static` from `main`?
   A) Compiles but throws `NoSuchMethodError` at runtime
   B) Compilation error
   C) Runs normally
   D) JVM ignores it
   Answer: A) Compiles but throws `NoSuchMethodError` at runtime

7. How do you run a compiled class `Hello`?
   A) `javac Hello`
   B) `java Hello.class`
   C) `java Hello`

D) `run Hello`
Answer: C) `java Hello`

8. The `args` array in `main` is:
A) Always null
B) Empty if no arguments are passed
C) Contains program name as first element
D) Of type `Object[]`
Answer: B) Empty if no arguments are passed

9. Bytecode is platform-______?
A) Dependent
B) Independent
C) Specific
D) Locked
Answer: B) Independent

10. Which tool converts `.java` to `.class`?
A) JVM
B) JRE
C) JDK's `javac`
D) IDE debugger
Answer: C) JDK's `javac`

Medium MCQs (10)

1. What is the output of `java MyApp one two` if `main` prints `args.length`?
A) 0
B) 1
C) 2
D) 3
Answer: C) 2

2. Which statement about `main` is FALSE?
A) It can be overloaded
B) It must return `void`
C) It can have any name
D) It must be `static`
Answer: C) It can have any name

3. What does the JVM do after loading the class containing `main`?
   A) Instantiates the class
   B) Calls `main` directly (no instance needed)
   C) Runs a constructor
   D) Starts garbage collection
   Answer: B) Calls `main` directly (no instance needed)

4. If you run `java Test "a b" c`, what is `args[0]`?
   A) `"a`
   B) `a b`
   C) `"a b"`
   D) `b"`
   Answer: B) `a b`
   (Shell removes quotes; `args[0] = "a b"` as single string)

5. Which is a valid alternative signature for `main`?
   A) `public static void main(String... args)`
   B) `public void main(String[] args)`
   C) `static public int main(String[] args)`
   D) `public static void Main(String[] args)`
   Answer: A) `public static void main(String... args)`

6. What happens if multiple classes in a file have `main`?
   A) Compilation error
   B) JVM runs the first one
   C) You can choose which to run via class name
   D) Only one `main` allowed per package
   Answer: C) You can choose which to run via class name

7. The `.class` file contains:
   A) Source code
   B) Native machine code
   C) Bytecode
   D) HTML documentation
   Answer: C) Bytecode

8. Which phase checks for syntax errors?
   A) Interpretation

B) Compilation
C) Linking
D) Execution
Answer: B) Compilation

9. What is the minimum requirement for a runnable Java program?
A) A class with `main` method
B) A JAR file
C) An IDE
D) A `package` statement
Answer: A) A class with `main` method

10. If `args` is unused, can you omit it?
A) Yes, `main()` is valid
B) No, signature must match exactly
C) Only in Java 17+
D) Only with annotations
Answer: B) No, signature must match exactly

Hard MCQs (10)

1. What does the JVM do if it finds two `main` methods with valid signatures in the same class?
A) Compilation error
B) Picks the first one
C) Allows overloading; chooses based on invocation (but CLI always uses `String[]`)
D) Throws `AmbiguousMethodError`
Answer: C) Allows overloading; chooses based on invocation (but CLI always uses `String[]`)

2. In `public static void main(String[] args)`, why is `static` required?
A) To allow inheritance
B) So JVM can call it without creating an instance
C) To enable multithreading
D) For security reasons
Answer: B) So JVM can call it without creating an instance

3. What is the bytecode instruction that invokes `main`?
A) `invokevirtual`

B) `invokestatic`
C) `invokespecial`
D) `invokeinterface`
Answer: B) `invokestatic`

4. If you run `java -cp lib MyApp` and `MyApp.class` is in `lib/`, what must be true?
A) `MyApp` must be in default package
B) `lib` must contain `MyApp.class` at root
C) Both A and B
D) `MyApp` must be in a JAR
Answer: C) Both A and B

5. Which JVM flag prints the arguments passed to `main`?
A) `-verbose:args`
B) `-Xlog:arguments`
C) No built-in flag; must print in code
D) `-Dprint.args=true`
Answer: C) No built-in flag; must print in code

6. What is the maximum number of command-line arguments limited by?
A) JVM specification
B) Operating system command length
C) `args` array size (max 2^31-1)
D) Both B and C
Answer: D) Both B and C

7. Can `main` be `final`?
A) No, it causes compilation error
B) Yes, and it's common practice
C) Only in abstract classes
D) Only if class is `final`
Answer: B) Yes, and it's common practice

8. What happens if `main` throws an unchecked exception?
A) JVM catches it and exits with code 1
B) Program continues
C) Stack trace printed, JVM exits
D) Exception ignored
Answer: C) Stack trace printed, JVM exits

9. Which class loader loads the class containing `main`?
A) Bootstrap
B) Extension
C) Application (System) ClassLoader
D) Custom
Answer: C) Application (System) ClassLoader

10. In modular Java (JPMS), what must the module containing `main` declare?
A) `requires java.base;`
B) `exports main.package;`
C) `provides main.class;`
D) Nothing extra; `main` works as before
Answer: D) Nothing extra; `main` works as before

Coding Problems

Easy Coding (10)

Q1. Write a program that prints “Hello, World” if no arguments are given.

```
public class Greet {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Hello, World");
        }
    }
}
```

Q2. Print all command-line arguments, one per line.

```
public class PrintArgs {
    public static void main(String[] args) {
        for (String arg : args) {
            System.out.println(arg);
        }
    }
}
```

Q3. Write a program that exits with code 0 if arguments are provided, else 1.

```
public class ArgChecker {
    public static void main(String[] args) {
        System.exit(args.length > 0 ? 0 : 1);
    }
}
```

Q4. Print the number of arguments.

```
public class ArgCount {
    public static void main(String[] args) {
        System.out.println(args.length);
    }
}
```

Q5. Concatenate all arguments into a single string and print.

```
public class JoinArgs {
    public static void main(String[] args) {
        System.out.println(String.join(" ", args));
    }
}
```

Q6. Print the first argument or “No input” if none.

```
public class FirstArg {
    public static void main(String[] args) {
        System.out.println(args.length > 0 ? args[0] : "No input");
    }
}
```

Q7. Write a program that does nothing (valid `main`).

```
public class EmptyMain {
```

```
    public static void main(String[] args) {
        // Intentionally empty
    }
}
```

Q8. Print “Running” at start and “Done” at end.

```
public class Lifecycle {
    public static void main(String[] args) {
        System.out.println("Running");
        System.out.println("Done");
    }
}
```

Q9. Create a program that prints its own class name.

```
public class SelfName {
    public static void main(String[] args) {
        System.out.println(SelfName.class.getSimpleName());
    }
}
```

Q10. Write a program that prints each argument with its index.

```
public class IndexedArgs {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(i + ": " + args[i]);
        }
    }
}
```

Medium Coding (10)

Q11. Sum all numeric arguments; ignore non-numeric.

```
public class SumNumbers {
    public static void main(String[] args) {
        double sum = 0;
```

```
        for (String arg : args) {
            try {
                sum += Double.parseDouble(arg);
            } catch (NumberFormatException e) {
                // Ignore
            }
        }
        System.out.println(sum);
    }
}
```

Q12. Find the longest argument string.

```
public class LongestArg {
    public static void main(String[] args) {
        if (args.length == 0) return;
        String longest = args[0];
        for (String arg : args) {
            if (arg.length() > longest.length()) {
                longest = arg;
            }
        }
        System.out.println(longest);
    }
}
```

Q13. Reverse the order of arguments and print.

```
public class ReverseArgs {
    public static void main(String[] args) {
        for (int i = args.length - 1; i >= 0; i--) {
            System.out.print(args[i] + " ");
        }
        System.out.println();
    }
}
```

Q14. Count how many arguments are palindromes.

```
public class PalindromeArgs {
    static boolean isPalin(String s) {
        return s.equals(new StringBuilder(s).reverse().toString());
    }

    public static void main(String[] args) {
        int count = 0;
        for (String arg : args) {
            if (isPalin(arg)) count++;
        }
        System.out.println(count);
    }
}
```

Q15. Simulate a calculator: `java Calc 5 + 3` → output 8.

```
public class SimpleCalc {
    public static void main(String[] args) {
        if (args.length != 3) return;
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[2]);
        switch (args[1]) {
            case "+": System.out.println(a + b); break;
            case "-": System.out.println(a - b); break;
            case "*": System.out.println(a * b); break;
            case "/": System.out.println(a / b); break;
        }
    }
}
```

Q16. Print arguments that start with ‘-’ (simulate flags).

```
public class FlagDetector {
    public static void main(String[] args) {
        for (String arg : args) {
            if (arg.startsWith("-")) {
                System.out.println(arg);
            }
```

```
        }
    }
}
```

Q17. Write a program that echoes its arguments as a Java array literal.

```
public class ArrayLiteral {
    public static void main(String[] args) {
        System.out.print("String[] args = {");
        for (int i = 0; i < args.length; i++) {
            if (i > 0) System.out.print(", ");
            System.out.print("\"" + args[i] + "\"");
        }
        System.out.println("};");
    }
}
```

Q18. Check if all arguments are digits.

```
public class AllDigits {
    public static void main(String[] args) {
        boolean all = args.length > 0;
        for (String arg : args) {
            if (!arg.matches("\\d+")) {
                all = false;
                break;
            }
        }
        System.out.println(all);
    }
}
```

Q19. Group arguments by length.

```
import java.util.*;

public class GroupByLength {
    public static void main(String[] args) {
        Map<Integer, List<String>> groups = new TreeMap<>();
```

```
        for (String arg : args) {
            groups.computeIfAbsent(arg.length(), k -> new ArrayList<>()).add(arg);
        }
        groups.forEach((len, list) -> System.out.println(len + ": " + list));
    }
}
```

Q20. Simulate `grep`: print lines (args) containing a pattern (first arg).

```
public class SimpleGrep {
    public static void main(String[] args) {
        if (args.length < 2) return;
        String pattern = args[0];
        for (int i = 1; i < args.length; i++) {
            if (args[i].contains(pattern)) {
                System.out.println(args[i]);
            }
        }
    }
}
```

Hard Coding (10)

Q21. Parse key-value arguments like `-name John -age 30`.

```
import java.util.*;

public class KeyValueParser {
    public static void main(String[] args) {
        Map<String, String> map = new LinkedHashMap<>();
        for (int i = 0; i < args.length; i++) {
            if (args[i].startsWith("-") && i + 1 < args.length) {
                String key = args[i].substring(1);
                String value = args[++i];
                map.put(key, value);
            }
        }
        map.forEach((k, v) -> System.out.println(k + "=" + v));
    }
```

```
}
```

Q22. Handle quoted arguments with spaces (simulate shell parsing).
> *Note: Real shell parsing is complex; this handles simple cases.*

```
import java.util.*;

public class QuotedArgParser {
    public static void main(String[] args) {
        // This simulates if args were pre-split incorrectly.
        // In reality, shell handles quotes before Java sees them.
        // So this is educational only.
        System.out.println("Args as received by JVM:");
        for (String arg : args) {
            System.out.println("""" + arg + """");
        }
    }
}
```

Q23. Dynamically invoke `main` of another class via reflection.

```
public class MainInvoker {
    public static void main(String[] args) throws Exception {
        if (args.length < 1) return;
        String className = args[0];
        String[] targetArgs = Arrays.copyOfRange(args, 1, args.length);
        Class<?> cls = Class.forName(className);
        cls.getMethod("main", String[].class).invoke(null, (Object) targetArgs);
    }
}
```

Q24. Validate that `main` signature matches exactly using reflection.

```
import java.lang.reflect.Method;

public class MainValidator {
    public static void main(String[] args) throws Exception {
        if (args.length < 1) return;
```

```
        Class<?> cls = Class.forName(args[0]);
        try {
            Method m = cls.getMethod("main", String[].class);
            if (m.getReturnType() == void.class &&
                java.lang.reflect.Modifier.isPublic(m.getModifiers()) &&
                java.lang.reflect.Modifier.isStatic(m.getModifiers())) {
                System.out.println("Valid main method");
            } else {
                System.out.println("Invalid modifiers or return type");
            }
        } catch (NoSuchMethodException e) {
            System.out.println("No valid main method");
        }
    }
}
```

Q25. Measure execution time of `main` logic (excluding JVM startup).

```
public class TimedMain {
    public static void main(String[] args) {
        long start = System.nanoTime();
        // Simulate work
        for (String arg : args) {
            System.out.println(arg.toUpperCase());
        }
        long end = System.nanoTime();
        System.out.printf("Execution time: %.3f ms%n", (end - start) / 1_000_000.0);
    }
}
```

Q26. Redirect `System.out` to a file specified in args[0].

```
import java.io.*;

public class OutputRedirect {
    public static void main(String[] args) throws Exception {
        if (args.length > 0) {
            System.setOut(new PrintStream(new FileOutputStream(args[0])));
        }
```

```
        System.out.println("This goes to " + (args.length > 0 ? args[0] : "console"));
    }
}
```

Q27. Create a chain of `main` calls: A → B → C.

```
// Class A
class A {
    public static void main(String[] args) {
        System.out.println("A");
        B.main(args);
    }
}
class B {
    public static void main(String[] args) {
        System.out.println("B");
        C.main(args);
    }
}
class C {
    public static void main(String[] args) {
        System.out.println("C");
    }
}
```

Q28. Handle very large number of arguments (stress test).

```
public class LargeArgTest {
    public static void main(String[] args) {
        System.out.println("Received " + args.length + " arguments.");
        if (args.length > 0) {
            System.out.println("First: " + args[0]);
            System.out.println("Last: " + args[args.length - 1]);
        }
    }
}

// Run with: java LargeArgTest $(seq 1 100000)
```

Q29. Parse arguments as JVM-like options: `-Dkey=value -verbose`.

```
import java.util.*;

public class JVMArgParser {
    public static void main(String[] args) {
        Map<String, String> props = new HashMap<>();
        List<String> flags = new ArrayList<>();

        for (String arg : args) {
            if (arg.startsWith("-D")) {
                String[] pair = arg.substring(2).split("=", 2);
                props.put(pair[0], pair.length > 1 ? pair[1] : "true");
            } else if (arg.startsWith("-")) {
                flags.add(arg);
            }
        }

        System.out.println("Properties: " + props);
        System.out.println("Flags: " + flags);
    }
}
```

Q30. Simulate argument validation with custom exception.

```
class InvalidArgsException extends Exception {
    InvalidArgsException(String msg) { super(msg); }
}

public class ValidatedMain {
    public static void main(String[] args) {
        try {
            if (args.length == 0) throw new InvalidArgsException("No arguments provided");
            if (args.length > 10) throw new InvalidArgsException("Too many arguments");
            System.out.println("Valid input");
        } catch (InvalidArgsException e) {
            System.err.println("Error: " + e.getMessage());
            System.exit(1);
        }
    }
```

}

TOPIC 5: Identifiers, Keywords, Java Data Types & Operators

Easy MCQs (10)

1. Which of the following is a valid Java identifier?
   A) `123var`
   B) `my-var`
   C) `_value`
   D) `class`
   Answer: C) `_value`

2. How many primitive data types does Java have?
   A) 6
   B) 7
   C) 8
   D) 9
   Answer: C) 8

3. Which keyword is used to declare a constant?
   A) `const`
   B) `static`
   C) `final`
   D) `immutable`
   Answer: C) `final`

4. What is the default value of a `boolean` instance variable?
   A) `true`
   B) `false`
   C) `null`
   D) `0`
   Answer: B) `false`

5. Which operator is used for logical AND?
   A) `&`
   B) `&&`

C) `and`
D) `||`
Answer: B) `&&`

6. Which of the following is NOT a Java keyword?
A) `goto`
B) `const`
C) `include`
D) `native`
Answer: C) `include`

7. What is the size of a `long` in bits?
A) 32
B) 64
C) 16
D) 8
Answer: B) 64

8. Which data type is used to store a single Unicode character?
A) `String`
B) `byte`
C) `char`
D) `int`
Answer: C) `char`

9. What does `5 % 2` evaluate to?
A) 2
B) 2.5
C) 1
D) 0
Answer: C) 1

10. Identifiers in Java are case-______?
A) Insensitive
B) Sensitive
C) Neutral
D) Dependent
Answer: B) Sensitive

Medium MCQs (10)

1. What is the output of the following?

   int x = 10;
   System.out.println(x++ + ++x);

   A) 20
   B) 21
   C) 22
   D) 23
   Answer: C) 22
   (x++ = 10 (x=11), ++x = 12 (x=12) → 10 + 12 = 22)

2. Which expression evaluates to `true`?
   A) `(5 > 3) & (2 / 0 == 0)`
   B) `(5 > 3) && (2 / 0 == 0)`
   C) `(5 < 3) | (1 == 1)`
   D) `(5 < 3) || (1 == 1)`
   Answer: D) `(5 < 3) || (1 == 1)`

3. What is the result of `10 + "20" + 30`?
   A) `"102030"`
   B) `"3030"`
   C) `60`
   D) Compilation error
   Answer: A) `"102030"`

4. Which operator has the highest precedence?
   A) `+`
   B) `*`
   C) `()`
   D) `++`
   Answer: C) `()`

5. What does `~7` evaluate to?
   A) -8
   B) 8
   C) -7

D) 0
Answer: A) -8
(7 = 00000111 → ~7 = 11111000 = -8 in two's complement)

6. Which assignment is valid?
A) `byte b = 200;`
B) `int i = 10L;`
C) `long l = 10;`
D) `char c = -1;`
Answer: C) `long l = 10;`

7. What is the value of `x` after:

```
short x = 10;
x = x + 5;
```

A) 15
B) Compilation error
C) Runtime exception
D) 5
Answer: B) Compilation error
(`x + 5` is `int`; requires cast to `short`)

8. Which is a reserved keyword but unused?
A) `goto`
B) `const`
C) Both A and B
D) Neither
Answer: C) Both A and B

9. What is printed?
java
System.out.println(0.1 + 0.2 == 0.3);

A) `true`
B) `false`
C) `NaN`
D) Compilation error
Answer: B) `false`

10. Which operator is short-circuit?
   A) `&`
   B) `|`
   C) `^`
   D) `||`
   Answer: D) `||`

Hard MCQs (10)

1. What is the output?

```
int a = 5;
int b = a++ + ++a - --a + a--;
System.out.println(b);
```

   A) 10
   B) 11
   C) 12
   D) 13
   Answer: C) 12
   (Step: a=5 → a++=5(a=6) → ++a=7(a=7) → --a=6(a=6) → a--=6(a=5) → 5+7-6+6=12)

2. What is `(byte)256`?
   A) 256
   B) 0
   C) -1
   D) Compilation error
   Answer: B) 0
   (256 = 0x100 → lower 8 bits = 0x00 → 0)

3. Which expression causes integer overflow silently?
   A) `int x = Integer.MAX_VALUE + 1;`
   B) `byte b = 128;`
   C) `char c = 70000;`
   D) `float f = 1e40f;`
   Answer: A) `int x = Integer.MAX_VALUE + 1;`

4. What does `1 << 4` evaluate to?
   A) 4
   B) 8
   C) 16
   D) 32
   Answer: C) 16

5. After:

```
double d = Double.NaN;
System.out.println(d != d);
```

   Output?
   A) `true`
   B) `false`
   C) `NaN`
   D) Compilation error
   Answer: A) `true`

6. Which statement about `float` is FALSE?
   A) Uses 32 bits
   B) `1.0` is a `double` literal
   C) `1.0f` is a `float` literal
   D) More precise than `double`
   Answer: D) More precise than `double`

7. What is the value of `x`?

```
int x = 0xF & 010;
```

   A) 0
   B) 8
   C) 15
   D) 7
   Answer: A) 0
   (0xF = 15 = 1111, 010 = octal 8 = 1000 → 1111 & 1000 = 1000 = 8? Waitcorrection: 0xF = 15 (1111), 010 = 8 (1000) → 1111 & 1000 = 1000 = 8. But let's use a safer question:)
   Revised Q7:

java
int x = 0x7 & 010;

0x7 = 7 (0111), 010 = 8 (1000) → 0111 & 1000 = 0000 → Answer: A) 0

8. Which operator cannot be used with booleans?
A) `&&`
B) `||`
C) `&`
D) `+`
Answer: D) `+`

9. What is the output?

System.out.println((int)(char)(byte)-1);

A) -1
B) 65535
C) 255
D) 0
Answer: B) 65535
(-1 as byte = 0xFF → char = 0xFFFF → int = 65535)

10. Which is a valid identifier?
A) `$myVar`
B) `my Var`
C) `2ndPlace`
D) `class`
Answer: A) `$myVar`

Coding Problems

Easy Coding (10)

Q1. Swap two integers without a temporary variable.

```
public class Swap {
    public static void main(String[] args) {
        int a = 5, b = 10;
```

```
        a = a + b;
        b = a - b;
        a = a - b;
        System.out.println(a + " " + b);
    }
}
```

Q2. Check if a number is even or odd using bitwise operator.

```
public class EvenOdd {
    public static void main(String[] args) {
        int n = 7;
        System.out.println((n & 1) == 0 ? "Even" : "Odd");
    }
}
```

Q3. Print the ASCII value of a character.

```
public class ASCIIValue {
    public static void main(String[] args) {
        char c = 'A';
        System.out.println((int) c);
    }
}
```

Q4. Compute area of a circle (use `double` and `Math.PI`).

```
public class CircleArea {
    public static void main(String[] args) {
        double r = 5.0;
        double area = Math.PI * r * r;
        System.out.println(area);
    }
}
```

Q5. Convert Fahrenheit to Celsius.

```
public class TempConvert {
    public static void main(String[] args) {
```

```
        double f = 98.6;
        double c = (f - 32) * 5 / 9;
        System.out.println(c);
    }
}
```

Q6. Find maximum of two numbers using ternary operator.

```
public class MaxTwo {
    public static void main(String[] args) {
        int a = 10, b = 20;
        int max = (a > b) ? a : b;
        System.out.println(max);
    }
}
```

Q7. Check if a character is uppercase.

```
public class UpperCheck {
    public static void main(String[] args) {
        char c = 'A';
        System.out.println(c >= 'A' && c <= 'Z');
    }
}
```

Q8. Compute simple interest: `P * R * T / 100`.

```
public class SimpleInterest {
    public static void main(String[] args) {
        double p = 1000, r = 5, t = 2;
        double si = p * r * t / 100;
        System.out.println(si);
    }
}
```

Q9. Print the size of each primitive type in bytes.

```
public class TypeSizes {
    public static void main(String[] args) {
```

```
        System.out.println("byte: " + Byte.SIZE / 8);
        System.out.println("short: " + Short.SIZE / 8);
        System.out.println("int: " + Integer.SIZE / 8);
        System.out.println("long: " + Long.SIZE / 8);
        System.out.println("float: " + Float.SIZE / 8);
        System.out.println("double: " + Double.SIZE / 8);
        System.out.println("char: " + Character.SIZE / 8);
        // boolean size is not defined
    }
}
```

Q10. Check if a number is divisible by 5.

```
public class DivBy5 {
    public static void main(String[] args) {
        int n = 25;
        System.out.println(n % 5 == 0);
    }
}
```

Medium Coding (10)

Q11. Reverse the bits of a byte.

```
public class BitReverse {
    public static byte reverseBits(byte b) {
        byte result = 0;
        for (int i = 0; i < 8; i++) {
            result = (byte)((result << 1) | (b & 1));
            b = (byte)(b >> 1);
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println(reverseBits((byte)0b11001001)); // Example
    }
}
```

Q12. Check if a number is a power of two.

```
public class PowerOfTwo {
    public static void main(String[] args) {
        int n = 16;
        System.out.println(n > 0 && (n & (n - 1)) == 0);
    }
}
```

Q13. Compute compound interest.

```
public class CompoundInterest {
    public static void main(String[] args) {
        double p = 1000, r = 5, t = 2;
        double ci = p * Math.pow(1 + r / 100, t) - p;
        System.out.println(ci);
    }
}
```

Q14. Find the number of set bits in an integer.

```
public class SetBits {
    public static void main(String[] args) {
        int n = 15; // 1111
        int count = 0;
        while (n != 0) {
            count += n & 1;
            n >>>= 1;
        }
        System.out.println(count);
    }
}
```

Q15. Swap two numbers using XOR.

```
public class XORSwap {
    public static void main(String[] args) {
        int a = 5, b = 10;
        a ^= b;
```

```
        b ^= a;
        a ^= b;
        System.out.println(a + " " + b);
    }
}
```

Q16. Check if two numbers have opposite signs.

```
public class OppositeSigns {
    public static void main(String[] args) {
        int a = -5, b = 10;
        System.out.println((a ^ b) < 0);
    }
}
```

Q17. Compute average of three numbers without overflow.

```
public class SafeAverage {
    public static void main(String[] args) {
        int a = 2000000000, b = 2000000000, c = 2000000000;
        double avg = a / 3.0 + b / 3.0 + c / 3.0;
        System.out.println(avg);
    }
}
```

Q18. Convert lowercase to uppercase without `toUpperCase()`.

```
public class ToUpper {
    public static void main(String[] args) {
        char c = 'a';
        if (c >= 'a' && c <= 'z') {
            c = (char)(c - 'a' + 'A');
        }
        System.out.println(c);
    }
}
```

Q19. Check if a year is leap year.

```
public class LeapYear {
    public static void main(String[] args) {
        int year = 2024;
        boolean leap = (year % 4 == 0) && (year % 100 != 0 || year % 400 == 0);
        System.out.println(leap);
    }
}
```

Q20. Compute `a^b` using only bitwise and arithmetic ops (for small b).

```
public class Power {
    public static void main(String[] args) {
        int a = 2, b = 10;
        int result = 1;
        for (int i = 0; i < b; i++) {
            result *= a;
        }
        System.out.println(result);
    }
}
```

Hard Coding (10)

Q21. Multiply two integers without using `*` operator.

```
public class MultiplyWithoutStar {
    public static int multiply(int a, int b) {
        if (b == 0) return 0;
        if (b > 0) return a + multiply(a, b - 1);
        return -multiply(a, -b);
    }

    public static void main(String[] args) {
        System.out.println(multiply(7, 6)); // 42
    }
}
```

Q22. Divide two integers without using `/`, `%`, or `*`.

```
public class DivideWithoutOps {
    public static int divide(int dividend, int divisor) {
        if (divisor == 0) throw new ArithmeticException();
        boolean negative = (dividend < 0) ^ (divisor < 0);
        long dvd = Math.abs((long) dividend);
        long dvs = Math.abs((long) divisor);
        long quotient = 0;

        while (dvd >= dvs) {
            long temp = dvs, multiple = 1;
            while (dvd >= (temp << 1)) {
                temp <<= 1;
                multiple <<= 1;
            }
            dvd -= temp;
            quotient += multiple;
        }
        quotient = negative ? -quotient : quotient;
        return (int) Math.max(Math.min(quotient, Integer.MAX_VALUE),
Integer.MIN_VALUE);
    }

    public static void main(String[] args) {
        System.out.println(divide(10, 3)); // 3
    }
}
```

Q23. Find the missing number in an array of 1 to n.

```
public class MissingNumber {
    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5, 6}; // n=6, missing=3
        int n = arr.length + 1;
        int expected = n * (n + 1) / 2;
        int actual = 0;
        for (int x : arr) actual += x;
        System.out.println(expected - actual);
    }
}
```

Q24. Compute factorial using only bitwise and addition (no `*`).

```
public class BitwiseFactorial {
    static int multiply(int a, int b) {
        int res = 0;
        while (b > 0) {
            if ((b & 1) == 1) res += a;
            a <<= 1;
            b >>= 1;
        }
        return res;
    }

    public static void main(String[] args) {
        int n = 5, fact = 1;
        for (int i = 2; i <= n; i++) {
            fact = multiply(fact, i);
        }
        System.out.println(fact); // 120
    }
}
```

Q25. Check if a number is palindrome without converting to string.

```
public class NumericPalindrome {
    public static void main(String[] args) {
        int n = 12321, original = n, reversed = 0;
        while (n > 0) {
            reversed = reversed * 10 + n % 10;
            n /= 10;
        }
        System.out.println(original == reversed);
    }
}
```

Q26. Find the single number in an array where every other appears twice.

```
public class SingleNumber {
```

```
    public static void main(String[] args) {
        int[] arr = {4, 1, 2, 1, 2};
        int result = 0;
        for (int x : arr) result ^= x;
        System.out.println(result); // 4
    }
}
```

Q27. Compute square root using binary search (integer part).

```
public class Sqrt {
    public static void main(String[] args) {
        int n = 20;
        long low = 0, high = n;
        while (low <= high) {
            long mid = (low + high) / 2;
            if (mid * mid <= n) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        System.out.println(high); // 4
    }
}
```

Q28. Reverse an integer (handle overflow).

```
public class ReverseInteger {
    public static void main(String[] args) {
        int x = 123;
        long rev = 0;
        while (x != 0) {
            rev = rev * 10 + x % 10;
            x /= 10;
        }
        if (rev > Integer.MAX_VALUE || rev < Integer.MIN_VALUE) {
            System.out.println(0);
        } else {
```

```
            System.out.println(rev);
        }
    }
}
```

Q29. Find the greatest common divisor (GCD) using Euclidean algorithm.

```
public class GCD {
    public static int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    public static void main(String[] args) {
        System.out.println(gcd(48, 18)); // 6
    }
}
```

Q30. Implement fast exponentiation (a^b mod m).

```
public class FastExponentiation {
    public static long power(long a, long b, long m) {
        long result = 1;
        a %= m;
        while (b > 0) {
            if ((b & 1) == 1) result = (result * a) % m;
            a = (a * a) % m;
            b >>= 1;
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println(power(2, 10, 1000)); // 24
    }
```

}

TOPIC 6: Control Statements in Java

Easy MCQs (10)

1. Which statement is used to make a decision based on a condition?
   A) `for`
   B) `while`
   C) `if`
   D) `return`
   Answer: C) `if`

2. How many times does the following loop run?

```
for (int i = 0; i < 5; i++) { }
```

   A) 4
   B) 5
   C) 6
   D) Infinite
   Answer: B) 5

3. What does `break` do inside a loop?
   A) Skips one iteration
   B) Exits the loop immediately
   C) Returns a value
   D) Pauses execution
   Answer: B) Exits the loop immediately

4. Which loop guarantees at least one execution?
   A) `for`
   B) `while`
   C) `do-while`
   D) `foreach`
   Answer: C) `do-while`

5. What is the output?

```
if (true) System.out.print("A");
else System.out.print("B");
```

A) A
B) B
C) AB
D) Nothing
Answer: A) A

6. Which keyword skips the rest of the current loop iteration?
A) `break`
B) `exit`
C) `continue`
D) `skip`
Answer: C) `continue`

7. What is the purpose of `return` in a `void` method?
A) Returns 0
B) Exits the method
C) Throws an exception
D) Restarts the method
Answer: B) Exits the method

8. Which is a valid `switch` expression type?
A) `float`
B) `boolean`
C) `String`
D) `double`
Answer: C) `String` (since Java 7)

9. What is the output?

```
int x = 3;
switch (x) {
    case 1: System.out.print("1");
    case 3: System.out.print("3");
    default: System.out.print("D");
}
```

A) 3
B) 3D
C) D
D) 13D
Answer: B) 3D *(no `break` → fall-through)*

10. Which loop is best for iterating over an array?
A) `while`
B) `do-while`
C) `for`
D) All are equal
Answer: C) `for`

Medium MCQs (10)

1. What is the output?
```java
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++) {
        if (i == 1) break;
        System.out.print(i + "" + j + " ");
    }
}
```

A) `00 01 20 21`
B) `00 01 10 20 21`
C) `00 01 20`
D) `00 01`
Answer: A) `00 01 20 21`

2. What does this print?
```java
int i = 0;
while (i++ < 3) {
    continue;
}
System.out.println(i);
```

A) 3

B) 4
C) 2
D) Infinite loop
Answer: B) 4

3. Which statement about `switch` is TRUE?
A) `case` labels can be variables
B) `default` must be last
C) `switch` supports `long`
D) `case` values must be compile-time constants
Answer: D) `case` values must be compile-time constants

4. What is the output?
java
```
for (int i = 0; i < 5; i++) {
    if (i % 2 == 0) continue;
    System.out.print(i + " ");
}
```

A) `0 2 4`
B) `1 3`
C) `1 3 5`
D) `0 1 2 3 4`
Answer: B) `1 3`

5. How to exit an outer loop from inside a nested loop?
A) Use `break` twice
B) Use labeled `break`
C) Use `return`
D) Both B and C
Answer: D) Both B and C

6. What is printed?
java
```
int x = 5;
if (x > 3) {
    if (x < 10) System.out.print("A");
} else System.out.print("B");
```

A) A
B) B
C) AB
D) Nothing
Answer: A) A

7. Which loop condition causes infinite loop?
A) `for (;;)`
B) `while (true)`
C) `do {} while (true);`
D) All of the above
Answer: D) All of the above

8. What is the value of `sum`?
```java
int sum = 0;
for (int i = 1; i <= 4; i++) {
    if (i == 3) continue;
    sum += i;
}
```

A) 7
B) 10
C) 6
D) 3
Answer: A) 7 *(1+2+4)*

9. In a `switch`, what happens if no `case` matches and no `default`?
A) Compilation error
B) Runtime exception
C) Nothing executes
D) Executes first `case`
Answer: C) Nothing executes

10. What does `return` do in a non-void method?
A) Exits and returns a value
B) Only exits
C) Skips to next method
D) Restarts JVM

Answer: A) Exits and returns a value

Hard MCQs (10)

1. What is the output?
java
```
outer: for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        if (i == j) continue outer;
        System.out.print(i + "" + j + " ");
    }
}
```

A) `01 10`
B) `01`
C) `10`
D) Nothing
Answer: C) `10`
*(i=0,j=0 → continue outer; i=0,j=1 → print "01"? Waitlet's trace:
i=0: j=0 → i==j → continue outer → skip j=1
i=1: j=0 → i!=j → print "10"; j=1 → i==j → continue outer
So only "10" → Answer: C)*

2. What is printed?
java
```
int i = 0;
do {
    System.out.print(i + " ");
} while (i++ < 2);
```

A) `0 1 2`
B) `0 1`
C) `1 2`
D) `0 1 2 3`
Answer: A) `0 1 2`

3. Which code snippet has unreachable statement?
A) `if (true) return; System.out.println("X");`
B) `while (false) System.out.println("X");`

C) `if (false) System.out.println("X");`
D) Both A and B
Answer: D) Both A and B

4. What is the output?
java
```
for (int i = 0; i < 3; i++) {
    System.out.print(i + " ");
    if (i == 1) return;
}
```

A) `0 1`
B) `0 1 2`
C) `0`
D) Compilation error
Answer: A) `0 1`

5. In which scenario is `continue` equivalent to `break`?
A) Last statement in loop body
B) First statement in loop body
C) Never
D) In `do-while` only
Answer: A) Last statement in loop body

6. What is the output?
java
```
switch (2) {
    case 1: System.out.print("1 ");
    case 2: System.out.print("2 ");
    case 3: System.out.print("3 ");
    default: System.out.print("D ");
}
```

A) `2`
B) `2 3 D`
C) `1 2 3 D`
D) `D`
Answer: B) `2 3 D`

7. Which loop is most efficient for known iterations?
   A) `while`
   B) `do-while`
   C) `for`
   D) All same
   Answer: C) `for` *(compiler optimizes better)*

8. What happens if `break` is used outside loop/switch?
   A) Compilation error
   B) Runtime exception
   C) Exits method
   D) Ignored
   Answer: A) Compilation error

9. What is the output?
```java
int x = 0;
while (x < 3) {
    x++;
    if (x == 2) continue;
    System.out.print(x + " ");
}
```

   A) `1 3`
   B) `1 2 3`
   C) `3`
   D) `1`
   Answer: A) `1 3`

10. Which statement about labeled statements is FALSE?
    A) Labels can be applied to any block
    B) `break label;` exits the labeled block
    C) Labels improve performance
    D) Labels must be identifiers
    Answer: C) Labels improve performance

Coding Problems

Easy Coding (10)

Q1. Print numbers 1 to 10 using a `for` loop.

```
public class Print1To10 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

Q2. Print even numbers from 1 to 20 using `continue`.

```
public class EvenWithContinue {
    public static void main(String[] args) {
        for (int i = 1; i <= 20; i++) {
            if (i % 2 != 0) continue;
            System.out.println(i);
        }
    }
}
```

Q3. Find the largest of three numbers using `if-else`.

```
public class MaxOfThree {
    public static void main(String[] args) {
        int a = 10, b = 20, c = 15;
        int max = a;
        if (b > max) max = b;
        if (c > max) max = c;
        System.out.println(max);
    }
}
```

Q4. Print multiplication table of 5.

```
public class TableOf5 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
```

```
            System.out.println("5 * " + i + " = " + (5 * i));
        }
    }
}
```

Q5. Check if a number is prime using a loop.

```
public class PrimeCheck {
    public static void main(String[] args) {
        int n = 17;
        boolean isPrime = true;
        for (int i = 2; i < n; i++) {
            if (n % i == 0) {
                isPrime = false;
                break;
            }
        }
        System.out.println(isPrime);
    }
}
```

Q6. Print numbers 10 to 1 in reverse using `while`.

```
public class ReverseCount {
    public static void main(String[] args) {
        int i = 10;
        while (i >= 1) {
            System.out.println(i--);
        }
    }
}
```

Q7. Use `switch` to print day name for 1=Monday, ..., 7=Sunday.

```
public class DayName {
    public static void main(String[] args) {
        int day = 3;
        switch (day) {
            case 1: System.out.println("Monday"); break;
```

```
            case 2: System.out.println("Tuesday"); break;
            case 3: System.out.println("Wednesday"); break;
            case 4: System.out.println("Thursday"); break;
            case 5: System.out.println("Friday"); break;
            case 6: System.out.println("Saturday"); break;
            case 7: System.out.println("Sunday"); break;
            default: System.out.println("Invalid");
        }
    }
}
```

Q8. Sum first 100 natural numbers using a loop.

```
public class Sum100 {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 1; i <= 100; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

Q9. Print "Hello" 5 times using `do-while`.

```
public class DoWhileHello {
    public static void main(String[] args) {
        int i = 0;
        do {
            System.out.println("Hello");
            i++;
        } while (i < 5);
    }
}
```

Q10. Exit a loop when the user enters 0 (simulate with fixed input).

```
public class ExitOnZero {
    public static void main(String[] args) {
```

```
        int[] inputs = {5, 3, 0, 7};
        for (int x : inputs) {
            if (x == 0) break;
            System.out.println(x);
        }
    }
}
```

Medium Coding (10)

Q11. Print Floyd's triangle (1, 2 3, 4 5 6, ...).

```
public class FloydsTriangle {
    public static void main(String[] args) {
        int n = 4, num = 1;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(num++ + " ");
            }
            System.out.println();
        }
    }
}
```

Q12. Find GCD of two numbers using the Euclidean algorithm with `while`.

```
public class GCDEuclid {
    public static void main(String[] args) {
        int a = 48, b = 18;
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        System.out.println(a);
    }
}
```

Q13. Print all Armstrong numbers between 1 and 1000.

```
public class ArmstrongNumbers {
    public static void main(String[] args) {
        for (int num = 1; num <= 1000; num++) {
            int sum = 0, temp = num, digits = String.valueOf(num).length();
            while (temp > 0) {
                int digit = temp % 10;
                sum += Math.pow(digit, digits);
                temp /= 10;
            }
            if (sum == num) System.out.println(num);
        }
    }
}
```

Q14. Simulate a menu-driven calculator using `switch` and `do-while`.

```
public class MenuCalculator {
    public static void main(String[] args) {
        // Simulated input
        char op = '+';
        int a = 10, b = 5;
        switch (op) {
            case '+': System.out.println(a + b); break;
            case '-': System.out.println(a - b); break;
            case '*': System.out.println(a * b); break;
            case '/': System.out.println(a / b); break;
            default: System.out.println("Invalid");
        }
    }
}
```

Q15. Print numbers 1 to 100, but for multiples of 3 print "Fizz", 5 print "Buzz".

```
public class FizzBuzz {
    public static void main(String[] args) {
        for (int i = 1; i <= 100; i++) {
            if (i % 15 == 0) System.out.println("FizzBuzz");
```

```
            else if (i % 3 == 0) System.out.println("Fizz");
            else if (i % 5 == 0) System.out.println("Buzz");
            else System.out.println(i);
        }
    }
}
```

Q16. Find the first duplicate in an array using nested loops.

```
public class FirstDuplicate {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 4};
        outer: for (int i = 0; i < arr.length; i++) {
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[i] == arr[j]) {
                    System.out.println(arr[i]);
                    break outer;
                }
            }
        }
    }
}
```

Q17. Print all factors of a number.

```
public class Factors {
    public static void main(String[] args) {
        int n = 12;
        for (int i = 1; i <= n; i++) {
            if (n % i == 0) {
                System.out.println(i);
            }
        }
    }
}
```

Q18. Check if a number is perfect (sum of divisors = number).

```
public class PerfectNumber {
```

```
    public static void main(String[] args) {
        int n = 28, sum = 0;
        for (int i = 1; i < n; i++) {
            if (n % i == 0) sum += i;
        }
        System.out.println(sum == n);
    }
}
```

Q19. Print ASCII values of all uppercase letters.

```
public class ASCIILetters {
    public static void main(String[] args) {
        for (char c = 'A'; c <= 'Z'; c++) {
            System.out.println(c + " = " + (int) c);
        }
    }
}
```

Q20. Simulate a login attempt with max 3 tries.

```
public class LoginSim {
    public static void main(String[] args) {
        String correct = "secret";
        String[] attempts = {"wrong", "oops", "secret"};
        boolean success = false;
        for (int i = 0; i < attempts.length; i++) {
            if (attempts[i].equals(correct)) {
                System.out.println("Login successful");
                success = true;
                break;
            }
        }
        if (!success) System.out.println("Account locked");
    }
}
```

Hard Coding (10)

Q21. Generate all permutations of a string using recursion and control flow.

```
public class StringPermutations {
    static void permute(String str, String ans) {
        if (str.length() == 0) {
            System.out.println(ans);
            return;
        }
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            String ros = str.substring(0, i) + str.substring(i + 1);
            permute(ros, ans + ch);
        }
    }

    public static void main(String[] args) {
        permute("ABC", "");
    }
}
```

Q22. Implement binary search using `while` (no recursion).

```
public class BinarySearchIterative {
    public static void main(String[] args) {
        int[] arr = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
        int target = 23;
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == target) {
                System.out.println("Found at " + mid);
                return;
            } else if (arr[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        System.out.println("Not found");
```

```
    }
}
```

Q23. Print all prime numbers up to N using Sieve of Eratosthenes.

```
public class SieveOfEratosthenes {
    public static void main(String[] args) {
        int n = 30;
        boolean[] prime = new boolean[n + 1];
        for (int i = 2; i <= n; i++) prime[i] = true;

        for (int p = 2; p * p <= n; p++) {
            if (prime[p]) {
                for (int i = p * p; i <= n; i += p) {
                    prime[i] = false;
                }
            }
        }

        for (int i = 2; i <= n; i++) {
            if (prime[i]) System.out.print(i + " ");
        }
    }
}
```

Q24. Solve the Tower of Hanoi problem (n=3) with step-by-step moves.

```
public class TowerOfHanoi {
    static void move(int n, char from, char to, char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + from + " to " + to);
            return;
        }
        move(n - 1, from, aux, to);
        System.out.println("Move disk " + n + " from " + from + " to " + to);
        move(n - 1, aux, to, from);
    }

    public static void main(String[] args) {
        move(3, 'A', 'C', 'B');
    }
```

```
}
```

Q25. Find the longest common prefix among an array of strings.

```
public class LongestCommonPrefix {
    public static void main(String[] args) {
        String[] words = {"flower", "flow", "flight"};
        if (words.length == 0) { System.out.println(""); return; }
        String prefix = words[0];
        for (int i = 1; i < words.length; i++) {
            while (!words[i].startsWith(prefix)) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) {
                    System.out.println("");
                    return;
                }
            }
        }
        System.out.println(prefix);
    }
}
```

Q26. Simulate a state machine for parsing integers (with signs).

```
public class IntegerParser {
    public static void main(String[] args) {
        String input = "-123";
        int i = 0, sign = 1, result = 0;

        // Skip whitespace (none here)
        if (input.charAt(i) == '-') {
            sign = -1;
            i++;
        } else if (input.charAt(i) == '+') {
            i++;
        }

        while (i < input.length() && Character.isDigit(input.charAt(i))) {
            int digit = input.charAt(i) - '0';
            // Check overflow
            if (result > (Integer.MAX_VALUE - digit) / 10) {
```

```
                result = sign == 1 ? Integer.MAX_VALUE : Integer.MIN_VALUE;
                break;
            }
            result = result * 10 + digit;
            i++;
        }

        System.out.println(sign * result);
    }
}
```

Q27. Print all combinations of balanced parentheses for n=3.

```
public class BalancedParentheses {
    static void generate(String current, int open, int close, int max) {
        if (current.length() == max * 2) {
            System.out.println(current);
            return;
        }
        if (open < max) generate(current + "(", open + 1, close, max);
        if (close < open) generate(current + ")", open, close + 1, max);
    }

    public static void main(String[] args) {
        generate("", 0, 0, 3);
    }
}
```

Q28. Implement a simple finite automaton for accepting strings with even 0s and 1s.

java

```
public class EvenZerosOnes {
    public static void main(String[] args) {
        String s = "0011";
        boolean even0 = true, even1 = true;
        for (char c : s.toCharArray()) {
            if (c == '0') even0 = !even0;
            else if (c == '1') even1 = !even1;
        }
        System.out.println(even0 && even1 ? "Accepted" : "Rejected");
    }
```

}

Q29. Find the maximum subarray sum (Kadane's algorithm).

```java
public class KadaneAlgorithm {
    public static void main(String[] args) {
        int[] arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        int maxSoFar = arr[0], maxEndingHere = arr[0];
        for (int i = 1; i < arr.length; i++) {
            maxEndingHere = Math.max(arr[i], maxEndingHere + arr[i]);
            maxSoFar = Math.max(maxSoFar, maxEndingHere);
        }
        System.out.println(maxSoFar); // 6
    }
}
```

Q30. Simulate a producer-consumer using control flow (simplified with flags).

```java
public class ProducerConsumerSim {
    public static void main(String[] args) throws InterruptedException {
        boolean hasItem = false;
        int item = 0;

        // Producer
        for (int i = 1; i <= 3; i++) {
            while (hasItem) Thread.sleep(10); // busy wait (simplified)
            item = i;
            hasItem = true;
            System.out.println("Produced: " + item);
        }

        // Consumer
        for (int i = 0; i < 3; i++) {
            while (!hasItem) Thread.sleep(10);
            System.out.println("Consumed: " + item);
            hasItem = false;
        }
    }
}
```

TOPIC 7: Pattern Problems

Easy MCQs (10)

1. What does this code print?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

A)

```
*
**
***
```

B)

```
***
***
***
```

C)

```
*
*
*
```

D) ``
Answer: A)

2. How many stars are printed in total?

```
for (int i = 1; i <= 4; i++)
```

```
    for (int j = 1; j <= 3; j++)
        System.out.print("*");
```

A) 7
B) 12
C) 4
D) 3
Answer: B) 12

3. Which pattern is produced by:

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 5; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

A) Right triangle
B) Rectangle
C) Pyramid
D) Diamond
Answer: B) Rectangle

4. What is printed?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 3; j >= i; j--) {
        System.out.print("*");
    }
    System.out.println();
}
```

A)

```
***
**
*
```

B)

```
*
**
***
```

C)

```
**
**
**
```

D) None
Answer: A)

5. Which loop prints numbers 1 to 3 in a column?
A) `for (int i=1; i<=3; i++) System.out.println(i);`
B) `for (int i=1; i<=3; i++) System.out.print(i);`
C) Both
D) Neither
Answer: A)

6. What is the output?

```
for (int i = 1; i <= 2; i++) {
    for (int j = 1; j <= 2; j++) {
        System.out.print(i);
    }
    System.out.println();
}
```

A)

```
11
22
```

B)

```
12
```

12

C) `1122`
D) `1212`
Answer: A)

7. Which pattern uses `j <= i` in inner loop?
A) Rectangle
B) Right-angled triangle
C) Hollow square
D) Diamond
Answer: B) Right-angled triangle

8. What is printed?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(j);
    }
    System.out.println();
}
```

A)

1
12
123

B)

1
22
333

C)

1
21
321

D) None
Answer: A)

9. How to print a single row of 5 stars?
A) `for (int i=0; i<5; i++) System.out.print("*");`
B) `System.out.println("*");`
C) Both
D) Neither
Answer: C) Both

10. What is the output?

```
for (int i = 3; i >= 1; i--) {
    for (int j = 1; j <= i; j++) {
        System.out.print(j);
    }
    System.out.println();
}
```

A)

```
123
12
1
```

B)

```
321
21
1
```

C)

```
1
12
123
```

D) None

Answer: A)

Medium MCQs (10)

1. What does this print?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i);
    }
    System.out.println();
}
```

A)

1
22
333

B)

1
12
123

C)

1
21
321

D) None
Answer: A)

2. Which code prints:

1
21
321

A)

```
for (int i = 1; i <= 3; i++) {
    for (int j = i; j >= 1; j--) {
        System.out.print(j);
    }
    System.out.println();
}
```

B)

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i - j + 1);
    }
    System.out.println();
}
```

C) Both A and B
D) Neither
Answer: C) Both A and B

3. What is the output?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3 - i; j++) {
        System.out.print(" ");
    }
    for (int k = 1; k <= i; k++) {
        System.out.print("*");
    }
    System.out.println();
}
```

A) Left-aligned triangle
B) Right-aligned triangle
C) Pyramid
D) Diamond

Answer: B) Right-aligned triangle

4. How many spaces are printed in total?

```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= 4 - i; j++) {
        System.out.print(" ");
    }
}
```

A) 6
B) 10
C) 4
D) 0
Answer: A) 6 *(3+2+1+0)*

5. Which pattern is this?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(j);
    }
    for (int k = i - 1; k >= 1; k--) {
        System.out.print(k);
    }
    System.out.println();
}
```

A)

```
1
121
12321
```

B)

```
1
212
32123
```

C) Palindromic triangle
D) Both A and C
Answer: D) Both A and C

6. What is printed?

```
int num = 1;
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(num++ + " ");
    }
    System.out.println();
}
```

A)

1
2 3
4 5 6

B)

1
1 2
1 2 3

C)

1
3 2
6 5 4

D) None
Answer: A)

7. Which loop prints a hollow rectangle?
A) Print `*` only on borders
B) Print `*` everywhere

C) Print only corners
D) Print alternating `*` and space
Answer: A) Print `*` only on borders

8. What is the output?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 2 * i - 1; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

A)

```
*
**
***
```

B)

```
***
**
*
```

C)

```
***
***
***
```

D) None
Answer: A)

9. How to print a pyramid of height 3?
A)

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3 - i; j++) System.out.print(" ");
```

```
    for (int k = 1; k <= 2 * i - 1; k++) System.out.print("*");
    System.out.println();
}
```

B) Print increasing stars with leading spaces
C) Both A and B
D) Neither
Answer: C) Both A and B

10. What is printed?

```
for (int i = 1; i <= 2; i++) {
    for (int j = 1; j <= 3; j++) {
        System.out.print((i + j) % 2 == 0 ? "*" : " ");
    }
    System.out.println();
}
```

A)
```
* * *
* *
 *
```

B) Checkerboard pattern
C) All stars
D) All spaces
Answer: B) Checkerboard pattern

Hard MCQs (10)

1. What does this print?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3 - i; j++) System.out.print(" ");
    for (int k = 1; k <= 2 * i - 1; k++) System.out.print(k <= i ? k : 2 * i - k);
    System.out.println();
}
```

A)

```
  1
 121
12321
```

B)

```
  1
 232
34543
```

C) Number pyramid
D) Both A and C
Answer: D) Both A and C

2. Which pattern is generated by:

```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= i; j++) System.out.print(j);
    for (int j = i - 1; j >= 1; j--) System.out.print(j);
    System.out.println();
}
```

A) Palindromic number triangle
B) Floyd's triangle
C) Pascal's triangle
D) None
Answer: A) Palindromic number triangle

3. What is the output?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) System.out.print((char)('A' + j - 1));
    System.out.println();
}
```

A)

A

AB
ABC

B)

A
BB
CCC

C)

A
BA
CBA

D) None
Answer: A)

4. How many total characters (stars + spaces) are printed?

```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= 4 - i; j++) System.out.print(" ");
    for (int k = 1; k <= 2 * i - 1; k++) System.out.print("*");
    System.out.println();
}
```

A) 16
B) 20
C) 24
D) 28
Answer: A) 16
*(Row1: 3 spaces + 1 star = 4; Row2: 2+3=5; Row3:1+5=6; Row4:0+7=7 → 4+5+6+7=22? Waitrecompute:
Height=4 →
Row1: 3 spaces, 1 star → 4
Row2: 2 spaces, 3 stars → 5
Row3: 1 space, 5 stars → 6
Row4: 0 spaces, 7 stars → 7
Total = 4+5+6+7 = 22. But options don't have 22. Revised question:)*

Revised Q4: For height=3:
Row1: 2+1=3, Row2:1+3=4, Row3:0+5=5 → total=12 → Answer: A) 12

5. Which code prints a diamond of height 5?
A) Upper pyramid + inverted pyramid (without middle repeat)
B) Two pyramids
C) Spiral loop
D) None
Answer: A) Upper pyramid + inverted pyramid (without middle repeat)

6. What is printed?

```
for (int i = 1; i <= 3; i++) {
   for (int j = 1; j <= 3 - i; j++) System.out.print(" ");
   for (int k = 1; k <= i; k++) System.out.print("* ");
   System.out.println();
}
```

A)

```
  *
 * *
* * *
```

B) Solid pyramid
C) Right triangle
D) None
Answer: A)

7. Which pattern uses `2*i-1` for inner loop count?
A) Rectangle
B) Pyramid
C) Hollow square
D) Diamond
Answer: B) Pyramid

8. What is the output?

```
for (int i = 1; i <= 3; i++) {
```

```
    for (int j = 1; j <= i; j++) System.out.print(i % 2 == 1 ? "*" : " ");
    System.out.println();
}
```

A)

```
*
**
***
```

B) Alternating solid and empty rows
C) Checkerboard
D) None
Answer: B) Alternating solid and empty rows

9. How to print a butterfly pattern?
A) Left stars + spaces + right stars
B) Two triangles facing each other
C) Both A and B
D) Spiral
Answer: C) Both A and B

10. What is the time complexity of printing a pyramid of height n?
A) $O(n)$
B) $O(n \log n)$
C) $O(n^2)$
D) $O(2^n)$
Answer: C) $O(n^2)$

Coding Problems

Easy Coding (10)

Q1. Print a right-angled triangle of stars (height = 5).

```
public class RightTriangle {
    public static void main(String[] args) {
        int n = 5;
        for (int i = 1; i <= n; i++) {
```

```
        for (int j = 1; j <= i; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
  }
}
```

Q2. Print an inverted right-angled triangle (base = 5).

```
public class InvertedTriangle {
  public static void main(String[] args) {
    int n = 5;
    for (int i = n; i >= 1; i--) {
        for (int j = 1; j <= i; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
  }
}
```

Q3. Print a rectangle of 3 rows and 6 columns.

```
public class Rectangle {
  public static void main(String[] args) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 6; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
  }
}
```

Q4. Print numbers 1 to 4 in a right triangle.

```
public class NumberTriangle {
  public static void main(String[] args) {
```

```
        for (int i = 1; i <= 4; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(j);
            }
            System.out.println();
        }
    }
}
```

Q5. Print a triangle with row numbers (1, 22, 333).

```
public class RepeatedNumbers {
    public static void main(String[] args) {
        for (int i = 1; i <= 4; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(i);
            }
            System.out.println();
        }
    }
}
```

Q6. Print a single line of 8 stars.

```
public class SingleLine {
    public static void main(String[] args) {
        for (int i = 0; i < 8; i++) {
            System.out.print("*");
        }
    }
}
```

Q7. Print a hollow rectangle (4x5).

```
public class HollowRectangle {
    public static void main(String[] args) {
        int rows = 4, cols = 5;
        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= cols; j++) {
```

```
            if (i == 1 || i == rows || j == 1 || j == cols)
                System.out.print("*");
            else
                System.out.print(" ");
        }
        System.out.println();
    }
  }
}
```

Q8. Print alphabets A to E in a column.

```
public class AlphabetColumn {
    public static void main(String[] args) {
        for (char c = 'A'; c <= 'E'; c++) {
            System.out.println(c);
        }
    }
}
```

Q9. Print a right triangle of alphabets (A, AB, ABC).

```
public class AlphabetTriangle {
    public static void main(String[] args) {
        int n = 4;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print((char)('A' + j - 1));
            }
            System.out.println();
        }
    }
}
```

Q10. Print a triangle of odd numbers (1, 3 5, 7 9 11).

```
public class OddNumberTriangle {
    public static void main(String[] args) {
        int n = 3, num = 1;
        for (int i = 1; i <= n; i++) {
```

```
            for (int j = 1; j <= i; j++) {
                System.out.print(num + " ");
                num += 2;
            }
            System.out.println();
        }
    }
}
```

Medium Coding (10)

Q11. Print a pyramid of stars (height = 4).

```
public class StarPyramid {
    public static void main(String[] args) {
        int n = 4;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int k = 1; k <= 2 * i - 1; k++) System.out.print("*");
            System.out.println();
        }
    }
}
```

Q12. Print a diamond pattern (height = 5).

```
public class DiamondPattern {
    public static void main(String[] args) {
        int n = 5;
        // Upper half including middle
        for (int i = 1; i <= n; i += 2) {
            for (int j = 0; j < (n - i) / 2; j++) System.out.print(" ");
            for (int j = 0; j < i; j++) System.out.print("*");
            System.out.println();
        }
        // Lower half
        for (int i = n - 2; i >= 1; i -= 2) {
            for (int j = 0; j < (n - i) / 2; j++) System.out.print(" ");
            for (int j = 0; j < i; j++) System.out.print("*");
            System.out.println();
```

```
        }
    }
}
```

Q13. Print Pascal's Triangle (n = 5).

```
public class PascalsTriangle {
    public static void main(String[] args) {
        int n = 5;
        for (int i = 0; i < n; i++) {
            int num = 1;
            for (int j = 0; j <= i; j++) {
                System.out.print(num + " ");
                num = num * (i - j) / (j + 1);
            }
            System.out.println();
        }
    }
}
```

Q14. Print a Floyd's Triangle (1, 2 3, 4 5 6).

```
public class FloydsTriangle {
    public static void main(String[] args) {
        int n = 4, num = 1;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(num++ + " ");
            }
            System.out.println();
        }
    }
}
```

Q15. Print a palindromic number triangle (1, 121, 12321).

```
public class PalindromicTriangle {
    public static void main(String[] args) {
        int n = 4;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print(j);
```

```
        for (int j = i - 1; j >= 1; j--) System.out.print(j);
        System.out.println();
      }
    }
}
```

Q16. Print a butterfly pattern (n = 3).

```
public class ButterflyPattern {
    public static void main(String[] args) {
        int n = 3;
        // Upper half
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) System.out.print("*");
            for (int j = 1; j <= 2 * (n - i); j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print("*");
            System.out.println();
        }
        // Lower half
        for (int i = n - 1; i >= 1; i--) {
            for (int j = 1; j <= i; j++) System.out.print("*");
            for (int j = 1; j <= 2 * (n - i); j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print("*");
            System.out.println();
        }
    }
}
```

Q17. Print a sandglass pattern (n = 5).

```
public class Sandglass {
    public static void main(String[] args) {
        int n = 5;
        // Upper half
        for (int i = n; i >= 1; i--) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= 2 * i - 1; j++) System.out.print("*");
            System.out.println();
        }
        // Lower half
        for (int i = 2; i <= n; i++) {
```

```
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= 2 * i - 1; j++) System.out.print("*");
            System.out.println();
        }
    }
}
```

Q18. Print a number pyramid (1, 22, 333, 22, 1).

```
public class NumberPyramid {
    public static void main(String[] args) {
        int n = 3;
        // Upper
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print(i);
            for (int j = 1; j < i; j++) System.out.print(i);
            System.out.println();
        }
        // Lower
        for (int i = n - 1; i >= 1; i--) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print(i);
            for (int j = 1; j < i; j++) System.out.print(i);
            System.out.println();
        }
    }
}
```

Q19. Print a checkerboard pattern (4x4).

```
public class Checkerboard {
    public static void main(String[] args) {
        int n = 4;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print((i + j) % 2 == 0 ? "*" : " ");
            }
            System.out.println();
        }
    }
```

```
}
```

Q20. Print a heart pattern (simplified).

```
public class HeartPattern {
   public static void main(String[] args) {
      // Top curves
      System.out.println("  *     *  ");
      System.out.println("*   *");
      System.out.println("* *");
      // Bottom triangle
      for (int i = 7; i >= 1; i -= 2) {
         for (int j = 0; j < (7 - i) / 2; j++) System.out.print(" ");
         for (int j = 0; j < i; j++) System.out.print("*");
         System.out.println();
      }
   }
}
```

Hard Coding (10)

Q21. Print a spiral number pattern (1-9 in 3x3 spiral).

```
public class SpiralNumbers {
   public static void main(String[] args) {
      int n = 3;
      int[][] mat = new int[n][n];
      int num = 1, top = 0, bottom = n - 1, left = 0, right = n - 1;

      while (top <= bottom && left <= right) {
         for (int i = left; i <= right; i++) mat[top][i] = num++;
         top++;
         for (int i = top; i <= bottom; i++) mat[i][right] = num++;
         right--;
         if (top <= bottom) {
            for (int i = right; i >= left; i--) mat[bottom][i] = num++;
            bottom--;
         }
         if (left <= right) {
            for (int i = bottom; i >= top; i--) mat[i][left] = num++;
            left++;
         }
```

```
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Q22. Print a multiplication table in pyramid form.

```
public class MultiplicationPyramid {
    public static void main(String[] args) {
        int n = 4;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print("  ");
            for (int j = 1; j <= i; j++) System.out.printf("%2d ", i * j);
            for (int j = i - 1; j >= 1; j--) System.out.printf("%2d ", i * j);
            System.out.println();
        }
    }
}
```

Q23. Print a binary triangle (1, 01, 101, 0101).

```
public class BinaryTriangle {
    public static void main(String[] args) {
        int n = 4;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print((i + j) % 2);
            }
            System.out.println();
        }
    }
}
```

Q24. Print a hollow diamond.

```
public class HollowDiamond {
```

```
    public static void main(String[] args) {
        int n = 5;
        // Upper
        for (int i = 1; i <= n; i += 2) {
            for (int j = 0; j < (n - i) / 2; j++) System.out.print(" ");
            for (int j = 0; j < i; j++) {
                if (j == 0 || j == i - 1) System.out.print("*");
                else System.out.print(" ");
            }
            System.out.println();
        }
        // Lower
        for (int i = n - 2; i >= 1; i -= 2) {
            for (int j = 0; j < (n - i) / 2; j++) System.out.print(" ");
            for (int j = 0; j < i; j++) {
                if (j == 0 || j == i - 1) System.out.print("*");
                else System.out.print(" ");
            }
            System.out.println();
        }
    }
}
```

Q25. Print a number diamond (1, 121, 12321, 121, 1).

```
public class NumberDiamond {
    public static void main(String[] args) {
        int n = 4;
        // Upper
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print(j);
            for (int j = i - 1; j >= 1; j--) System.out.print(j);
            System.out.println();
        }
        // Lower
        for (int i = n - 1; i >= 1; i--) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print(j);
            for (int j = i - 1; j >= 1; j--) System.out.print(j);
```

```
            System.out.println();
        }
    }
}
```

Q26. Print a zig-zag pattern (3 rows, "WELCOME").

```
public class ZigZagPattern {
    public static void main(String[] args) {
        String s = "WELCOME";
        int rows = 3;
        char[][] grid = new char[rows][s.length()];
        for (char[] row : grid) java.util.Arrays.fill(row, ' ');

        int r = 0, dir = 1;
        for (int i = 0; i < s.length(); i++) {
            grid[r][i] = s.charAt(i);
            if (r == 0) dir = 1;
            else if (r == rows - 1) dir = -1;
            r += dir;
        }

        for (char[] row : grid) {
            for (char c : row) System.out.print(c);
            System.out.println();
        }
    }
}
```

Q27. Print a Pascal’s triangle with proper spacing.

```
public class SpacedPascals {
    public static void main(String[] args) {
        int n = 5;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n - i; j++) System.out.print(" ");
            int num = 1;
            for (int j = 0; j <= i; j++) {
                System.out.printf("%3d ", num);
                num = num * (i - j) / (j + 1);
            }
```

```
            System.out.println();
        }
    }
}
```

Q28. Print a mirror image of a right triangle.

```
public class MirrorTriangle {
    public static void main(String[] args) {
        int n = 4;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= i; j++) System.out.print("*");
            System.out.println();
        }
    }
}
```

Q29. Print a combination of pyramid and inverted pyramid (hourglass).

```
public class Hourglass {
    public static void main(String[] args) {
        int n = 5;
        // Upper
        for (int i = n; i >= 1; i--) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= 2 * i - 1; j++) System.out.print("*");
            System.out.println();
        }
        // Lower
        for (int i = 2; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= 2 * i - 1; j++) System.out.print("*");
            System.out.println();
        }
    }
}
```

Q30. Print a complex pattern: alternating stars and numbers.

```
public class AlternatingPattern {
    public static void main(String[] args) {
```

```
        int n = 4;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                if ((i + j) % 2 == 0) System.out.print("*");
                else System.out.print(j);
            }
            System.out.println();
        }
    }
}
```

TOPIC 8: Linear Arrays Memory, Traversal, Insertion & Deletion

Easy MCQs (10)

1. In Java, arrays are:
   A) Primitive types
   B) Objects
   C) Pointers
   D) References only
   Answer: B) Objects

2. How is a 1D array stored in memory?
   A) Randomly
   B) As a linked list
   C) In contiguous memory locations
   D) In heap only, non-contiguous
   Answer: C) In contiguous memory locations

3. What is the index of the first element in a Java array?
   A) 1
   B) 0
   C) -1
   D) None
   Answer: B) 0

4. Which statement creates an array of 5 integers?
   A) `int[] arr = new int[5];`
   B) `int arr[5];`

C) `array<int> arr(5);`
D) `int arr = {5};`
Answer: A) `int[] arr = new int[5];`

5. The default value of `int` array elements is:
A) `null`
B) `0`
C) `-1`
D) Garbage
Answer: B) `0`

6. How to get the length of an array `arr`?
A) `arr.length()`
B) `arr.size()`
C) `arr.length`
D) `length(arr)`
Answer: C) `arr.length`

7. Which loop is best for traversing an array?
A) `while`
B) `do-while`
C) `for`
D) All are equal
Answer: C) `for`

8. What happens if you access `arr[5]` in an array of size 5?
A) Returns 0
B) Compilation error
C) `ArrayIndexOutOfBoundsException`
D) Undefined behavior
Answer: C) `ArrayIndexOutOfBoundsException`

9. Arrays in Java are:
A) Fixed-size
B) Dynamic
C) Resizable
D) Linked
Answer: A) Fixed-size

10. The base address of an array refers to:
   A) Last element
   B) Middle element
   C) First element
   D) Null pointer
   Answer: C) First element

Medium MCQs (10)

1. In row-major order, the address of `arr[i]` is:
   A) `base + i`
   B) `base + i * size_of_element`
   C) `base * i`
   D) `i * size`
   Answer: B) `base + i * size_of_element`

2. What is the time complexity of accessing an array element by index?
   A) O(1)
   B) O(log n)
   C) O(n)
   D) $O(n^2)$
   Answer: A) O(1)

3. Which operation is O(n) in a linear array?
   A) Access
   B) Update
   C) Insertion at beginning
   D) Length check
   Answer: C) Insertion at beginning

4. After deletion at index `k`, elements from `k+1` to `n-1` are:
   A) Removed
   B) Shifted left by one
   C) Shifted right by one
   D) Set to zero
   Answer: B) Shifted left by one

5. What is the space complexity of an array of size n?
   A) O(1)

B) O(log n)
C) O(n)
D) O(n^2)
Answer: C) O(n)

6. Which is true about Java arrays?
A) They store primitives and objects
B) They are always initialized
C) They know their length
D) All of the above
Answer: D) All of the above

7. To insert an element at the end of a full array:
A) Possible without resizing
B) Requires new array
C) Automatically resizes
D) Throws exception
Answer: B) Requires new array

8. The `length` field of an array is:
A) A method
B) A final instance variable
C) Mutable
D) Not accessible
Answer: B) A final instance variable

9. What is printed?

```
int[] arr = new int[3];
System.out.println(arr[0]);
```

A) `null`
B) `0`
C) Garbage
D) Compilation error
Answer: B) `0`

10. Which loop avoids `ArrayIndexOutOfBoundsException`?
A) `for (int i = 0; i <= arr.length; i++)`

B) `for (int i = 1; i < arr.length; i++)`
C) `for (int i = 0; i < arr.length; i++)`
D) `for (int i = 0; i <= arr.length - 1; i++)`
Answer: C) `for (int i = 0; i < arr.length; i++)`

Hard MCQs (10)

1. In memory, a 2D array `int[][] arr = new int[3][4]` is stored as:
A) Single contiguous block of 12 ints
B) Array of 3 references, each to a 4-int array
C) Column-major order
D) Linked structure
Answer: B) Array of 3 references, each to a 4-int array

2. What is the address of `arr[i][j]` in row-major 2D array (base B, element size S)?
A) `B + (i * cols + j) * S`
B) `B + (j * rows + i) * S`
C) `B + i * S + j * S`
D) Not contiguous
Answer: A) `B + (i * cols + j) * S`

3. After inserting at index 0 in an array of size n, how many elements are moved?
A) 0
B) 1
C) n
D) n-1
Answer: C) n

4. Which deletion strategy leaves the array compact?
A) Swap with last element
B) Shift left
C) Mark as deleted
D) Both A and B
Answer: B) Shift left

5. What is the amortized cost of dynamic array insertion (like ArrayList)?
A) O(1)
B) O(log n)
C) O(n)

D) $O(n^2)$
Answer: A) O(1)

6. In Java, where is the array object stored?
A) Stack
B) Heap
C) Code segment
D) Register
Answer: B) Heap

7. The reference to an array is stored in:
A) Heap
B) Stack (for local variables)
C) Both A and B
D) Native memory
Answer: B) Stack (for local variables)

8. What is the output?

```
int[] a = {1, 2, 3};
int[] b = a;
b[0] = 10;
System.out.println(a[0]);
```

A) 1
B) 10
C) Compilation error
D) Runtime exception
Answer: B) 10

9. Which is NOT a limitation of arrays?
A) Fixed size
B) Costly insert/delete
C) Homogeneous elements
D) Slow access time
Answer: D) Slow access time

10. What is the minimum number of moves to delete all elements from an array of size n?

A) 0
B) n
C) n(n+1)/2
D) Not applicable
Answer: A) 0 *(Just dereference; GC handles it)*

Coding Problems

Easy Coding (10)

Q1. Traverse and print all elements of an array.

```
public class ArrayTraversal {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40};
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
        }
    }
}
```

Q2. Find the sum of all elements in an array.

```
public class ArraySum {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4};
        int sum = 0;
        for (int x : arr) sum += x;
        System.out.println(sum);
    }
}
```

Q3. Find the maximum element in an array.

```
public class MaxElement {
    public static void main(String[] args) {
        int[] arr = {5, 2, 8, 1};
        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > max) max = arr[i];
        }
        System.out.println(max);
```

```
    }
}
```

Q4. Check if an array contains a specific value.

```
public class ContainsValue {
    public static void main(String[] args) {
        int[] arr = {3, 7, 1, 9};
        int target = 7;
        boolean found = false;
        for (int x : arr) {
            if (x == target) {
                found = true;
                break;
            }
        }
        System.out.println(found);
    }
}
```

Q5. Reverse an array in place.

```
public class ReverseArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4};
        int n = arr.length;
        for (int i = 0; i < n / 2; i++) {
            int temp = arr[i];
            arr[i] = arr[n - 1 - i];
            arr[n - 1 - i] = temp;
        }
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q6. Copy elements from one array to another.

```
public class ArrayCopy {
    public static void main(String[] args) {
        int[] src = {1, 2, 3};
        int[] dest = new int[src.length];
        for (int i = 0; i < src.length; i++) {
```

```
            dest[i] = src[i];
        }
        for (int x : dest) System.out.print(x + " ");
    }
}
```

Q7. Initialize an array with user-defined values.

```
public class InitializeArray {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        for (int x : arr) System.out.println(x);
    }
}
```

Q8. Count even numbers in an array.

```
public class CountEvens {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6};
        int count = 0;
        for (int x : arr) {
            if (x % 2 == 0) count++;
        }
        System.out.println(count);
    }
}
```

Q9. Print array elements in reverse order (without modifying array).

```
public class ReversePrint {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4};
        for (int i = arr.length - 1; i >= 0; i--) {
            System.out.println(arr[i]);
        }
    }
}
```

Q10. Find the average of array elements.

```
public class ArrayAverage {
    public static void main(String[] args) {
```

```
        int[] arr = {10, 20, 30};
        double sum = 0;
        for (int x : arr) sum += x;
        System.out.println(sum / arr.length);
    }
}
```

Medium Coding (10)

Q11. Insert an element at a given index (assume space available).

```
public class InsertElement {
    // Returns new size
    public static int insert(int[] arr, int n, int index, int value) {
        for (int i = n; i > index; i--) {
            arr[i] = arr[i - 1];
        }
        arr[index] = value;
        return n + 1;
    }

    public static void main(String[] args) {
        int[] arr = new int[5];
        arr[0] = 10; arr[1] = 20; arr[2] = 30;
        int newSize = insert(arr, 3, 1, 15);
        for (int i = 0; i < newSize; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Q12. Delete an element at a given index.

```
public class DeleteElement {
    // Returns new size
    public static int delete(int[] arr, int n, int index) {
        for (int i = index; i < n - 1; i++) {
            arr[i] = arr[i + 1];
        }
        return n - 1;
    }
```

```
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40};
        int newSize = delete(arr, 4, 1);
        for (int i = 0; i < newSize; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Q13. Remove all occurrences of a value from an array.

```
public class RemoveAll {
    public static int removeAll(int[] arr, int n, int value) {
        int writeIndex = 0;
        for (int readIndex = 0; readIndex < n; readIndex++) {
            if (arr[readIndex] != value) {
                arr[writeIndex++] = arr[readIndex];
            }
        }
        return writeIndex;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 4};
        int newSize = removeAll(arr, 5, 2);
        for (int i = 0; i < newSize; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Q14. Rotate array to the right by k steps.

```
public class RotateArray {
    public static void rotate(int[] arr, int k) {
        k %= arr.length;
        reverse(arr, 0, arr.length - 1);
        reverse(arr, 0, k - 1);
        reverse(arr, k, arr.length - 1);
```

```
    }

    static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        rotate(arr, 2);
        for (int x : arr) System.out.print(x + " "); // 4 5 1 2 3
    }
}
```

Q15. Find the second smallest element.

```
public class SecondSmallest {
    public static void main(String[] args) {
        int[] arr = {5, 2, 8, 1, 9};
        int first = Integer.MAX_VALUE, second = Integer.MAX_VALUE;
        for (int x : arr) {
            if (x < first) {
                second = first;
                first = x;
            } else if (x < second && x != first) {
                second = x;
            }
        }
        System.out.println(second);
    }
}
```

Q16. Merge two sorted arrays into one sorted array.

```
public class MergeSorted {
    public static void main(String[] args) {
        int[] a = {1, 3, 5};
```

```
        int[] b = {2, 4, 6};
        int[] merged = new int[a.length + b.length];
        int i = 0, j = 0, k = 0;

        while (i < a.length && j < b.length) {
            if (a[i] <= b[j]) merged[k++] = a[i++];
            else merged[k++] = b[j++];
        }
        while (i < a.length) merged[k++] = a[i++];
        while (j < b.length) merged[k++] = b[j++];

        for (int x : merged) System.out.print(x + " ");
    }
}
```

Q17. Check if array is sorted in ascending order.

```
public class IsSorted {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4};
        boolean sorted = true;
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < arr[i - 1]) {
                sorted = false;
                break;
            }
        }
        System.out.println(sorted);
    }
}
```

Q18. Find the index of the first occurrence of a value.

```
public class FirstOccurrence {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 20, 40};
        int target = 20;
        int index = -1;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                index = i;
```

```
            break;
        }
    }
    System.out.println(index);
  }
}
```

Q19. Compute the frequency of each element.

```
import java.util.*;
public class FrequencyCount {
  public static void main(String[] args) {
    int[] arr = {1, 2, 2, 3, 1};
    Map<Integer, Integer> freq = new HashMap<>();
    for (int x : arr) {
      freq.put(x, freq.getOrDefault(x, 0) + 1);
    }
    freq.forEach((k, v) -> System.out.println(k + ": " + v));
  }
}
```

Q20. Find the equilibrium index (sum left = sum right).

```
public class EquilibriumIndex {
  public static void main(String[] args) {
    int[] arr = {-7, 1, 5, 2, -4, 3, 0};
    int total = 0;
    for (int x : arr) total += x;

    int leftSum = 0;
    for (int i = 0; i < arr.length; i++) {
      total -= arr[i];
      if (leftSum == total) {
        System.out.println("Equilibrium index: " + i);
        return;
      }
      leftSum += arr[i];
    }
    System.out.println("None");
  }
}
```

Hard Coding (10)

Q21. Implement dynamic array (like ArrayList) with resize.

```
public class DynamicArray {
    private int[] data;
    private int size;
    private static final int DEFAULT_CAPACITY = 4;

    public DynamicArray() {
        data = new int[DEFAULT_CAPACITY];
        size = 0;
    }

    public void add(int value) {
        if (size == data.length) resize();
        data[size++] = value;
    }

    private void resize() {
        int[] newData = new int[data.length * 2];
        System.arraycopy(data, 0, newData, 0, data.length);
        data = newData;
    }

    public int get(int index) {
        if (index < 0 || index >= size)
            throw new IndexOutOfBoundsException();
        return data[index];
    }

    public int size() { return size; }

    public static void main(String[] args) {
        DynamicArray da = new DynamicArray();
        for (int i = 1; i <= 10; i++) da.add(i);
        for (int i = 0; i < da.size(); i++) {
            System.out.print(da.get(i) + " ");
        }
```

```
    }
}
```

Q22. Delete duplicates from sorted array in place.

```
public class RemoveDuplicatesSorted {
    public static int removeDuplicates(int[] arr, int n) {
        if (n == 0) return 0;
        int writeIndex = 1;
        for (int readIndex = 1; readIndex < n; readIndex++) {
            if (arr[readIndex] != arr[readIndex - 1]) {
                arr[writeIndex++] = arr[readIndex];
            }
        }
        return writeIndex;
    }

    public static void main(String[] args) {
        int[] arr = {1, 1, 2, 2, 3};
        int newSize = removeDuplicates(arr, 5);
        for (int i = 0; i < newSize; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Q23. Move all zeros to the end while maintaining order.

```
public class MoveZerosToEnd {
    public static void moveZeros(int[] arr) {
        int writeIndex = 0;
        for (int readIndex = 0; readIndex < arr.length; readIndex++) {
            if (arr[readIndex] != 0) {
                arr[writeIndex++] = arr[readIndex];
            }
        }
        while (writeIndex < arr.length) {
            arr[writeIndex++] = 0;
        }
    }
```

```
    public static void main(String[] args) {
        int[] arr = {0, 1, 0, 3, 12};
        moveZeros(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q24. Find the majority element (appears > n/2 times).

```
public class MajorityElement {
    public static int findMajority(int[] arr) {
        int candidate = 0, count = 0;
        for (int x : arr) {
            if (count == 0) candidate = x;
            count += (x == candidate) ? 1 : -1;
        }
        return candidate;
    }

    public static void main(String[] args) {
        int[] arr = {2, 2, 1, 1, 1, 2, 2};
        System.out.println(findMajority(arr));
    }
}
```

Q25. Implement cyclic rotation (left by k).

```
public class CyclicRotateLeft {
    public static void rotate(int[] arr, int k) {
        k %= arr.length;
        reverse(arr, 0, k - 1);
        reverse(arr, k, arr.length - 1);
        reverse(arr, 0, arr.length - 1);
    }

    static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
```

```
            end--;
        }
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        rotate(arr, 2);
        for (int x : arr) System.out.print(x + " "); // 3 4 5 1 2
    }
}
```

Q26. Find the missing number in 1..n+1.

```
public class MissingNumber {
    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5, 6}; // n=6, missing=3
        int n = arr.length + 1;
        int expected = n * (n + 1) / 2;
        int actual = 0;
        for (int x : arr) actual += x;
        System.out.println(expected - actual);
    }
}
```

Q27. Sort array of 0s, 1s, and 2s in one pass (Dutch National Flag).

```
public class Sort012 {
    public static void sort(int[] arr) {
        int low = 0, mid = 0, high = arr.length - 1;
        while (mid <= high) {
            switch (arr[mid]) {
                case 0:
                    swap(arr, low++, mid++);
                    break;
                case 1:
                    mid++;
                    break;
                case 2:
                    swap(arr, mid, high--);
                    break;
            }
```

```
        }
    }

    static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public static void main(String[] args) {
        int[] arr = {0, 2, 1, 2, 0};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q28. Find the union of two unsorted arrays.

```
import java.util.*;
public class UnionOfArrays {
    public static void main(String[] args) {
        int[] a = {1, 2, 3};
        int[] b = {2, 3, 4};
        Set<Integer> union = new HashSet<>();
        for (int x : a) union.add(x);
        for (int x : b) union.add(x);
        System.out.println(union);
    }
}
```

Q29. Find the intersection of two sorted arrays.

```
public class IntersectionSorted {
    public static void main(String[] args) {
        int[] a = {1, 2, 2, 3};
        int[] b = {2, 2, 4};
        int i = 0, j = 0;
        while (i < a.length && j < b.length) {
            if (a[i] == b[j]) {
                System.out.print(a[i] + " ");
                i++;
```

```
            j++;
            // Skip duplicates
            while (i < a.length && a[i] == a[i - 1]) i++;
            while (j < b.length && b[j] == b[j - 1]) j++;
        } else if (a[i] < b[j]) {
            i++;
        } else {
            j++;
        }
      }
    }
}
```

Q30. Implement array-based stack with push/pop.

```
public class ArrayStack {
    private int[] stack;
    private int top;
    private static final int CAPACITY = 10;

    public ArrayStack() {
        stack = new int[CAPACITY];
        top = -1;
    }

    public void push(int x) {
        if (top == CAPACITY - 1) throw new RuntimeException("Stack overflow");
        stack[++top] = x;
    }

    public int pop() {
        if (top == -1) throw new RuntimeException("Stack underflow");
        return stack[top--];
    }

    public boolean isEmpty() { return top == -1; }

    public static void main(String[] args) {
        ArrayStack s = new ArrayStack();
        s.push(10);
```

```
        s.push(20);
        System.out.println(s.pop()); // 20
        System.out.println(s.pop()); // 10
    }
}
```

TOPIC 9: Multi-Dimensional Arrays as Collection of 1D Arrays

Easy MCQs (10)

1. In Java, a 2D array is:
   A) A single contiguous block
   B) An array of arrays
   C) A linked list of rows
   D) Column-major by default
   Answer: B) An array of arrays

2. How to declare a 3x4 integer matrix?
   A) `int[3][4] mat;`
   B) `int mat[3][4];`
   C) `int[][] mat = new int[3][4];`
   D) `matrix<int> mat(3,4);`
   Answer: C) `int[][] mat = new int[3][4];`

3. The number of rows in `int[][] arr = new int[5][3]` is:
   A) 3
   B) 5
   C) 15
   D) 8
   Answer: B) 5

4. How to access the element in row 2, column 1?
   A) `arr[2,1]`
   B) `arr[2][1]`
   C) `arr(2,1)`
   D) `arr[1][2]`
   Answer: B) `arr[2][1]`

5. What is `arr.length` for `int[][] arr = new int[4][5]`?

A) 4
B) 5
C) 20
D) 9
Answer: A) 4

6. Each row in a 2D array is:
A) A 1D array
B) An integer
C) A reference
D) Both A and C
Answer: D) Both A and C

7. Which loop is best for traversing a 2D array row by row?
A) Single `for`
B) Nested `for`
C) `while` only
D) Enhanced `for` only
Answer: B) Nested `for`

8. The default value of each element in `new int[2][3]` is:
A) `null`
B) `0`
C) `-1`
D) Garbage
Answer: B) `0`

9. Can rows in a 2D array have different lengths?
A) No
B) Yes (jagged array)
C) Only in C++
D) Only if declared as `Object[][]`
Answer: B) Yes (jagged array)

10. How to get the number of columns in row 0?
A) `arr[0].length()`
B) `arr[0].size()`
C) `arr[0].length`
D) `length(arr[0])`

Answer: C) `arr[0].length`

Medium MCQs (10)

1. What is printed?

```
int[][] arr = {{1,2}, {3,4,5}};
System.out.println(arr[1].length);
```

A) 2
B) 3
C) 5
D) Compilation error
Answer: B) 3

2. Which code correctly prints all elements of a 2D array?
A)

```
for (int i = 0; i < arr.length; i++)
    for (int j = 0; j < arr[i].length; j++)
        System.out.println(arr[i][j]);
```

B)

```
for (int[] row : arr)
    for (int x : row)
        System.out.println(x);
```

C) Both A and B
D) Neither
Answer: C) Both A and B

3. In memory, a 2D array `int[3][4]` occupies:
A) 12 contiguous integers
B) 1 array of 3 refs + 3 arrays of 4 ints
C) 4 arrays of 3 ints
D) Linked structure
Answer: B) 1 array of 3 refs + 3 arrays of 4 ints

4. What is the time complexity to traverse an m×n matrix?
   A) O(m)
   B) O(n)
   C) O(m+n)
   D) O(m×n)
   Answer: D) O(m×n)

5. Which operation is O(1) in a 2D array?
   A) Accessing `arr[i][j]`
   B) Inserting a row
   C) Deleting a column
   D) Resizing
   Answer: A) Accessing `arr[i][j]`

6. What is the output?

```
int[][] arr = new int[2][];
arr[0] = new int[3];
arr[1] = new int[2];
System.out.println(arr.length + " " + arr[0].length);
```

   A) `2 3`
   B) `3 2`
   C) `2 2`
   D) `3 3`
   Answer: A) `2 3`

7. To copy a 2D array, you must:
   A) Use `System.arraycopy` once
   B) Clone each row individually
   C) Assign reference
   D) Use `Arrays.copyOf`
   Answer: B) Clone each row individually

8. Which is a valid jagged array declaration?
   A) `int[][] arr = new int[3][];`
   B) `int[][] arr = {{1}, {2,3}, {4,5,6}};`
   C) Both A and B
   D) Neither

Answer: C) Both A and B

9. What is printed?

```
int[][] arr = {{1,2}, {3,4}};
int sum = 0;
for (int[] row : arr)
    for (int x : row)
        sum += x;
System.out.println(sum);
```

A) 6
B) 10
C) 4
D) 8
Answer: B) 10

10. The base address of a 2D array points to:
A) First element of first row
B) Array of row references
C) Last element
D) Middle element
Answer: B) Array of row references

Hard MCQs (10)

1. What is the memory layout of `int[][] arr = new int[3][4]`?
A) One block of 12 ints
B) One block of 3 refs + three blocks of 4 ints each
C) Column-major blocks
D) Single object with metadata
Answer: B) One block of 3 refs + three blocks of 4 ints each

2. After `int[][] a = b;`, modifying `a[0][0]` affects `b` because:
A) Deep copy
B) Shallow copy (shared references)
C) Immutable
D) Compiler optimization
Answer: B) Shallow copy (shared references)

3. Which is true about cache performance?
A) Row-major traversal is faster
B) Column-major is faster in Java
C) No difference
D) Depends on JVM version
Answer: A) Row-major traversal is faster

4. What is the address of `arr[i][j]` if each int is 4 bytes?
A) `base + i * 4 + j * 4`
B) `rowBase[i] + j * 4`
C) `base + (i * cols + j) * 4`
D) Not contiguous
Answer: B) `rowBase[i] + j * 4`

5. Creating a deep copy of a 2D array requires:
A) One `clone()`
B) Loop to clone each row
C) `Arrays.deepClone()`
D) Serialization
Answer: B) Loop to clone each row

6. In a jagged array, `arr[i].length` can be:
A) Same for all i
B) Different for each i
C) Zero
D) All of the above
Answer: D) All of the above

7. What is the output?

```
int[][] arr = new int[2][2];
arr[0] = arr[1];
arr[1][0] = 5;
System.out.println(arr[0][0]);
```

A) 0
B) 5
C) Compilation error

D) Runtime exception
Answer: B) 5

8. Which operation has O(m×n) time and O(1) space?
A) Matrix transpose (in-place for square)
B) Matrix multiplication
C) Row deletion
D) Column insertion
Answer: A) Matrix transpose (in-place for square)

9. The `Arrays.deepEquals()` method is used for:
A) Comparing 1D arrays
B) Comparing 2D arrays element-wise
C) Hashing
D) Sorting
Answer: B) Comparing 2D arrays element-wise

10. What is the minimum memory overhead for `new int[1000][1000]`?
A) 4,000,000 bytes
B) 4,000,000 + 8,000 bytes (refs)
C) 4,000,000 + 8 bytes
D) 4,000,000 + 8,008 bytes
Answer: D) 4,000,000 + 8,008 bytes
(1000 refs × 8 bytes + 1 array header + 1000 row headers)

Coding Problems

Easy Coding (10)

Q1. Print all elements of a 2D array row by row.

```
public class Print2D {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }
```

```
    }
}
```

Q2. Find the sum of all elements in a matrix.

```
public class MatrixSum {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        int sum = 0;
        for (int[] row : mat) {
            for (int x : row) {
                sum += x;
            }
        }
        System.out.println(sum);
    }
}
```

Q3. Print the diagonal elements of a square matrix.

```
public class DiagonalElements {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        for (int i = 0; i < mat.length; i++) {
            System.out.println(mat[i][i]);
        }
    }
}
```

Q4. Find the maximum element in a 2D array.

```
public class MaxInMatrix {
    public static void main(String[] args) {
        int[][] mat = {{1, 5}, {3, 2}};
        int max = mat[0][0];
        for (int[] row : mat) {
            for (int x : row) {
                if (x > max) max = x;
            }
        }
        System.out.println(max);
    }
```

```
}
```

Q5. Initialize a 3x3 matrix with zeros.

```
public class ZeroMatrix {
    public static void main(String[] args) {
        int[][] mat = new int[3][3];
        for (int[] row : mat) {
            for (int x : row) {
                System.out.print(x + " ");
            }
            System.out.println();
        }
    }
}
```

Q6. Copy a 2D array (shallow copy).

```
public class ShallowCopy2D {
    public static void main(String[] args) {
        int[][] src = {{1, 2}, {3, 4}};
        int[][] dest = new int[src.length][];
        for (int i = 0; i < src.length; i++) {
            dest[i] = src[i]; // shallow
        }
        dest[0][0] = 10;
        System.out.println(src[0][0]); // 10
    }
}
```

Q7. Print the last row of a matrix.

```
public class LastRow {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}, {5, 6}};
        int[] last = mat[mat.length - 1];
        for (int x : last) System.out.print(x + " ");
    }
}
```

Q8. Count even numbers in a matrix.

```
public class CountEvens2D {
```

```
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        int count = 0;
        for (int[] row : mat) {
            for (int x : row) {
                if (x % 2 == 0) count++;
            }
        }
        System.out.println(count);
    }
}
```

Q9. Find the sum of each row.

```
public class RowSums {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        for (int i = 0; i < mat.length; i++) {
            int sum = 0;
            for (int x : mat[i]) sum += x;
            System.out.println("Row " + i + ": " + sum);
        }
    }
}
```

Q10. Check if a matrix is square.

```
public class IsSquareMatrix {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        boolean square = true;
        for (int[] row : mat) {
            if (row.length != mat.length) {
                square = false;
                break;
            }
        }
        System.out.println(square);
    }
}
```

CodInClub
Powered By BridgeLabz

Medium Coding (10)

Q11. Transpose a square matrix in place.

```
public class InPlaceTranspose {
    public static void transpose(int[][] mat) {
        for (int i = 0; i < mat.length; i++) {
            for (int j = i + 1; j < mat[i].length; j++) {
                int temp = mat[i][j];
                mat[i][j] = mat[j][i];
                mat[j][i] = temp;
            }
        }
    }

    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        transpose(mat);
        for (int[] row : mat) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

Q12. Multiply two matrices (m×n and n×p).

```
public class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] a = {{1, 2}, {3, 4}};
        int[][] b = {{5, 6}, {7, 8}};
        int[][] c = new int[a.length][b[0].length];

        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < b[0].length; j++) {
                for (int k = 0; k < b.length; k++) {
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
        }
```

```
        for (int[] row : c) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

Q13. Create a jagged array and print it.

```
public class JaggedArray {
    public static void main(String[] args) {
        int[][] jagged = {
            {1},
            {2, 3},
            {4, 5, 6}
        };
        for (int i = 0; i < jagged.length; i++) {
            for (int j = 0; j < jagged[i].length; j++) {
                System.out.print(jagged[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Q14. Find the saddle point (min in row, max in column).

```
public class SaddlePoint {
    public static void main(String[] args) {
        int[][] mat = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        outer: for (int i = 0; i < mat.length; i++) {
            int minRow = mat[i][0], colIndex = 0;
            for (int j = 1; j < mat[i].length; j++) {
                if (mat[i][j] < minRow) {
                    minRow = mat[i][j];
                    colIndex = j;
                }
            }
            boolean isSaddle = true;
            for (int k = 0; k < mat.length; k++) {
                if (mat[k][colIndex] > minRow) {
```

```
                isSaddle = false;
                break;
            }
        }
        if (isSaddle) {
            System.out.println("Saddle point: " + minRow + " at (" + i + "," + colIndex +
")");
            break outer;
        }
    }
  }
}
```

Q15. Rotate a square matrix 90 degrees clockwise.

```
public class Rotate90Clockwise {
    public static void rotate(int[][] mat) {
        int n = mat.length;
        // Transpose
        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                int temp = mat[i][j];
                mat[i][j] = mat[j][i];
                mat[j][i] = temp;
            }
        }
        // Reverse each row
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n / 2; j++) {
                int temp = mat[i][j];
                mat[i][j] = mat[i][n - 1 - j];
                mat[i][n - 1 - j] = temp;
            }
        }
    }

    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        rotate(mat);
```

```
        for (int[] row : mat) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

Q16. Find the sum of boundary elements.

```
public class BoundarySum {
    public static void main(String[] args) {
        int[][] mat = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int sum = 0, rows = mat.length, cols = mat[0].length;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (i == 0 || i == rows - 1 || j == 0 || j == cols - 1) {
                    sum += mat[i][j];
                }
            }
        }
        System.out.println(sum);
    }
}
```

Q17. Check if two matrices are equal.

```
import java.util.Arrays;
public class MatrixEquality {
    public static void main(String[] args) {
        int[][] a = {{1, 2}, {3, 4}};
        int[][] b = {{1, 2}, {3, 4}};
        System.out.println(Arrays.deepEquals(a, b));
    }
}
```

Q18. Find the row with maximum sum.

```
public class MaxRowSum {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        int maxSum = Integer.MIN_VALUE, maxRow = -1;
        for (int i = 0; i < mat.length; i++) {
```

```
            int sum = 0;
            for (int x : mat[i]) sum += x;
            if (sum > maxSum) {
                maxSum = sum;
                maxRow = i;
            }
        }
        System.out.println("Row " + maxRow + " has max sum " + maxSum);
    }
}
```

Q19. Create an identity matrix of size n.

```
public class IdentityMatrix {
    public static void main(String[] args) {
        int n = 3;
        int[][] id = new int[n][n];
        for (int i = 0; i < n; i++) {
            id[i][i] = 1;
        }
        for (int[] row : id) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

Q20. Swap two rows in a matrix.

```
public class SwapRows {
    public static void swapRows(int[][] mat, int r1, int r2) {
        int[] temp = mat[r1];
        mat[r1] = mat[r2];
        mat[r2] = temp;
    }

    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        swapRows(mat, 0, 1);
        for (int[] row : mat) {
            for (int x : row) System.out.print(x + " ");
```

```
            System.out.println();
        }
    }
}
```

Hard Coding (10)

Q21. Perform deep copy of a 2D array.

```
public class DeepCopy2D {
    public static int[][] deepCopy(int[][] original) {
        if (original == null) return null;
        int[][] copy = new int[original.length][];
        for (int i = 0; i < original.length; i++) {
            copy[i] = original[i].clone();
        }
        return copy;
    }

    public static void main(String[] args) {
        int[][] src = {{1, 2}, {3, 4}};
        int[][] dest = deepCopy(src);
        dest[0][0] = 10;
        System.out.println(src[0][0]); // 1
    }
}
```

Q22. Find the determinant of a 3x3 matrix.

```
public class Determinant3x3 {
    public static int det(int[][] mat) {
        return mat[0][0] * (mat[1][1] * mat[2][2] - mat[1][2] * mat[2][1])
             - mat[0][1] * (mat[1][0] * mat[2][2] - mat[1][2] * mat[2][0])
             + mat[0][2] * (mat[1][0] * mat[2][1] - mat[1][1] * mat[2][0]);
    }

    public static void main(String[] args) {
        int[][] mat = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        System.out.println(det(mat)); // 0
    }
}
```

Q23. Solve a system of linear equations using Cramer's rule (2x2).

```
public class CramersRule {
    public static void main(String[] args) {
        // Equations: a1x + b1y = c1, a2x + b2y = c2
        int a1 = 2, b1 = 3, c1 = 8;
        int a2 = 4, b2 = 5, c2 = 14;

        int det = a1 * b2 - a2 * b1;
        if (det == 0) {
            System.out.println("No unique solution");
            return;
        }
        int detX = c1 * b2 - c2 * b1;
        int detY = a1 * c2 - a2 * c1;
        double x = (double) detX / det;
        double y = (double) detY / det;
        System.out.println("x = " + x + ", y = " + y);
    }
}
```

Q24. Implement sparse matrix using array of triplets (row, col, value).

```
public class SparseMatrix {
    static class Triplet {
        int row, col, val;
        Triplet(int r, int c, int v) { row = r; col = c; val = v; }
    }

    public static void main(String[] args) {
        int[][] mat = {{0, 0, 3}, {4, 0, 0}, {0, 5, 0}};
        java.util.List<Triplet> sparse = new java.util.ArrayList<>();
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                if (mat[i][j] != 0) {
                    sparse.add(new Triplet(i, j, mat[i][j]));
                }
            }
        }
        for (Triplet t : sparse) {
            System.out.println("(" + t.row + "," + t.col + ") = " + t.val);
```

```
        }
    }
}
```

Q25. Find the longest sequence of 1s in a binary matrix row.

```
public class LongestOnesRow {
    public static void main(String[] args) {
        int[][] mat = {{0, 1, 1, 1}, {0, 0, 1, 1}, {1, 1, 1, 1}, {0, 0, 0, 0}};
        int maxLen = 0, bestRow = -1;
        for (int i = 0; i < mat.length; i++) {
            int current = 0, maxInRow = 0;
            for (int j = 0; j < mat[i].length; j++) {
                if (mat[i][j] == 1) {
                    current++;
                    maxInRow = Math.max(maxInRow, current);
                } else {
                    current = 0;
                }
            }
            if (maxInRow > maxLen) {
                maxLen = maxInRow;
                bestRow = i;
            }
        }
        System.out.println("Row " + bestRow + " has " + maxLen + " consecutive 1s");
    }
}
```

Q26. Multiply a matrix by a scalar.

```
public class ScalarMultiplication {
    public static void multiply(int[][] mat, int scalar) {
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                mat[i][j] *= scalar;
            }
        }
    }

    public static void main(String[] args) {
```

```
        int[][] mat = {{1, 2}, {3, 4}};
        multiply(mat, 2);
        for (int[] row : mat) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

Q27. Find the trace of a square matrix (sum of diagonal).

```
public class MatrixTrace {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {3, 4}};
        int trace = 0;
        for (int i = 0; i < mat.length; i++) {
            trace += mat[i][i];
        }
        System.out.println(trace);
    }
}
```

Q28. Check if a matrix is symmetric.

```
public class SymmetricMatrix {
    public static void main(String[] args) {
        int[][] mat = {{1, 2}, {2, 1}};
        boolean symmetric = true;
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                if (mat[i][j] != mat[j][i]) {
                    symmetric = false;
                    break;
                }
            }
            if (!symmetric) break;
        }
        System.out.println(symmetric);
    }
}
```

Q29. Implement matrix addition.

```
public class MatrixAddition {
    public static int[][] add(int[][] a, int[][] b) {
        int[][] c = new int[a.length][a[0].length];
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                c[i][j] = a[i][j] + b[i][j];
            }
        }
        return c;
    }

    public static void main(String[] args) {
        int[][] a = {{1, 2}, {3, 4}};
        int[][] b = {{5, 6}, {7, 8}};
        int[][] c = add(a, b);
        for (int[] row : c) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

Q30. Find the submatrix with maximum sum (Kadane 2D).

```
public class MaxSubmatrixSum {
    public static void main(String[] args) {
        int[][] mat = {{1, 2, -1, -4, -20},
                       {-8, -3, 4, 2, 1},
                       {3, 8, 10, 1, 3},
                       {-4, -1, 1, 7, -6}};
        int maxSum = Integer.MIN_VALUE;
        int rows = mat.length, cols = mat[0].length;

        for (int left = 0; left < cols; left++) {
            int[] temp = new int[rows];
            for (int right = left; right < cols; right++) {
                for (int i = 0; i < rows; i++) {
                    temp[i] += mat[i][right];
                }
```

```
                // Apply Kadane's on temp
                int currentSum = 0;
                for (int x : temp) {
                    currentSum = Math.max(x, currentSum + x);
                    maxSum = Math.max(maxSum, currentSum);
                }
            }
        }
        System.out.println("Max submatrix sum: " + maxSum);
    }
}
```

TOPIC 10: Applications in Databases, Caching, and Matrix Operations

Easy MCQs (10)

1. In databases, arrays are commonly used to implement:
   A) B-trees
   B) Hash tables
   C) Page buffers
   D) All of the above
   Answer: D) All of the above

2. A cache hit occurs when:
   A) Data is not in cache
   B) Data is found in cache
   C) Cache is full
   D) Disk I/O happens
   Answer: B) Data is found in cache

3. Matrix multiplication is used in:
   A) Computer graphics
   B) Machine learning
   C) Physics simulations
   D) All of the above
   Answer: D) All of the above

4. Which data structure is used for LRU caching?
   A) Array

B) Linked list + hash map
C) Stack
D) Queue
Answer: B) Linked list + hash map

5. In row-major matrix storage, consecutive elements of a row are:
A) Stored far apart
B) Stored contiguously
C) Stored in reverse
D) Not stored
Answer: B) Stored contiguously

6. Database indexes often use:
A) Unsorted arrays
B) Sorted arrays or trees
C) Linked lists only
D) Stacks
Answer: B) Sorted arrays or trees

7. Caching improves performance by reducing:
A) CPU cycles
B) Memory usage
C) Disk I/O or network calls
D) Code size
Answer: C) Disk I/O or network calls

8. A dense matrix has:
A) Mostly zero elements
B) Mostly non-zero elements
C) Random elements
D) No elements
Answer: B) Mostly non-zero elements

9. Which is a common matrix operation in recommendation systems?
A) Transpose
B) Matrix factorization
C) Scalar multiplication
D) Row deletion
Answer: B) Matrix factorization

10. In-memory databases rely heavily on:
A) Disk storage
B) Array-based structures
C) Tape backups
D) Network latency
Answer: B) Array-based structures

Medium MCQs (10)

1. Why is row-major order preferred in Java for matrix operations?
A) Better cache locality during row traversal
B) Required by JVM
C) Easier to implement
D) Column access is faster
Answer: A) Better cache locality during row traversal

2. In a database buffer pool, pages are often stored in:
A) Linked list
B) Array or hash table
C) Stack
D) Queue
Answer: B) Array or hash table

3. The time complexity of naive matrix multiplication ($n \times n$) is:
A) $O(n)$
B) $O(n^2)$
C) $O(n^3)$
D) $O(2^n)$
Answer: C) $O(n^3)$

4. Which caching strategy discards the least recently used item?
A) FIFO
B) LIFO
C) LRU
D) Random
Answer: C) LRU

5. Sparse matrices are stored efficiently using:

A) Full 2D arrays
B) Coordinate list (COO) or CSR
C) Stacks
D) Queues
Answer: B) Coordinate list (COO) or CSR

6. In database query execution, materialized results are often stored in:
A) Temporary arrays
B) Disk only
C) Network buffers
D) CPU registers
Answer: A) Temporary arrays

7. Cache locality is improved by:
A) Random access
B) Sequential access
C) Deep recursion
D) Frequent allocation
Answer: B) Sequential access

8. Which matrix property is exploited in Strassen's algorithm?
A) Sparsity
B) Divide-and-conquer
C) Symmetry
D) Diagonal dominance
Answer: B) Divide-and-conquer

9. In caching, a "cold start" means:
A) Cache is full
B) Cache is empty
C) System is overheating
D) Data is corrupted
Answer: B) Cache is empty

10. Database record storage often uses:
A) Jagged arrays
B) Fixed-size arrays (slotted pages)
C) Linked lists only
D) Trees only

Answer: B) Fixed-size arrays (slotted pages)

Hard MCQs (10)

1. In column-oriented databases, data is stored by:
   A) Row
   B) Column
   C) Page
   D) Index
   Answer: B) Column

2. The memory bandwidth bottleneck in matrix multiplication is mitigated by:
   A) Loop tiling (blocking)
   B) Increasing thread count
   C) Using double precision
   D) Reducing matrix size
   Answer: A) Loop tiling (blocking)

3. In LRU cache implementation, why is a doubly linked list used?
   A) To allow O(1) removal from middle
   B) To save memory
   C) To support FIFO
   D) To avoid hashing
   Answer: A) To allow O(1) removal from middle

4. What is the space complexity of storing a sparse n×n matrix with k non-zeros?
   A) $O(n^2)$
   B) O(k)
   C) O(n)
   D) O(k log n)
   Answer: B) O(k)

5. In database buffer management, the LRU-K algorithm uses:
   A) Last K references
   B) First K references
   C) Random K samples
   D) K hash functions

Answer: A) Last K references

6. Which is true about cache-oblivious algorithms?
A) They don't use cache
B) They optimize for unknown cache size
C) They disable cache
D) They use only registers
Answer: B) They optimize for unknown cache size

7. In matrix chain multiplication, dynamic programming reduces complexity from:
A) $O(n!)$ to $O(n^3)$
B) $O(2^n)$ to $O(n^3)$
C) $O(n^4)$ to $O(n^2)$
D) $O(n^3)$ to $O(n \log n)$
Answer: A) $O(n!)$ to $O(n^3)$

8. Write-back vs write-through caching differs in:
A) When data is written to main memory
B) Cache size
C) Replacement policy
D) Associativity
Answer: A) When data is written to main memory

9. In BLAS (Basic Linear Algebra Subprograms), Level 3 operations are:
A) Vector-vector
B) Matrix-vector
C) Matrix-matrix
D) Scalar operations
Answer: C) Matrix-matrix

10. The curse of dimensionality affects:
A) Only 1D arrays
B) High-dimensional matrices in ML
C) Cache size
D) JVM startup time
Answer: B) High-dimensional matrices in ML

Coding Problems

Easy Coding (10)

Q1. Simulate a simple cache with fixed size (FIFO).

```
import java.util.*;
public class FIFOCache {
    private final int capacity;
    private final Queue<Integer> cache = new LinkedList<>();

    public FIFOCache(int capacity) {
        this.capacity = capacity;
    }

    public boolean access(int key) {
        if (cache.contains(key)) {
            System.out.println("Cache hit: " + key);
            return true;
        } else {
            if (cache.size() >= capacity) cache.poll();
            cache.offer(key);
            System.out.println("Cache miss: " + key);
            return false;
        }
    }

    public static void main(String[] args) {
        FIFOCache cache = new FIFOCache(3);
        cache.access(1); // miss
        cache.access(2); // miss
        cache.access(1); // hit
    }
}
```

Q2. Store database records as array of objects.

```
class Record {
    int id; String name;
    Record(int id, String name) { this.id = id; this.name = name; }
}

public class DatabaseArray {
```

```
    public static void main(String[] args) {
        Record[] db = new Record[2];
        db[0] = new Record(1, "Alice");
        db[1] = new Record(2, "Bob");
        for (Record r : db) {
            System.out.println(r.id + ": " + r.name);
        }
    }
}
```

Q3. Compute dot product of two vectors (used in ML).

```
public class DotProduct {
    public static void main(String[] args) {
        double[] a = {1.0, 2.0, 3.0};
        double[] b = {4.0, 5.0, 6.0};
        double dot = 0.0;
        for (int i = 0; i < a.length; i++) {
            dot += a[i] * b[i];
        }
        System.out.println(dot); // 32.0
    }
}
```

Q4. Initialize a matrix for image processing (grayscale).

```
public class ImageMatrix {
    public static void main(String[] args) {
        int rows = 2, cols = 3;
        int[][] image = new int[rows][cols];
        // Simulate pixel values 0-255
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                image[i][j] = (i + j) * 50;
            }
        }
        for (int[] row : image) {
            for (int pixel : row) System.out.print(pixel + " ");
            System.out.println();
        }
    }
```

```
}
```

Q5. Check if a matrix is diagonal (used in linear algebra).

```
public class DiagonalMatrix {
    public static void main(String[] args) {
        int[][] mat = {{1, 0}, {0, 2}};
        boolean diagonal = true;
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                if (i != j && mat[i][j] != 0) {
                    diagonal = false;
                    break;
                }
            }
            if (!diagonal) break;
        }
        System.out.println(diagonal);
    }
}
```

Q6. Simulate a page buffer (fixed-size array of pages).

```
public class PageBuffer {
    public static void main(String[] args) {
        String[] buffer = new String[3];
        buffer[0] = "Page1";
        buffer[1] = "Page2";
        for (String page : buffer) {
            if (page != null) System.out.println(page);
        }
    }
}
```

Q7. Compute matrix norm (max row sum).

```
public class MatrixNorm {
    public static void main(String[] args) {
        double[][] mat = {{1, -2}, {3, 4}};
        double maxSum = 0;
        for (double[] row : mat) {
            double sum = 0;
```

```
            for (double x : row) sum += Math.abs(x);
            maxSum = Math.max(maxSum, sum);
        }
        System.out.println(maxSum); // 7.0
    }
}
```

Q8. Store sparse matrix in COO format (triplets).

```
import java.util.*;
public class COOSparse {
    public static void main(String[] args) {
        int[][] mat = {{0, 2}, {3, 0}};
        List<int[]> coo = new ArrayList<>();
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                if (mat[i][j] != 0) {
                    coo.add(new int[]{i, j, mat[i][j]});
                }
            }
        }
        for (int[] triplet : coo) {
            System.out.println("(" + triplet[0] + "," + triplet[1] + ") = " + triplet[2]);
        }
    }
}
```

Q9. Simulate a write-through cache (immediate write to DB).

```
import java.util.*;
public class WriteThroughCache {
    private Map<Integer, String> cache = new HashMap<>();
    private Map<Integer, String> db = new HashMap<>();
    private static final int CAPACITY = 2;

    public void put(int key, String value) {
        if (cache.size() >= CAPACITY) {
            // Evict first entry (simplified)
            int firstKey = cache.keySet().iterator().next();
            cache.remove(firstKey);
        }
```

```
        cache.put(key, value);
        db.put(key, value); // Write-through
        System.out.println("Wrote " + key + " to cache and DB");
    }

    public static void main(String[] args) {
        WriteThroughCache c = new WriteThroughCache();
        c.put(1, "A");
        c.put(2, "B");
    }
}
```

Q10. Compute the Frobenius norm of a matrix.

```
public class FrobeniusNorm {
    public static void main(String[] args) {
        double[][] mat = {{1, 2}, {3, 4}};
        double sum = 0;
        for (double[] row : mat) {
            for (double x : row) {
                sum += x * x;
            }
        }
        System.out.println(Math.sqrt(sum)); // ~5.477
    }
}
```

Medium Coding (10)

Q11. Implement LRU cache using LinkedHashMap.

```
import java.util.*;
public class LRUCache<K, V> extends LinkedHashMap<K, V> {
    private final int capacity;

    public LRUCache(int capacity) {
        super(capacity, 0.75f, true); // accessOrder=true
        this.capacity = capacity;
    }

    @Override
```

```
    protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
        return size() > capacity;
    }

    public static void main(String[] args) {
        LRUCache<Integer, String> cache = new LRUCache<>(2);
        cache.put(1, "A");
        cache.put(2, "B");
        cache.get(1); // access
        cache.put(3, "C"); // evicts 2
        System.out.println(cache); // {1=A, 3=C}
    }
}
```

Q12. Multiply two matrices optimized for cache (row-major).

```
public class CacheOptimizedMultiply {
    public static double[][] multiply(double[][] a, double[][] b) {
        int m = a.length, n = b[0].length, p = b.length;
        double[][] c = new double[m][n];
        for (int i = 0; i < m; i++) {
            for (int k = 0; k < p; k++) { // Interchange loops for cache
                for (int j = 0; j < n; j++) {
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
        }
        return c;
    }

    public static void main(String[] args) {
        double[][] a = {{1, 2}, {3, 4}};
        double[][] b = {{5, 6}, {7, 8}};
        double[][] c = multiply(a, b);
        for (double[] row : c) {
            for (double x : row) System.out.printf("%.0f ", x);
            System.out.println();
        }
    }
}
```

Q13. Simulate a database index using sorted array (binary search).

```
import java.util.*;
public class DatabaseIndex {
    static class Entry {
        int key; String value;
        Entry(int k, String v) { key = k; value = v; }
    }

    public static void main(String[] args) {
        Entry[] index = {
            new Entry(1, "Alice"),
            new Entry(3, "Bob"),
            new Entry(5, "Charlie")
        };
        // Binary search for key=3
        int low = 0, high = index.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (index[mid].key == 3) {
                System.out.println("Found: " + index[mid].value);
                break;
            } else if (index[mid].key < 3) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }
}
```

Q14. Implement sparse matrix-vector multiplication.

```
import java.util.*;

public class SparseMatVec {
    static class Triplet {
        int row, col, val;
        Triplet(int r, int c, int v) { row = r; col = c; val = v; }
    }
```

```
    public static int[] multiply(List<Triplet> sparse, int[] vec) {
        int[] result = new int[vec.length]; // assume square
        for (Triplet t : sparse) {
            result[t.row] += t.val * vec[t.col];
        }
        return result;
    }

    public static void main(String[] args) {
        List<Triplet> mat = Arrays.asList(
            new Triplet(0, 1, 2),
            new Triplet(1, 0, 3)
        );
        int[] vec = {10, 20};
        int[] res = multiply(mat, vec);
        System.out.println(Arrays.toString(res)); // [40, 30]
    }
}
```

Q15. Simulate a buffer pool with pinning (reference count).

```
import java.util.*;
public class BufferPool {
    static class Page {
        int id; boolean pinned = false;
        Page(int id) { this.id = id; }
    }

    private Page[] buffers = new Page[2];
    private int nextVictim = 0;

    public Page pin(int pageId) {
        // Check if already in buffer
        for (Page p : buffers) {
            if (p != null && p.id == pageId) {
                p.pinned = true;
                return p;
            }
        }
```

```
        // Evict unpinned page
        while (buffers[nextVictim] != null && buffers[nextVictim].pinned) {
            nextVictim = (nextVictim + 1) % buffers.length;
        }
        Page newPage = new Page(pageId);
        newPage.pinned = true;
        buffers[nextVictim] = newPage;
        nextVictim = (nextVictim + 1) % buffers.length;
        return newPage;
    }

    public static void main(String[] args) {
        BufferPool pool = new BufferPool();
        pool.pin(1);
        pool.pin(2);
        System.out.println("Pinned pages 1 and 2");
    }
}
```

Q16. Compute covariance matrix from data (ML preprocessing).

```
public class CovarianceMatrix {
    public static void main(String[] args) {
        double[][] data = {{1, 2}, {2, 3}, {3, 4}}; // 3 samples, 2 features
        int n = data.length, d = data[0].length;
        // Compute mean
        double[] mean = new double[d];
        for (double[] sample : data) {
            for (int j = 0; j < d; j++) {
                mean[j] += sample[j];
            }
        }
        for (int j = 0; j < d; j++) mean[j] /= n;
        // Compute covariance
        double[][] cov = new double[d][d];
        for (double[] sample : data) {
            for (int i = 0; i < d; i++) {
                for (int j = 0; j < d; j++) {
                    cov[i][j] += (sample[i] - mean[i]) * (sample[j] - mean[j]);
                }
```

```
            }
        }
        for (int i = 0; i < d; i++) {
            for (int j = 0; j < d; j++) {
                cov[i][j] /= (n - 1);
                System.out.printf("%.2f ", cov[i][j]);
            }
            System.out.println();
        }
    }
}
```

Q17. Implement a simple hash table using array of lists (chaining).

```
import java.util.*;
public class SimpleHashTable {
    private static final int SIZE = 10;
    private List<Entry>[] table = new List[SIZE];

    static class Entry { String key; String value; Entry(String k, String v) { key = k; value =
v; } }

    public SimpleHashTable() {
        for (int i = 0; i < SIZE; i++) table[i] = new ArrayList<>();
    }

    private int hash(String key) { return Math.abs(key.hashCode()) % SIZE; }

    public void put(String key, String value) {
        int index = hash(key);
        for (Entry e : table[index]) {
            if (e.key.equals(key)) {
                e.value = value;
                return;
            }
        }
        table[index].add(new Entry(key, value));
    }

    public String get(String key) {
```

```
        int index = hash(key);
        for (Entry e : table[index]) {
            if (e.key.equals(key)) return e.value;
        }
        return null;
    }

    public static void main(String[] args) {
        SimpleHashTable ht = new SimpleHashTable();
        ht.put("name", "Alice");
        System.out.println(ht.get("name"));
    }
}
```

Q18. Apply Gaussian elimination to solve 2x2 system.

```
public class GaussianElimination {
    public static void main(String[] args) {
        // Equations: 2x + y = 5, x - y = 1
        double[][] mat = {{2, 1, 5}, {1, -1, 1}};
        int n = 2;
        // Forward elimination
        double factor = mat[1][0] / mat[0][0];
        for (int j = 0; j <= n; j++) {
            mat[1][j] -= factor * mat[0][j];
        }
        // Back substitution
        double y = mat[1][2] / mat[1][1];
        double x = (mat[0][2] - mat[0][1] * y) / mat[0][0];
        System.out.println("x = " + x + ", y = " + y);
    }
}
```

Q19. Simulate a write-back cache (delayed write).

```
import java.util.*;
public class WriteBackCache {
    static class CacheEntry {
        int key; String value; boolean dirty;
        CacheEntry(int k, String v, boolean d) { key = k; value = v; dirty = d; }
    }
```

```
    private Map<Integer, CacheEntry> cache = new HashMap<>();
    private Map<Integer, String> db = new HashMap<>();
    private static final int CAPACITY = 2;

    public void put(int key, String value) {
        if (cache.size() >= CAPACITY) {
            // Evict first dirty page
            for (Map.Entry<Integer, CacheEntry> e : cache.entrySet()) {
                if (e.getValue().dirty) {
                    db.put(e.getKey(), e.getValue().value);
                    cache.remove(e.getKey());
                    break;
                }
            }
        }
        cache.put(key, new CacheEntry(key, value, true));
    }

    public void flush() {
        for (CacheEntry e : cache.values()) {
            if (e.dirty) {
                db.put(e.key, e.value);
                e.dirty = false;
            }
        }
    }

    public static void main(String[] args) {
        WriteBackCache c = new WriteBackCache();
        c.put(1, "A");
        c.flush();
        System.out.println("Flushed to DB");
    }
}
```

Q20. Compute the rank of a binary matrix (simplified).

```
public class BinaryMatrixRank {
    public static int rank(int[][] mat) {
```

```
        int rows = mat.length, cols = mat[0].length;
        int rank = 0;
        boolean[] rowSelected = new boolean[rows];

        for (int col = 0; col < cols && rank < rows; col++) {
            // Find pivot
            int pivot = -1;
            for (int i = 0; i < rows; i++) {
                if (!rowSelected[i] && mat[i][col] == 1) {
                    pivot = i;
                    break;
                }
            }
            if (pivot == -1) continue;

            // Eliminate
            for (int i = 0; i < rows; i++) {
                if (i != pivot && mat[i][col] == 1) {
                    for (int j = 0; j < cols; j++) {
                        mat[i][j] ^= mat[pivot][j];
                    }
                }
            }
            rowSelected[pivot] = true;
            rank++;
        }
        return rank;
    }

    public static void main(String[] args) {
        int[][] mat = {{1, 0, 1}, {0, 1, 1}, {1, 1, 0}};
        System.out.println("Rank: " + rank(mat));
    }
}
```

Hard Coding (10)

Q21. Implement blocked matrix multiplication (cache-aware).

```
public class BlockedMatrixMultiply {
```

```
    private static final int BLOCK_SIZE = 64;

    public static void multiply(double[][] a, double[][] b, double[][] c) {
        int n = a.length;
        for (int i0 = 0; i0 < n; i0 += BLOCK_SIZE) {
            for (int j0 = 0; j0 < n; j0 += BLOCK_SIZE) {
                for (int k0 = 0; k0 < n; k0 += BLOCK_SIZE) {
                    for (int i = i0; i < Math.min(i0 + BLOCK_SIZE, n); i++) {
                        for (int k = k0; k < Math.min(k0 + BLOCK_SIZE, n); k++) {
                            for (int j = j0; j < Math.min(j0 + BLOCK_SIZE, n); j++) {
                                c[i][j] += a[i][k] * b[k][j];
                            }
                        }
                    }
                }
            }
        }
    }

    public static void main(String[] args) {
        int n = 4;
        double[][] a = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
        double[][] b = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
        double[][] c = new double[n][n];
        multiply(a, b, c);
        for (double[] row : c) {
            for (double x : row) System.out.printf("%.0f ", x);
            System.out.println();
        }
    }
}
```

Q22. Simulate a database with B+ tree index (simplified array-based).

```
import java.util.*;
public class BPlusTreeSim {
    // Simulate leaf nodes as sorted array
    static class LeafNode {
        int[] keys;
        String[] values;
```

```java
        LeafNode(int[] k, String[] v) { keys = k; values = v; }
    }

    public static void main(String[] args) {
        // Leaf node with keys [10, 20, 30]
        LeafNode leaf = new LeafNode(
            new int[]{10, 20, 30},
            new String[]{"A", "B", "C"}
        );
        // Binary search for key=20
        int key = 20;
        int index = Arrays.binarySearch(leaf.keys, key);
        if (index >= 0) {
            System.out.println("Value: " + leaf.values[index]);
        }
    }
}
```

Q23. Implement CSR (Compressed Sparse Row) format.

```java
import java.util.*;
public class CSRMatrix {
    int[] values;    // Non-zero values
    int[] colIndex;  // Column indices
    int[] rowPtr;    // Row pointers

    public CSRMatrix(int[][] dense) {
        List<Integer> vals = new ArrayList<>();
        List<Integer> cols = new ArrayList<>();
        List<Integer> rows = new ArrayList<>();
        rows.add(0);

        for (int i = 0; i < dense.length; i++) {
            for (int j = 0; j < dense[i].length; j++) {
                if (dense[i][j] != 0) {
                    vals.add(dense[i][j]);
                    cols.add(j);
                }
            }
            rows.add(vals.size());
```

```
        }

        values = vals.stream().mapToInt(i -> i).toArray();
        colIndex = cols.stream().mapToInt(i -> i).toArray();
        rowPtr = rows.stream().mapToInt(i -> i).toArray();
    }

    public static void main(String[] args) {
        int[][] mat = {{0, 2}, {3, 0}};
        CSRMatrix csr = new CSRMatrix(mat);
        System.out.println("Values: " + Arrays.toString(csr.values));
        System.out.println("ColIndex: " + Arrays.toString(csr.colIndex));
        System.out.println("RowPtr: " + Arrays.toString(csr.rowPtr));
    }
}
```

Q24. Compute SVD of 2x2 matrix (simplified).

```
public class SVD2x2 {
    public static void main(String[] args) {
        // Matrix A = [[2, 0], [0, 1]]
        // A^T A = [[4,0],[0,1]] → eigenvalues 4,1 → singular values 2,1
        double[][] A = {{2, 0}, {0, 1}};
        double sigma1 = 2, sigma2 = 1;
        System.out.println("Singular values: " + sigma1 + ", " + sigma2);
        // U = I, V = I for diagonal matrix
    }
}
```

Q25. Simulate a multi-level cache (L1 and L2).

```
import java.util.*;
public class MultiLevelCache {
    private Map<Integer, String> l1 = new LinkedHashMap<>(2, 0.75f, true) {
        protected boolean removeEldestEntry(Map.Entry eldest) {
            return size() > 2;
        }
    };
    private Map<Integer, String> l2 = new LinkedHashMap<>(4, 0.75f, true) {
        protected boolean removeEldestEntry(Map.Entry eldest) {
            return size() > 4;
```

```
        }
    };
    private Map<Integer, String> db = new HashMap<>();

    public String get(int key) {
        if (l1.containsKey(key)) {
            System.out.println("L1 hit");
            return l1.get(key);
        } else if (l2.containsKey(key)) {
            System.out.println("L2 hit");
            String val = l2.get(key);
            l1.put(key, val); // Promote to L1
            return val;
        } else {
            System.out.println("DB access");
            String val = db.get(key);
            if (val != null) {
                l2.put(key, val);
                l1.put(key, val);
            }
            return val;
        }
    }

    public static void main(String[] args) {
        MultiLevelCache cache = new MultiLevelCache();
        cache.db.put(1, "A");
        cache.get(1);
    }
}
```

Q26. Implement matrix exponentiation (for Fibonacci).

```
public class MatrixExponentiation {
    public static long[][] multiply(long[][] a, long[][] b) {
        long[][] c = new long[2][2];
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                for (int k = 0; k < 2; k++) {
                    c[i][j] += a[i][k] * b[k][j];
```

```
                }
            }
        }
        return c;
    }

    public static long[][] power(long[][] base, int exp) {
        if (exp == 1) return base;
        long[][] half = power(base, exp / 2);
        long[][] result = multiply(half, half);
        if (exp % 2 == 1) result = multiply(result, base);
        return result;
    }

    public static long fibonacci(int n) {
        if (n <= 1) return n;
        long[][] base = {{1, 1}, {1, 0}};
        long[][] result = power(base, n);
        return result[0][1];
    }

    public static void main(String[] args) {
        System.out.println(fibonacci(10)); // 55
    }
}
```

Q27. Simulate a column-oriented database storage.

```
import java.util.*;
public class ColumnStore {
    Map<String, List<Object>> columns = new HashMap<>();

    public void addColumn(String name, Object[] values) {
        columns.put(name, new ArrayList<>(Arrays.asList(values)));
    }

    public List<Object> getColumn(String name) {
        return columns.get(name);
    }
```

```
    public static void main(String[] args) {
        ColumnStore db = new ColumnStore();
        db.addColumn("id", new Integer[]{1, 2, 3});
        db.addColumn("name", new String[]{"A", "B", "C"});
        System.out.println("IDs: " + db.getColumn("id"));
    }
}
```

Q28. Compute the condition number of a matrix (simplified).

```
public class ConditionNumber {
    // For diagonal matrix, cond = max(|λ|) / min(|λ|)
    public static void main(String[] args) {
        double[] eigenvalues = {4.0, 1.0}; // From A^T A
        double max = 4.0, min = 1.0;
        double cond = max / min;
        System.out.println("Condition number: " + cond);
    }
}
```

Q29. Implement a cache with time-based expiration.

```
import java.util.*;
public class TimedCache {
    static class Entry {
        String value;
        long expiryTime;
        Entry(String v, long ttl) {
            value = v;
            expiryTime = System.currentTimeMillis() + ttl;
        }
    }

    private Map<String, Entry> cache = new HashMap<>();

    public void put(String key, String value, long ttlMillis) {
        cache.put(key, new Entry(value, ttlMillis));
    }

    public String get(String key) {
        Entry e = cache.get(key);
```

```
        if (e == null) return null;
        if (System.currentTimeMillis() > e.expiryTime) {
            cache.remove(key);
            return null;
        }
        return e.value;
    }

    public static void main(String[] args) throws InterruptedException {
        TimedCache cache = new TimedCache();
        cache.put("key", "value", 1000); // 1 sec
        System.out.println(cache.get("key")); // value
        Thread.sleep(1500);
        System.out.println(cache.get("key")); // null
    }
}
```

Q30. Simulate a distributed matrix multiplication (sharded).

```
public class DistributedMatrixMultiply {
    // Simulate sharding A by rows, B by columns
    public static void main(String[] args) {
        double[][] A = {{1, 2}, {3, 4}};
        double[][] B = {{5, 6}, {7, 8}};
        // Shard 1: A[0] * B
        double[] row0 = A[0];
        double[] result0 = new double[2];
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                result0[j] += row0[k] * B[k][j];
            }
        }
        // Shard 2: A[1] * B
        double[] row1 = A[1];
        double[] result1 = new double[2];
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                result1[j] += row1[k] * B[k][j];
            }
        }
```

```
        System.out.println(Arrays.toString(result0)); // [19.0, 22.0]
        System.out.println(Arrays.toString(result1)); // [43.0, 50.0]
    }
}
```

TOPIC 11: Recursion using Java and its Applications

Easy MCQs (10)

1. What is recursion?
   A) A loop
   B) A function that calls itself
   C) A compiler directive
   D) A memory leak
   Answer: B) A function that calls itself

2. Every recursive function must have:
   A) A loop
   B) A base case
   C) A return type
   D) Global variables
   Answer: B) A base case

3. What is the output?

```
static void print(int n) {
    if (n == 0) return;
    System.out.print(n + " ");
    print(n - 1);
}
// Called as print(3);
```

   A) `3 2 1`
   B) `1 2 3`
   C) `3 2 1 0`
   D) Infinite loop
   Answer: A) `3 2 1`

4. Which problem is naturally recursive?

A) Sum of array
B) Factorial
C) Linear search
D) All of the above
Answer: D) All of the above

5. What happens without a base case?
A) Compilation error
B) StackOverflowError
C) Returns 0
D) Skips recursion
Answer: B) StackOverflowError

6. Recursion uses which memory structure?
A) Heap
B) Stack
C) Queue
D) Register
Answer: B) Stack

7. What is the base case for factorial?
A) `n == 1`
B) `n <= 1`
C) `n == 0`
D) Both B and C
Answer: D) Both B and C

8. Which is a recursive data structure?
A) Array
B) Linked List
C) Stack
D) Queue
Answer: B) Linked List

9. What is printed?

```
static int f(int n) {
    if (n <= 1) return 1;
    return n * f(n - 1);
```

```
}
// f(4)
```

A) 4
B) 12
C) 24
D) 6
Answer: C) 24

10. Recursion is an alternative to:
A) Conditional statements
B) Iteration
C) Exception handling
D) Inheritance
Answer: B) Iteration

Medium MCQs (10)

1. What is tail recursion?
A) Recursion with two calls
B) Recursive call is the last operation
C) Recursion in main method
D) Recursion with arrays
Answer: B) Recursive call is the last operation

2. Why is recursion inefficient for Fibonacci (naive)?
A) Uses too much heap
B) Repeated subproblems
C) No base case
D) JVM bug
Answer: B) Repeated subproblems

3. What is the output?

```
static void rev(String s) {
    if (s.length() == 0) return;
    rev(s.substring(1));
    System.out.print(s.charAt(0));
}
```

```
// rev("abc")
```

A) `abc`
B) `cba`
C) `bac`
D) `acb`
Answer: B) `cba`

4. Which sorting algorithm is recursive?
A) Bubble Sort
B) Selection Sort
C) Quick Sort
D) Insertion Sort
Answer: C) Quick Sort

5. What is the time complexity of naive recursive Fibonacci?
A) $O(n)$
B) $O(\log n)$
C) $O(2^n)$
D) $O(n^2)$
Answer: C) $O(2^n)$

6. In recursion, the call stack stores:
A) Only return values
B) Local variables and return addresses
C) Global state
D) Heap references
Answer: B) Local variables and return addresses

7. What is printed?

```
static int sum(int n) {
   if (n == 0) return 0;
   return n + sum(n - 1);
}
// sum(3)
```

A) 3
B) 6

C) 9
D) 0
Answer: B) 6

8. Which is NOT a recursive application?
A) Directory traversal
B) Tower of Hanoi
C) Linear search
D) Binary search
Answer: C) Linear search *(though it can be, it's not natural)*

9. What is the space complexity of recursive factorial?
A) O(1)
B) O(log n)
C) O(n)
D) $O(n^2)$
Answer: C) O(n)

10. Mutual recursion involves:
A) One function
B) Two functions calling each other
C) Loop inside recursion
D) Static methods only
Answer: B) Two functions calling each other

Hard MCQs (10)

1. Which JVM option increases stack size for deep recursion?
A) `-Xmx`
B) `-Xss`
C) `-Xms`
D) `-XX:+UseGC`
Answer: B) `-Xss`

2. What is the output?

```
static int f(int n) {
    if (n <= 1) return n;
    return f(n - 1) + f(n - 2);
```

}
// f(5)

A) 5
B) 8
C) 13
D) 21
Answer: A) 5
(f(0)=0, f(1)=1, f(2)=1, f(3)=2, f(4)=3, f(5)=5)

3. In tail-recursive factorial, why is it optimized in some languages?
A) No stack growth
B) Faster multiplication
C) Uses heap
D) JVM supports it
Answer: A) No stack growth
(Note: Java does NOT optimize tail recursion, but the concept is tested)

4. What is the recurrence for binary search recursion?
A) T(n) = T(n-1) + O(1)
B) T(n) = 2T(n/2) + O(1)
C) T(n) = T(n/2) + O(1)
D) T(n) = T(n) + O(1)
Answer: C) T(n) = T(n/2) + O(1)

5. Which problem has overlapping subproblems?
A) Factorial
B) Fibonacci
C) Linear search
D) Array reversal
Answer: B) Fibonacci

6. What is the maximum depth of recursion in Java (default stack)?
A) 100
B) 1,000
C) 10,000–20,000
D) Unlimited
Answer: C) 10,000–20,000

7. What is printed?

```
static void f(int n) {
    if (n == 0) return;
    System.out.print(n + " ");
    f(n - 1);
    System.out.print(n + " ");
}
// f(2)
```

A) `2 1 1 2`
B) `2 1 2 1`
C) `1 2 2 1`
D) `2 2 1 1`
Answer: A) `2 1 1 2`

8. Recursive descent parsing is used in:
A) Compilers
B) Databases
C) Graphics
D) Caching
Answer: A) Compilers

9. What is the time complexity of recursive Tower of Hanoi?
A) O(n)
B) O(n log n)
C) $O(2^n)$
D) $O(n^2)$
Answer: C) $O(2^n)$

10. Which is true about recursion vs iteration?
A) Recursion is always faster
B) Iteration uses more memory
C) Recursion is more readable for tree problems
D) JVM optimizes all recursion
Answer: C) Recursion is more readable for tree problems

Coding Problems

Easy Coding (10)

Q1. Compute factorial recursively.
```
public class Factorial {
    static long fact(int n) {
        if (n <= 1) return 1L;
        return n * fact(n - 1);
    }
    public static void main(String[] args) {
        System.out.println(fact(5)); // 120
    }
}
```

Q2. Compute sum of first n natural numbers.
```
public class SumN {
    static int sum(int n) {
        if (n == 0) return 0;
        return n + sum(n - 1);
    }
    public static void main(String[] args) {
        System.out.println(sum(4)); // 10
    }
}
```

Q3. Print numbers from n to 1.
```
public class Countdown {
    static void print(int n) {
        if (n == 0) return;
        System.out.println(n);
        print(n - 1);
    }
    public static void main(String[] args) {
        print(3);
    }
}
```

Q4. Check if a string is palindrome.
```
public class Palindrome {
    static boolean isPalin(String s, int l, int r) {
```

```
        if (l >= r) return true;
        if (s.charAt(l) != s.charAt(r)) return false;
        return isPalin(s, l + 1, r - 1);
    }
    public static void main(String[] args) {
        System.out.println(isPalin("aba", 0, 2)); // true
    }
}
```

Q5. Compute power: a^b.

```
public class Power {
    static long pow(int a, int b) {
        if (b == 0) return 1;
        return a * pow(a, b - 1);
    }
    public static void main(String[] args) {
        System.out.println(pow(2, 3)); // 8
    }
}
```

Q6. Count digits in a number.

```
public class DigitCount {
    static int count(int n) {
        if (n == 0) return 0;
        return 1 + count(n / 10);
    }
    public static void main(String[] args) {
        System.out.println(count(123)); // 3
    }
}
```

Q7. Find maximum in array recursively.

```
public class MaxInArray {
    static int max(int[] arr, int n) {
        if (n == 1) return arr[0];
        return Math.max(arr[n - 1], max(arr, n - 1));
    }
    public static void main(String[] args) {
        int[] arr = {3, 1, 4};
```

```
        System.out.println(max(arr, 3)); // 4
    }
}
```

Q8. Reverse a string.

```
public class ReverseString {
    static String rev(String s) {
        if (s.length() <= 1) return s;
        return rev(s.substring(1)) + s.charAt(0);
    }
    public static void main(String[] args) {
        System.out.println(rev("hello")); // "olleh"
    }
}
```

Q9. Compute GCD using Euclidean algorithm.

```
public class GCD {
    static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }
    public static void main(String[] args) {
        System.out.println(gcd(48, 18)); // 6
    }
}
```

Q10. Print Fibonacci series up to n terms.

```
public class FibSeries {
    static int fib(int n) {
        if (n <= 1) return n;
        return fib(n - 1) + fib(n - 2);
    }
    public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            System.out.print(fib(i) + " "); // 0 1 1 2 3 5
        }
    }
}
```

Medium Coding (10)

Q11. Solve Tower of Hanoi.

```
public class TowerOfHanoi {
   static void move(int n, char from, char to, char aux) {
      if (n == 1) {
         System.out.println("Move disk 1 from " + from + " to " + to);
         return;
      }
      move(n - 1, from, aux, to);
      System.out.println("Move disk " + n + " from " + from + " to " + to);
      move(n - 1, aux, to, from);
   }
   public static void main(String[] args) {
      move(3, 'A', 'C', 'B');
   }
}
```

Q12. Generate all subsets of a set.

```
import java.util.*;
public class Subsets {
   static void generate(List<Integer> current, int[] nums, int index, List<List<Integer>>
result) {
      if (index == nums.length) {
         result.add(new ArrayList<>(current));
         return;
      }
      // Exclude
      generate(current, nums, index + 1, result);
      // Include
      current.add(nums[index]);
      generate(current, nums, index + 1, result);
      current.remove(current.size() - 1);
   }
   public static void main(String[] args) {
      int[] nums = {1, 2};
      List<List<Integer>> result = new ArrayList<>();
      generate(new ArrayList<>(), nums, 0, result);
      System.out.println(result);
```

```
    }
}
```

Q13. Implement binary search recursively.

```
public class BinarySearch {
    static int search(int[] arr, int low, int high, int target) {
        if (low > high) return -1;
        int mid = (low + high) / 2;
        if (arr[mid] == target) return mid;
        else if (arr[mid] > target) return search(arr, low, mid - 1, target);
        else return search(arr, mid + 1, high, target);
    }
    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7};
        System.out.println(search(arr, 0, 3, 5)); // 2
    }
}
```

Q14. Check if array is sorted.

```
public class IsSorted {
    static boolean sorted(int[] arr, int n) {
        if (n == 1) return true;
        return (arr[n - 2] <= arr[n - 1]) && sorted(arr, n - 1);
    }
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        System.out.println(sorted(arr, 3)); // true
    }
}
```

Q15. Find the first occurrence of a value.

```
public class FirstOccurrence {
    static int first(int[] arr, int index, int target) {
        if (index == arr.length) return -1;
        if (arr[index] == target) return index;
        return first(arr, index + 1, target);
    }
    public static void main(String[] args) {
        int[] arr = {2, 4, 4, 5};
```

```
        System.out.println(first(arr, 0, 4)); // 1
    }
}
```

Q16. Compute digital root (sum digits until single digit).

```
public class DigitalRoot {
    static int root(int n) {
        if (n < 10) return n;
        int sum = 0;
        while (n > 0) {
            sum += n % 10;
            n /= 10;
        }
        return root(sum);
    }
    public static void main(String[] args) {
        System.out.println(root(942)); // 6 (9+4+2=15 → 1+5=6)
    }
}
```

Q17. Print all permutations of a string.

```
public class Permutations {
    static void permute(String str, String ans) {
        if (str.length() == 0) {
            System.out.println(ans);
            return;
        }
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            String ros = str.substring(0, i) + str.substring(i + 1);
            permute(ros, ans + ch);
        }
    }
    public static void main(String[] args) {
        permute("ABC", "");
    }
}
```

Q18. Count paths in a grid (only right/down).

```
public class GridPaths {
    static int paths(int m, int n) {
        if (m == 1 || n == 1) return 1;
        return paths(m - 1, n) + paths(m, n - 1);
    }
    public static void main(String[] args) {
        System.out.println(paths(3, 3)); // 6
    }
}
```

Q19. Check if a number is prime recursively.

```
public class PrimeCheck {
    static boolean isPrime(int n, int i) {
        if (n <= 2) return n == 2;
        if (n % i == 0) return false;
        if (i * i > n) return true;
        return isPrime(n, i + 1);
    }
    public static void main(String[] args) {
        System.out.println(isPrime(17, 2)); // true
    }
}
```

Q20. Merge two sorted arrays recursively.

```
public class MergeArrays {
    static void merge(int[] a, int[] b, int[] c, int i, int j, int k) {
        if (i == a.length) {
            while (j < b.length) c[k++] = b[j++];
            return;
        }
        if (j == b.length) {
            while (i < a.length) c[k++] = a[i++];
            return;
        }
        if (a[i] <= b[j]) {
            c[k] = a[i];
            merge(a, b, c, i + 1, j, k + 1);
        } else {
            c[k] = b[j];
```

```
        merge(a, b, c, i, j + 1, k + 1);
      }
    }
    public static void main(String[] args) {
      int[] a = {1, 3}, b = {2, 4};
      int[] c = new int[4];
      merge(a, b, c, 0, 0, 0);
      for (int x : c) System.out.print(x + " "); // 1 2 3 4
    }
}
```

Hard Coding (10)

Q21. Solve N-Queens problem (print all solutions).

```
public class NQueensAll {
    static boolean isSafe(int[][] board, int row, int col, int n) {
        for (int i = 0; i < col; i++) if (board[row][i] == 1) return false;
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) if (board[i][j] == 1) return false;
        for (int i = row, j = col; i < n && j >= 0; i++, j--) if (board[i][j] == 1) return false;
        return true;
    }

    static boolean solve(int[][] board, int col, int n) {
        if (col == n) {
            printBoard(board, n);
            return true;
        }
        boolean res = false;
        for (int i = 0; i < n; i++) {
            if (isSafe(board, i, col, n)) {
                board[i][col] = 1;
                res = solve(board, col + 1, n) || res;
                board[i][col] = 0;
            }
        }
        return res;
    }

    static void printBoard(int[][] board, int n) {
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n = 4;
        int[][] board = new int[n][n];
        solve(board, 0, n);
    }
}
```

Q22. Implement recursive quick sort.

```
public class QuickSort {
    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    static void sort(int[] arr, int low, int high) {
        if (low < high) {
```

```
            int pi = partition(arr, low, high);
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }

    public static void main(String[] args) {
        int[] arr = {3, 1, 4, 1, 5};
        sort(arr, 0, 4);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q23. Generate all valid parentheses combinations.

```
import java.util.*;
public class ValidParentheses {
    static void generate(List<String> result, String current, int open, int close, int max) {
        if (current.length() == max * 2) {
            result.add(current);
            return;
        }
        if (open < max) generate(result, current + "(", open + 1, close, max);
        if (close < open) generate(result, current + ")", open, close + 1, max);
    }

    public static void main(String[] args) {
        List<String> result = new ArrayList<>();
        generate(result, "", 0, 0, 3);
        System.out.println(result);
    }
}
```

Q24. Solve Sudoku puzzle.

```
public class SudokuSolver {
    static boolean isSafe(int[][] board, int row, int col, int num) {
        for (int x = 0; x < 9; x++) if (board[row][x] == num) return false;
        for (int x = 0; x < 9; x++) if (board[x][col] == num) return false;
        int startRow = row - row % 3, startCol = col - col % 3;
        for (int i = 0; i < 3; i++)
```

```
        for (int j = 0; j < 3; j++)
            if (board[i + startRow][j + startCol] == num) return false;
    return true;
}

static boolean solve(int[][] board) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (board[i][j] == 0) {
                for (int num = 1; num <= 9; num++) {
                    if (isSafe(board, i, j, num)) {
                        board[i][j] = num;
                        if (solve(board)) return true;
                        board[i][j] = 0;
                    }
                }
                return false;
            }
        }
    }
    return true;
}

public static void main(String[] args) {
    int[][] board = {
        {5,3,0,0,7,0,0,0,0},
        {6,0,0,1,9,5,0,0,0},
        {0,9,8,0,0,0,0,6,0},
        {8,0,0,0,6,0,0,0,3},
        {4,0,0,8,0,3,0,0,1},
        {7,0,0,0,2,0,0,0,6},
        {0,6,0,0,0,0,2,8,0},
        {0,0,0,4,1,9,0,0,5},
        {0,0,0,0,8,0,0,7,9}
    };
    if (solve(board)) {
        for (int[] row : board) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
```

```
            }
        }
    }
}
```

Q25. Compute binomial coefficient C(n, k) recursively.

```
public class BinomialCoeff {
    static int C(int n, int k) {
        if (k == 0 || k == n) return 1;
        return C(n - 1, k - 1) + C(n - 1, k);
    }
    public static void main(String[] args) {
        System.out.println(C(5, 2)); // 10
    }
}
```

Q26. Find all paths from root to leaf in binary tree (simulated with array).

```
import java.util.*;
public class RootToLeafPaths {
    static void paths(int[] tree, int index, List<Integer> current, List<List<Integer>> result)
{
        if (index >= tree.length || tree[index] == -1) return;
        current.add(tree[index]);
        // Check if leaf (no children)
        int left = 2 * index + 1, right = 2 * index + 2;
        if ((left >= tree.length || tree[left] == -1) &&
            (right >= tree.length || tree[right] == -1)) {
            result.add(new ArrayList<>(current));
        } else {
            paths(tree, left, current, result);
            paths(tree, right, current, result);
        }
        current.remove(current.size() - 1);
    }

    public static void main(String[] args) {
        int[] tree = {1, 2, 3, -1, 5, -1, -1}; // -1 = null
        List<List<Integer>> result = new ArrayList<>();
        paths(tree, 0, new ArrayList<>(), result);
```

```
        System.out.println(result);
    }
}
```

Q27. Implement recursive merge sort.

```
public class MergeSort {
    static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, l, L, 0, n1);
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    static void sort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void main(String[] args) {
        int[] arr = {3, 1, 4, 1, 5};
        sort(arr, 0, 4);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q28. Solve the knapsack problem (0/1) recursively.

```
public class Knapsack {
    static int knapSack(int W, int[] wt, int[] val, int n) {
```

```
        if (n == 0 || W == 0) return 0;
        if (wt[n - 1] > W) return knapSack(W, wt, val, n - 1);
        else return Math.max(
            val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),
            knapSack(W, wt, val, n - 1)
        );
    }

    public static void main(String[] args) {
        int[] val = {60, 100, 120};
        int[] wt = {10, 20, 30};
        int W = 50;
        System.out.println(knapSack(W, wt, val, 3)); // 220
    }
}
```

Q29. Generate all combinations of k numbers from 1..n.

```
import java.util.*;
public class Combinations {
    static void combine(int n, int k, int start, List<Integer> current, List<List<Integer>>
result) {
        if (k == 0) {
            result.add(new ArrayList<>(current));
            return;
        }
        for (int i = start; i <= n - k + 1; i++) {
            current.add(i);
            combine(n, k - 1, i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }

    public static void main(String[] args) {
        List<List<Integer>> result = new ArrayList<>();
        combine(4, 2, 1, new ArrayList<>(), result);
        System.out.println(result);
    }
}
```

Q30. Implement recursive depth-first search (DFS) on graph.

```
import java.util.*;
public class DFSRecursive {
    static void dfs(Map<Integer, List<Integer>> graph, int node, Set<Integer> visited) {
        visited.add(node);
        System.out.print(node + " ");
        for (int neighbor : graph.getOrDefault(node, new ArrayList<>())) {
            if (!visited.contains(neighbor)) {
                dfs(graph, neighbor, visited);
            }
        }
    }

    public static void main(String[] args) {
        Map<Integer, List<Integer>> graph = new HashMap<>();
        graph.put(0, Arrays.asList(1, 2));
        graph.put(1, Arrays.asList(3));
        graph.put(2, Arrays.asList(3));
        dfs(graph, 0, new HashSet<>());
    }
}
```

TOPIC 12: Linear and Binary Search

Easy MCQs (10)

1. Linear search works on:
   A) Sorted arrays only
   B) Unsorted arrays
   C) Only linked lists
   D) Only trees
   Answer: B) Unsorted arrays

2. Binary search requires the array to be:
   A) Unsorted
   B) Sorted
   C) Reversed
   D) Jagged
   Answer: B) Sorted

3. What is the worst-case time complexity of linear search?
   A) O(1)
   B) O(log n)
   C) O(n)
   D) $O(n^2)$
   Answer: C) O(n)

4. What is the best-case time complexity of binary search?
   A) O(1)
   B) O(log n)
   C) O(n)
   D) O(n log n)
   Answer: A) O(1)

5. Which search checks elements one by one?
   A) Binary
   B) Linear
   C) Jump
   D) Interpolation
   Answer: B) Linear

6. In binary search, the search space is reduced by:
   A) 1 element
   B) Half
   C) One-third
   D) Random amount
   Answer: B) Half

7. Linear search is also known as:
   A) Sequential search
   B) Jump search
   C) Exponential search
   D) Hash search
   Answer: A) Sequential search

8. What does linear search return if the element is not found?
   A) `null`
   B) `-1`

C) `0`
D) Exception
Answer: B) `-1`

9. Binary search uses which divide strategy?
A) Divide by 3
B) Divide by 2
C) Divide by n
D) No division
Answer: B) Divide by 2

10. Which search is simpler to implement?
A) Binary
B) Linear
C) Both same
D) Neither
Answer: B) Linear

Medium MCQs (10)

1. What is the time complexity of binary search on a sorted array of size n?
A) O(1)
B) O(log n)
C) O(n)
D) O(n log n)
Answer: B) O(log n)

2. Why can't binary search be used on a linked list efficiently?
A) No random access
B) Too slow
C) Memory issue
D) Not sorted
Answer: A) No random access

3. What is the output of binary search for key=5 in [1,3,5,7,9]?
A) 0
B) 1
C) 2
D) -1

Answer: C) 2

4. In linear search, average number of comparisons is:
   A) n
   B) n/2
   C) log n
   D) 1
   Answer: B) n/2

5. Which condition must hold for binary search to work?
   A) Array must be of even length
   B) Elements must be comparable and sorted
   C) Only positive numbers
   D) Must be 2D array
   Answer: B) Elements must be comparable and sorted

6. What is the space complexity of iterative binary search?
   A) O(1)
   B) O(log n)
   C) O(n)
   D) $O(n^2)$
   Answer: A) O(1)

7. What is the main disadvantage of linear search?
   A) Requires sorting
   B) Slow for large datasets
   C) Complex code
   D) Uses extra memory
   Answer: B) Slow for large datasets

8. In binary search, `mid = (low + high) / 2` can cause:
   A) Overflow
   B) Underflow
   C) Division by zero
   D) No issue
   Answer: A) Overflow
   (Safe: `mid = low + (high - low) / 2`)

9. Which search is used in `Arrays.binarySearch()`?

A) Linear
B) Binary
C) Hash
D) Tree
Answer: B) Binary

10. Linear search is preferred when:
A) Array is large and sorted
B) Array is small or unsorted
C) Memory is limited
D) Speed is critical
Answer: B) Array is small or unsorted

Hard MCQs (10)

1. What is the recurrence relation for binary search?
A) $T(n) = T(n-1) + O(1)$
B) $T(n) = 2T(n/2) + O(1)$
C) $T(n) = T(n/2) + O(1)$
D) $T(n) = T(n) + O(1)$
Answer: C) $T(n) = T(n/2) + O(1)$

2. In a rotated sorted array, binary search can be modified to run in:
A) O(n)
B) O(log n)
C) O(n log n)
D) O(1)
Answer: B) O(log n)

3. The decision tree height for binary search on n elements is:
A) n
B) $\log_2 n$
C) n log n
D) 2^n
Answer: B) $\log_2 n$

4. Which is true about interpolation search?
A) Always faster than binary
B) O(log log n) for uniform data

C) Works on unsorted data
D) Uses hashing
Answer: B) O(log log n) for uniform data

5. What is the minimum number of comparisons to find an element in worst-case binary search for n=8?
A) 1
B) 3
C) 4
D) 8
Answer: B) 3
*($\lceil \log_2(8+1) \rceil = 4$? Wait: for n=8, max comparisons = $\lfloor \log_2 n \rfloor + 1 = 3 + 1 = 4$. But standard: $8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow$ found in 3 or 4. Let's clarify:
Array of 8: indices 0-7.
Step1: mid=3
Step2: mid=1 or 5
Step3: mid=0,2,4,6
Step4: leaves
So worst-case = 4. But many sources say $\lfloor \log_2 n \rfloor + 1 = 3 + 1 = 4$. So Answer: C) 4)*

6. Exponential search is useful when:
A) Array size is unknown
B) Array is unsorted
C) Only first element is known
D) Memory is full
Answer: A) Array size is unknown

7. In binary search, if `low > high`, it means:
A) Element found
B) Element not found
C) Array is empty
D) Infinite loop
Answer: B) Element not found

8. Which search algorithm is stable?
A) Binary search
B) Linear search
C) Both
D) Neither

Answer: B) Linear search
(Stable: preserves order of equal elements; binary doesn't apply as it assumes distinct or doesn't care)

9. The information-theoretic lower bound for comparison-based search is:
A) O(1)
B) O(log n)
C) O(n)
D) O(n log n)
Answer: B) O(log n)

10. In a database index, binary search is used on:
A) Heap files
B) Sorted files or B-trees
C) Hash buckets
D) Log files
Answer: B) Sorted files or B-trees

Coding Problems

Easy Coding (10)

Q1. Implement linear search for an integer array.

```
public class LinearSearch {
    public static int search(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) return i;
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {10, 20, 30};
        System.out.println(search(arr, 20)); // 1
    }
}
```

Q2. Implement binary search (iterative).

```
public class BinarySearchIterative {
```

```
    public static int search(int[] arr, int target) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) return mid;
            else if (arr[mid] < target) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7};
        System.out.println(search(arr, 5)); // 2
    }
}
```

Q3. Check if an array contains a target (linear).

```
public class Contains {
    public static boolean contains(int[] arr, int target) {
        for (int x : arr) {
            if (x == target) return true;
        }
        return false;
    }
    public static void main(String[] args) {
        System.out.println(contains(new int[]{1,2,3}, 2)); // true
    }
}
```

Q4. Find the first occurrence of a target (linear).

```
public class FirstOccurrence {
    public static int first(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) return i;
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {2, 4, 4, 5};
```

```
        System.out.println(first(arr, 4)); // 1
    }
}
```

Q5. Find the last occurrence of a target (linear).

```
public class LastOccurrence {
    public static int last(int[] arr, int target) {
        for (int i = arr.length - 1; i >= 0; i--) {
            if (arr[i] == target) return i;
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {2, 4, 4, 5};
        System.out.println(last(arr, 4)); // 2
    }
}
```

Q6. Count occurrences of a target (linear).

```
public class CountOccurrences {
    public static int count(int[] arr, int target) {
        int c = 0;
        for (int x : arr) if (x == target) c++;
        return c;
    }
    public static void main(String[] args) {
        System.out.println(count(new int[]{1,2,2,3}, 2)); // 2
    }
}
```

Q7. Search in a string array (linear).

```
public class StringSearch {
    public static int search(String[] arr, String target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i].equals(target)) return i;
        }
        return -1;
    }
    public static void main(String[] args) {
```

```
        String[] arr = {"apple", "banana"};
        System.out.println(search(arr, "banana")); // 1
    }
}
```

Q8. Find minimum element in unsorted array (linear).

```
public class MinElement {
    public static int min(int[] arr) {
        int m = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < m) m = arr[i];
        }
        return m;
    }
    public static void main(String[] args) {
        System.out.println(min(new int[]{3,1,4})); // 1
    }
}
```

Q9. Check if array is sorted (linear scan).

```
public class IsSorted {
    public static boolean sorted(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < arr[i-1]) return false;
        }
        return true;
    }
    public static void main(String[] args) {
        System.out.println(sorted(new int[]{1,2,3})); // true
    }
}
```

Q10. Search in a 2D row-wise sorted matrix (linear per row).

```
public class Search2DLinear {
    public static boolean search(int[][] mat, int target) {
        for (int[] row : mat) {
            for (int x : row) {
                if (x == target) return true;
            }
```

```java
        }
        return false;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,2},{3,4}};
        System.out.println(search(mat, 3)); // true
    }
}
```

Medium Coding (10)

Q11. Implement binary search (recursive).

```java
public class BinarySearchRecursive {
    public static int search(int[] arr, int low, int high, int target) {
        if (low > high) return -1;
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) return mid;
        else if (arr[mid] < target) return search(arr, mid + 1, high, target);
        else return search(arr, low, mid - 1, target);
    }
    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7};
        System.out.println(search(arr, 0, 3, 5)); // 2
    }
}
```

Q12. Find the first occurrence in a sorted array with duplicates.

```java
public class FirstOccurrenceSorted {
    public static int first(int[] arr, int target) {
        int low = 0, high = arr.length - 1, result = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) {
                result = mid;
                high = mid - 1; // continue left
            } else if (arr[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
```

```
            }
        }
        return result;
    }
    public static void main(String[] args) {
        int[] arr = {2, 4, 4, 5};
        System.out.println(first(arr, 4)); // 1
    }
}
```

Q13. Find the last occurrence in a sorted array with duplicates.

```
public class LastOccurrenceSorted {
    public static int last(int[] arr, int target) {
        int low = 0, high = arr.length - 1, result = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) {
                result = mid;
                low = mid + 1; // continue right
            } else if (arr[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return result;
    }
    public static void main(String[] args) {
        int[] arr = {2, 4, 4, 5};
        System.out.println(last(arr, 4)); // 2
    }
}
```

Q14. Count occurrences in sorted array using binary search.

```
public class CountInSorted {
    public static int count(int[] arr, int target) {
        int first = first(arr, target);
        if (first == -1) return 0;
        int last = last(arr, target);
```

```
        return last - first + 1;
    }
    // Include first() and last() methods from Q12, Q13
    public static void main(String[] args) {
        int[] arr = {2, 4, 4, 5};
        System.out.println(count(arr, 4)); // 2
    }
}
```

Q15. Search in a rotated sorted array (no duplicates).

```
public class SearchRotated {
    public static int search(int[] arr, int target) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) return mid;
            if (arr[low] <= arr[mid]) { // Left sorted
                if (arr[low] <= target && target < arr[mid]) high = mid - 1;
                else low = mid + 1;
            } else { // Right sorted
                if (arr[mid] < target && target <= arr[high]) low = mid + 1;
                else high = mid - 1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {4,5,6,7,0,1,2};
        System.out.println(search(arr, 0)); // 4
    }
}
```

Q16. Find the peak element in an array (greater than neighbors).

```
public class PeakElement {
    public static int findPeak(int[] arr) {
        int low = 0, high = arr.length - 1;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] > arr[mid + 1]) high = mid;
```

```
            else low = mid + 1;
        }
        return low;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,3,1};
        System.out.println(findPeak(arr)); // 2
    }
}
```

Q17. Find the insertion position in sorted array (lower bound).

```
public class InsertPosition {
    public static int searchInsert(int[] arr, int target) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] < target) low = mid + 1;
            else high = mid - 1;
        }
        return low;
    }
    public static void main(String[] args) {
        int[] arr = {1,3,5,6};
        System.out.println(searchInsert(arr, 5)); // 2
        System.out.println(searchInsert(arr, 2)); // 1
    }
}
```

Q18. Search in a 2D matrix where each row and column is sorted.

```
public class Search2DSorted {
    public static boolean searchMatrix(int[][] matrix, int target) {
        if (matrix.length == 0) return false;
        int row = 0, col = matrix[0].length - 1;
        while (row < matrix.length && col >= 0) {
            if (matrix[row][col] == target) return true;
            else if (matrix[row][col] > target) col--;
            else row++;
        }
        return false;
```

```
    }
    public static void main(String[] args) {
        int[][] mat = {{1,4,7,11},{2,5,8,12},{3,6,9,16}};
        System.out.println(searchMatrix(mat, 5)); // true
    }
}
```

Q19. Find the square root of x using binary search.

```
public class SqrtBinary {
    public static int mySqrt(int x) {
        if (x == 0) return 0;
        long low = 1, high = x;
        while (low <= high) {
            long mid = low + (high - low) / 2;
            if (mid * mid == x) return (int) mid;
            else if (mid * mid < x) low = mid + 1;
            else high = mid - 1;
        }
        return (int) high;
    }
    public static void main(String[] args) {
        System.out.println(mySqrt(8)); // 2
    }
}
```

Q20. Find the smallest letter greater than target (circular).

```
public class NextGreatestLetter {
    public static char nextGreatestLetter(char[] letters, char target) {
        int low = 0, high = letters.length;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (letters[mid] <= target) low = mid + 1;
            else high = mid;
        }
        return letters[low % letters.length];
    }
    public static void main(String[] args) {
        char[] letters = {'c','f','j'};
        System.out.println(nextGreatestLetter(letters, 'a')); // 'c'
```

```
    }
}
```

Hard Coding (10)

Q21. Find the minimum in a rotated sorted array (with duplicates).

```
public class MinInRotatedWithDup {
    public static int findMin(int[] nums) {
        int low = 0, high = nums.length - 1;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] > nums[high]) low = mid + 1;
            else if (nums[mid] < nums[high]) high = mid;
            else high--; // skip duplicate
        }
        return nums[low];
    }
    public static void main(String[] args) {
        int[] nums = {2,2,2,0,1};
        System.out.println(findMin(nums)); // 0
    }
}
```

Q22. Find the k-th smallest element in a sorted matrix.

```
public class KthSmallestInMatrix {
    public static int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        int low = matrix[0][0], high = matrix[n-1][n-1];
        while (low < high) {
            int mid = low + (high - low) / 2;
            int count = 0, j = n - 1;
            for (int i = 0; i < n; i++) {
                while (j >= 0 && matrix[i][j] > mid) j--;
                count += j + 1;
            }
            if (count < k) low = mid + 1;
            else high = mid;
        }
        return low;
```

```
    }
    public static void main(String[] args) {
        int[][] mat = {{1,5,9},{10,11,13},{12,13,15}};
        System.out.println(kthSmallest(mat, 8)); // 13
    }
}
```

Q23. Search for a range (first and last position of target).

```
public class SearchRange {
    public static int[] searchRange(int[] nums, int target) {
        int first = first(nums, target);
        int last = last(nums, target);
        return new int[]{first, last};
    }
    // Include first() and last() from Q12, Q13
    public static void main(String[] args) {
        int[] nums = {5,7,7,8,8,10};
        int[] res = searchRange(nums, 8);
        System.out.println(res[0] + " " + res[1]); // 3 4
    }
}
```

Q24. Find the median of two sorted arrays (logarithmic time).

```
public class MedianOfTwoSorted {
    public static double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) return findMedianSortedArrays(nums2, nums1);
        int x = nums1.length, y = nums2.length;
        int low = 0, high = x;
        while (low <= high) {
            int partitionX = (low + high) / 2;
            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxX = (partitionX == 0) ? Integer.MIN_VALUE : nums1[partitionX - 1];
            int minX = (partitionX == x) ? Integer.MAX_VALUE : nums1[partitionX];
            int maxY = (partitionY == 0) ? Integer.MIN_VALUE : nums2[partitionY - 1];
            int minY = (partitionY == y) ? Integer.MAX_VALUE : nums2[partitionY];

            if (maxX <= minY && maxY <= minX) {
                if ((x + y) % 2 == 0) {
```

```
                return (Math.max(maxX, maxY) + Math.min(minX, minY)) / 2.0;
            } else {
                return Math.max(maxX, maxY);
            }
        } else if (maxX > minY) {
            high = partitionX - 1;
        } else {
            low = partitionX + 1;
        }
    }
    throw new IllegalArgumentException();
  }
  public static void main(String[] args) {
    int[] a = {1,3}, b = {2};
    System.out.println(findMedianSortedArrays(a, b)); // 2.0
  }
}
```

Q25. Find the single element in a sorted array where every other appears twice.

```
public class SingleElementInSorted {
  public static int singleNonDuplicate(int[] nums) {
    int low = 0, high = nums.length - 1;
    while (low < high) {
      int mid = low + (high - low) / 2;
      if (mid % 2 == 1) mid--; // ensure even index
      if (nums[mid] == nums[mid + 1]) low = mid + 2;
      else high = mid;
    }
    return nums[low];
  }
  public static void main(String[] args) {
    int[] nums = {1,1,2,3,3,4,4,8,8};
    System.out.println(singleNonDuplicate(nums)); // 2
  }
}
```

Q26. Find the first bad version (API: isBadVersion).

```
public class FirstBadVersion {
  // Simulated API
```

```
    static boolean isBadVersion(int version) {
        return version >= 4; // assume 4 is first bad
    }

    public static int firstBadVersion(int n) {
        int low = 1, high = n;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (isBadVersion(mid)) high = mid;
            else low = mid + 1;
        }
        return low;
    }
    public static void main(String[] args) {
        System.out.println(firstBadVersion(5)); // 4
    }
}
```

Q27. Find the smallest divisor given a threshold.

```
public class SmallestDivisor {
    public static int smallestDivisor(int[] nums, int threshold) {
        int low = 1, high = 1000000;
        while (low < high) {
            int mid = low + (high - low) / 2;
            int sum = 0;
            for (int x : nums) sum += (x + mid - 1) / mid; // ceil division
            if (sum <= threshold) high = mid;
            else low = mid + 1;
        }
        return low;
    }
    public static void main(String[] args) {
        int[] nums = {1,2,5,9};
        System.out.println(smallestDivisor(nums, 6)); // 5
    }
}
```

Q28. Find the k closest elements to x in a sorted array.

```
import java.util.*;
```

```
public class FindClosestElements {
    public static List<Integer> findClosestElements(int[] arr, int k, int x) {
        int low = 0, high = arr.length - k;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (x - arr[mid] > arr[mid + k] - x) low = mid + 1;
            else high = mid;
        }
        List<Integer> result = new ArrayList<>();
        for (int i = low; i < low + k; i++) result.add(arr[i]);
        return result;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5};
        System.out.println(findClosestElements(arr, 4, 3)); // [1,2,3,4]
    }
}
```

Q29. Find the minimum element in a sorted rotated array (no duplicates).

```
public class MinInRotated {
    public static int findMin(int[] nums) {
        int low = 0, high = nums.length - 1;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] > nums[high]) low = mid + 1;
            else high = mid;
        }
        return nums[low];
    }
    public static void main(String[] args) {
        int[] nums = {3,4,5,1,2};
        System.out.println(findMin(nums)); // 1
    }
}
```

Q30. Allocate books to students (minimize maximum pages).

```
public class AllocateBooks {
    public static int books(int[] A, int B) {
        if (B > A.length) return -1;
```

```
        long low = 0, high = 0;
        for (int x : A) {
            low = Math.max(low, x);
            high += x;
        }
        while (low < high) {
            long mid = low + (high - low) / 2;
            int students = 1, pages = 0;
            for (int x : A) {
                if (pages + x > mid) {
                    students++;
                    pages = x;
                } else {
                    pages += x;
                }
            }
            if (students > B) low = mid + 1;
            else high = mid;
        }
        return (int) low;
    }
    public static void main(String[] args) {
        int[] A = {12, 34, 67, 90};
        System.out.println(books(A, 2)); // 113
    }
}
```

TOPIC 13: Complexity Analysis and Problems

Easy MCQs (10)

1. What does Big O notation describe?
   A) Best-case performance
   B) Worst-case performance
   C) Average-case performance
   D) Exact runtime
   Answer: B) Worst-case performance

2. O(1) means:

A) Linear time
B) Constant time
C) Logarithmic time
D) Quadratic time
Answer: B) Constant time

3. Accessing an array element by index is:
A) O(n)
B) O(log n)
C) O(1)
D) $O(n^2)$
Answer: C) O(1)

4. A single loop from 1 to n has complexity:
A) O(1)
B) O(log n)
C) O(n)
D) $O(n^2)$
Answer: C) O(n)

5. Which is the slowest growth rate?
A) O(1)
B) O(log n)
C) O(n)
D) $O(n^2)$
Answer: D) $O(n^2)$

6. Binary search has time complexity:
A) O(n)
B) O(log n)
C) O(n log n)
D) O(1)
Answer: B) O(log n)

7. Space complexity refers to:
A) CPU cycles
B) Memory usage
C) Disk I/O
D) Network bandwidth

Answer: B) Memory usage

8. What is the time complexity of nested loops (both 1 to n)?
A) O(n)
B) O(n log n)
C) $O(n^2)$
D) $O(2^n)$
Answer: C) $O(n^2)$

9. O(n log n) is typical for:
A) Bubble Sort
B) Quick Sort (average)
C) Linear Search
D) Array access
Answer: B) Quick Sort (average)

10. Which notation ignores constants and lower-order terms?
A) Big O
B) Big Omega
C) Big Theta
D) All of the above
Answer: D) All of the above

Medium MCQs (10)

1. What is the time complexity of the following?

```
for (int i = 1; i <= n; i *= 2) {
    // O(1) work
}
```

A) O(n)
B) O(log n)
C) O(n log n)
D) O(1)
Answer: B) O(log n)

2. The recurrence T(n) = 2T(n/2) + O(n) solves to:
A) O(n)

B) O(n log n)
C) $O(n^2)$
D) O(log n)
Answer: B) O(n log n) *(Master Theorem)*

3. Which algorithm has $O(n^2)$ worst-case time?
A) Merge Sort
B) Quick Sort
C) Heap Sort
D) Radix Sort
Answer: B) Quick Sort

4. What is the space complexity of recursive factorial?
A) O(1)
B) O(log n)
C) O(n)
D) $O(n^2)$
Answer: C) O(n)

5. Amortized analysis is used for:
A) Worst-case per operation over a sequence
B) Best-case only
C) Average input only
D) Compiler optimization
Answer: A) Worst-case per operation over a sequence

6. What is the time complexity of HashMap `get()`?
A) O(1) average
B) O(log n)
C) O(n) worst
D) Both A and C
Answer: D) Both A and C

7. Which loop has $O(\sqrt{n})$ complexity?

```
for (int i = 1; i * i <= n; i++) { }
```

A) Yes
B) No

C) O(n)
D) O(log n)
Answer: A) Yes

8. The time complexity of building a heap is:
A) O(n)
B) O(n log n)
C) O(log n)
D) O(1)
Answer: A) O(n)

9. What is the complexity of printing all subsets of a set?
A) O(n)
B) $O(n^2)$
C) $O(2^n)$
D) O(n!)
Answer: C) $O(2^n)$

10. In complexity analysis, 'n' usually represents:
A) Input size
B) Output size
C) Memory size
D) CPU speed
Answer: A) Input size

Hard MCQs (10)

1. The Master Theorem applies to recurrences of the form:
A) T(n) = aT(n/b) + f(n)
B) T(n) = T(n-1) + f(n)
C) T(n) = T(n/2) + T(n/2) + f(n)
D) All recursive functions
Answer: A) T(n) = aT(n/b) + f(n)

2. What is the time complexity of Strassen's matrix multiplication?
A) $O(n^3)$
B) $O(n^2.807)$
C) $O(n^2)$
D) O(n log n)

Answer: B) $O(n^{2.807})$

3. Which problem has $\Omega(n \log n)$ lower bound for comparison-based sorting?
   A) Searching
   B) Sorting
   C) Matrix multiplication
   D) Graph traversal
   Answer: B) Sorting

4. The space complexity of iterative binary search is:
   A) $O(1)$
   B) $O(\log n)$
   C) $O(n)$
   D) $O(n \log n)$
   Answer: A) $O(1)$

5. What is the amortized time per operation for dynamic array insertion?
   A) $O(1)$
   B) $O(\log n)$
   C) $O(n)$
   D) $O(n^2)$
   Answer: A) $O(1)$

6. Which is true about NP-complete problems?
   A) Solvable in polynomial time
   B) No known polynomial-time solution
   C) Always $O(2^n)$
   D) Not in NP
   Answer: B) No known polynomial-time solution

7. The time complexity of the Euclidean algorithm for GCD is:
   A) $O(n)$
   B) $O(\log \min(a,b))$
   C) $O(a + b)$
   D) $O(1)$
   Answer: B) $O(\log \min(a,b))$

8. In cache complexity, what does "cache-oblivious" mean?
   A) Ignores cache

B) Optimized without knowing cache parameters
C) Disables cache
D) Uses only registers
Answer: B) Optimized without knowing cache parameters

9. What is the complexity of the Floyd-Warshall algorithm?
A) $O(V^2)$
B) $O(V^3)$
C) $O(E \log V)$
D) $O(V + E)$
Answer: B) $O(V^3)$

10. Which recurrence describes naive recursive Fibonacci?
A) $T(n) = T(n-1) + T(n-2) + O(1)$
B) $T(n) = 2T(n/2) + O(1)$
C) $T(n) = T(n/2) + O(1)$
D) $T(n) = T(n-1) + O(1)$
Answer: A) $T(n) = T(n-1) + T(n-2) + O(1)$

Coding Problems

Easy Coding (10)

Q1. Measure execution time of a loop.

```
public class TimeLoop {
    public static void main(String[] args) {
        long start = System.nanoTime();
        for (int i = 0; i < 1000000; i++) {
            // Empty loop
        }
        long end = System.nanoTime();
        System.out.println("Time: " + (end - start) / 1_000_000.0 + " ms");
    }
}
```

Q2. Count operations in linear search.

```
public class LinearSearchCount {
    public static int search(int[] arr, int target) {
        int count = 0;
```

```
        for (int i = 0; i < arr.length; i++) {
            count++;
            if (arr[i] == target) {
                System.out.println("Operations: " + count);
                return i;
            }
        }
        System.out.println("Operations: " + count);
        return -1;
    }
    public static void main(String[] args) {
        search(new int[]{1,2,3,4,5}, 5);
    }
}
```

Q3. Verify O(1) array access.

```
public class ArrayAccessTime {
    public static void main(String[] args) {
        int[] arr = new int[1000000];
        for (int i = 0; i < arr.length; i++) arr[i] = i;

        long start = System.nanoTime();
        int x = arr[500000]; // O(1)
        long end = System.nanoTime();
        System.out.println("Access time: " + (end - start) + " ns");
    }
}
```

Q4. Count comparisons in bubble sort.

```
public class BubbleSortCount {
    public static void sort(int[] arr) {
        int comparisons = 0;
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                comparisons++;
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
```

```
            }
        }
    }
    System.out.println("Comparisons: " + comparisons);
  }
  public static void main(String[] args) {
    sort(new int[]{5,4,3,2,1});
  }
}
```

Q5. Demonstrate O(n) vs O(n²) with timing.

```
public class ComplexityDemo {
  public static void main(String[] args) {
    int n = 10000;
    // O(n)
    long start = System.nanoTime();
    for (int i = 0; i < n; i++) { }
    long end = System.nanoTime();
    System.out.println("O(n): " + (end - start) / 1e6 + " ms");

    // O(n²)
    start = System.nanoTime();
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) { }
    }
    end = System.nanoTime();
    System.out.println("O(n²): " + (end - start) / 1e6 + " ms");
  }
}
```

Q6. Count recursive calls in factorial.

```
public class FactorialCount {
  static int calls = 0;
  static long fact(int n) {
    calls++;
    if (n <= 1) return 1L;
    return n * fact(n - 1);
  }
  public static void main(String[] args) {
```

```java
        fact(5);
        System.out.println("Recursive calls: " + calls); // 6
    }
}
```

Q7. Measure memory usage of an array.

```java
public class MemoryUsage {
    public static void main(String[] args) {
        Runtime rt = Runtime.getRuntime();
        long before = rt.totalMemory() - rt.freeMemory();
        int[] arr = new int[1000000]; // ~4MB
        long after = rt.totalMemory() - rt.freeMemory();
        System.out.println("Memory used: " + (after - before) / (1024.0 * 1024) + " MB");
    }
}
```

Q8. Verify binary search is O(log n) with operation count.

```java
public class BinarySearchCount {
    public static int search(int[] arr, int target) {
        int low = 0, high = arr.length - 1, count = 0;
        while (low <= high) {
            count++;
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) {
                System.out.println("Operations: " + count);
                return mid;
            } else if (arr[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        System.out.println("Operations: " + count);
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = new int[1024];
        for (int i = 0; i < arr.length; i++) arr[i] = i;
        search(arr, 512);
```

```
    }
}
```

Q9. Count swaps in selection sort.

```
public class SelectionSortCount {
    public static void sort(int[] arr) {
        int swaps = 0;
        for (int i = 0; i < arr.length - 1; i++) {
            int min = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[min]) min = j;
            }
            if (min != i) {
                int temp = arr[i];
                arr[i] = arr[min];
                arr[min] = temp;
                swaps++;
            }
        }
        System.out.println("Swaps: " + swaps);
    }
    public static void main(String[] args) {
        sort(new int[]{5,4,3,2,1});
    }
}
```

Q10. Measure time for different input sizes.

```
public class InputSizeTime {
    public static void main(String[] args) {
        for (int n : new int[]{1000, 2000, 4000}) {
            long start = System.nanoTime();
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) { }
            }
            long end = System.nanoTime();
            System.out.println("n=" + n + ", time=" + (end - start) / 1e6 + " ms");
        }
    }
}
```

Medium Coding (10)

Q11. Solve recurrence T(n) = 2T(n/2) + n using iteration.

```
public class RecurrenceSolver {
    public static void main(String[] args) {
        // T(n) = 2T(n/2) + n
        // T(n) = n + 2*(n/2) + 4*(n/4) + ... + n * T(1)
        // = n log n + n
        // So O(n log n)
        System.out.println("T(n) = O(n log n)");
    }
}
```

Q12. Implement and time merge sort vs bubble sort.

```
import java.util.*;
public class SortComparison {
    // Merge sort and bubble sort implementations
    static void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }
    static void merge(int[] arr, int l, int m, int r) {
        // ... (standard merge)
    }
    static void bubbleSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
```

```
    }
    public static void main(String[] args) {
        int n = 10000;
        int[] arr1 = new int[n], arr2 = new int[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            arr1[i] = rand.nextInt();
            arr2[i] = arr1[i];
        }
        long start = System.nanoTime();
        bubbleSort(arr1);
        long end = System.nanoTime();
        System.out.println("Bubble: " + (end - start) / 1e6 + " ms");

        start = System.nanoTime();
        mergeSort(arr2, 0, n - 1);
        end = System.nanoTime();
        System.out.println("Merge: " + (end - start) / 1e6 + " ms");
    }
}
```

Q13. Analyze space complexity of recursive vs iterative Fibonacci.

```
public class FibonacciSpace {
    static long recursive(int n) {
        if (n <= 1) return n;
        return recursive(n - 1) + recursive(n - 2);
    }
    static long iterative(int n) {
        if (n <= 1) return n;
        long a = 0, b = 1;
        for (int i = 2; i <= n; i++) {
            long c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
    public static void main(String[] args) {
        // Recursive uses O(n) stack space, iterative uses O(1)
```

```
        System.out.println("Recursive: O(n) space, Iterative: O(1) space");
    }
}
```

Q14. Count total operations in quick sort.

```
public class QuickSortCount {
    static int comparisons = 0;
    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            comparisons++;
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }
    static void sort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }
    public static void main(String[] args) {
        int[] arr = {3,1,4,1,5};
        sort(arr, 0, 4);
        System.out.println("Comparisons: " + comparisons);
    }
}
```

Q15. Verify O(log n) for binary search with large n.

```
public class BinarySearchLogN {
    public static void main(String[] args) {
        int n = 1000000;
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = i;
        int target = n / 2;
        int low = 0, high = n - 1, count = 0;
        while (low <= high) {
            count++;
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) break;
            else if (arr[mid] < target) low = mid + 1;
            else high = mid - 1;
        }
        System.out.println("n=" + n + ", operations=" + count + ", log2(n)=" +
(int)(Math.log(n) / Math.log(2)));
    }
}
```

Q16. Measure amortized cost of dynamic array.

```
public class DynamicArrayAmortized {
    public static void main(String[] args) {
        int capacity = 1, size = 0, resizeCount = 0;
        long totalCost = 0;
        for (int i = 1; i <= 1000; i++) {
            totalCost++; // insert cost
            if (size == capacity) {
                capacity *= 2;
                totalCost += size; // copy cost
                resizeCount++;
            }
            size++;
        }
        System.out.println("Total cost: " + totalCost + ", Amortized: " + (double)totalCost /
1000);
    }
}
```

Q17. Compare HashMap vs ArrayList for search.

```
import java.util.*;
public class SearchComparison {
    public static void main(String[] args) {
        int n = 100000;
        List<Integer> list = new ArrayList<>();
        Set<Integer> set = new HashSet<>();
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            int val = rand.nextInt();
            list.add(val);
            set.add(val);
        }
        int target = list.get(n / 2);

        // ArrayList search O(n)
        long start = System.nanoTime();
        list.contains(target);
        long end = System.nanoTime();
        System.out.println("ArrayList: " + (end - start) / 1e6 + " ms");

        // HashSet search O(1)
        start = System.nanoTime();
        set.contains(target);
        end = System.nanoTime();
        System.out.println("HashSet: " + (end - start) / 1e6 + " ms");
    }
}
```

Q18. Analyze time complexity of recursive vs iterative factorial.

```
public class FactorialComplexity {
    static long recursive(int n) {
        if (n <= 1) return 1L;
        return n * recursive(n - 1);
    }
    static long iterative(int n) {
        long result = 1;
        for (int i = 2; i <= n; i++) result *= i;
        return result;
    }
```

```
    public static void main(String[] args) {
        int n = 20;
        long start = System.nanoTime();
        recursive(n);
        long end = System.nanoTime();
        System.out.println("Recursive time: " + (end - start) + " ns");

        start = System.nanoTime();
        iterative(n);
        end = System.nanoTime();
        System.out.println("Iterative time: " + (end - start) + " ns");
        // Both O(n) time, but iterative has less overhead
    }
}
```

Q19. Count operations in matrix multiplication.

```
public class MatrixMultCount {
    public static void main(String[] args) {
        int n = 100;
        int[][] a = new int[n][n];
        int[][] b = new int[n][n];
        int[][] c = new int[n][n];
        long operations = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    c[i][j] += a[i][k] * b[k][j];
                    operations++;
                }
            }
        }
        System.out.println("Operations: " + operations + " (should be " + (n * n * n) + ")");
    }
}
```

Q20. Verify O(n log n) for merge sort with operation count.

```
public class MergeSortCount {
    static long comparisons = 0;
    static void merge(int[] arr, int l, int m, int r) {
```

```
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, l, L, 0, n1);
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            comparisons++;
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }
    static void sort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }
    public static void main(String[] args) {
        int n = 1024;
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = n - i;
        sort(arr, 0, n - 1);
        System.out.println("Comparisons: " + comparisons + ", n log n = " + (n *
(int)(Math.log(n) / Math.log(2))));
    }
}
```

Hard Coding (10)

Q21. Implement and analyze Strassen's algorithm vs naive multiplication.

```
public class StrassenVsNaive {
    // Naive O(n³)
    static int[][] naive(int[][] a, int[][] b) {
        int n = a.length;
        int[][] c = new int[n][n];
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
        }
        return c;
    }
    // Strassen's is complex; for n=2 only
    static int[][] strassen2x2(int[][] a, int[][] b) {
        int p1 = (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
        int p2 = (a[1][0] + a[1][1]) * b[0][0];
        int p3 = a[0][0] * (b[0][1] - b[1][1]);
        int p4 = a[1][1] * (b[1][0] - b[0][0]);
        int p5 = (a[0][0] + a[0][1]) * b[1][1];
        int p6 = (a[1][0] - a[0][0]) * (b[0][0] + b[0][1]);
        int p7 = (a[0][1] - a[1][1]) * (b[1][0] + b[1][1]);
        int[][] c = new int[2][2];
        c[0][0] = p1 + p4 - p5 + p7;
        c[0][1] = p3 + p5;
        c[1][0] = p2 + p4;
        c[1][1] = p1 - p2 + p3 + p6;
        return c;
    }
    public static void main(String[] args) {
        int[][] a = {{1,2},{3,4}}, b = {{5,6},{7,8}};
        long start = System.nanoTime();
        naive(a, b);
        long end = System.nanoTime();
        System.out.println("Naive time: " + (end - start) + " ns");

        start = System.nanoTime();
        strassen2x2(a, b);
        end = System.nanoTime();
        System.out.println("Strassen time: " + (end - start) + " ns");
        // For small n, naive is faster due to overhead
    }
}
```

Q22. Solve T(n) = T(n/2) + O(1) and verify with binary search.
```
public class LogarithmicRecurrence {
    static int count = 0;
    static int binarySearch(int[] arr, int low, int high, int target) {
        count++;
        if (low > high) return -1;
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) return mid;
        else if (arr[mid] < target) return binarySearch(arr, mid + 1, high, target);
        else return binarySearch(arr, low, mid - 1, target);
    }
    public static void main(String[] args) {
        int n = 1024;
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = i;
        count = 0;
        binarySearch(arr, 0, n - 1, 512);
        System.out.println("Operations: " + count + ", log2(n) = " + (int)(Math.log(n) /
Math.log(2)));
        // T(n) = O(log n)
    }
}
```

Q23. Analyze the complexity of Dijkstra's algorithm with different data structures.
```
public class DijkstraComplexity {
    public static void main(String[] args) {
        // With array: O(V²)
        // With binary heap: O((V + E) log V)
        // With Fibonacci heap: O(E + V log V)
        System.out.println("Array: O(V²), Binary Heap: O((V+E) log V), Fib Heap: O(E + V
log V)");
    }
}
```

Q24. Measure the impact of cache on matrix traversal (row vs column).
```
public class CacheImpact {
    public static void main(String[] args) {
        int n = 2000;
```

```
        int[][] mat = new int[n][n];

        // Row-major (cache-friendly)
        long start = System.nanoTime();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                mat[i][j] = i + j;
            }
        }
        long end = System.nanoTime();
        System.out.println("Row-major: " + (end - start) / 1e6 + " ms");

        // Column-major (cache-unfriendly)
        start = System.nanoTime();
        for (int j = 0; j < n; j++) {
            for (int i = 0; i < n; i++) {
                mat[i][j] = i + j;
            }
        }
        end = System.nanoTime();
        System.out.println("Column-major: " + (end - start) / 1e6 + " ms");
    }
}
```

Q25. Verify the lower bound of comparison-based sorting.

```
public class SortingLowerBound {
    public static void main(String[] args) {
        // For n elements, min comparisons = log2(n!) ≈ n log n - 1.44n
        int n = 10;
        double minComparisons = Math.log(factorial(n)) / Math.log(2);
        System.out.println("n=" + n + ", min comparisons ≈ " + (int)minComparisons);
        // Any comparison sort must use at least this many comparisons in worst case
    }
    static long factorial(int n) {
        long f = 1;
        for (int i = 2; i <= n; i++) f *= i;
        return f;
    }
}
```

Q26. Analyze the complexity of recursive Fibonacci with memoization.

```
import java.util.*;
public class MemoizedFibonacci {
    static Map<Integer, Long> memo = new HashMap<>();
    static long fib(int n) {
        if (n <= 1) return n;
        if (memo.containsKey(n)) return memo.get(n);
        long result = fib(n - 1) + fib(n - 2);
        memo.put(n, result);
        return result;
    }
    public static void main(String[] args) {
        // Without memo: O(2^n)
        // With memo: O(n) time, O(n) space
        System.out.println("Memoized Fibonacci: O(n) time, O(n) space");
    }
}
```

Q27. Compare iterative and recursive DFS space complexity.

```
import java.util.*;
public class DFSComplexity {
    // Recursive DFS: O(V) space (call stack)
    static void dfsRecursive(List<List<Integer>> graph, int node, boolean[] visited) {
        visited[node] = true;
        for (int neighbor : graph.get(node)) {
            if (!visited[neighbor]) {
                dfsRecursive(graph, neighbor, visited);
            }
        }
    }
    // Iterative DFS: O(V) space (explicit stack)
    static void dfsIterative(List<List<Integer>> graph, int start) {
        Stack<Integer> stack = new Stack<>();
        boolean[] visited = new boolean[graph.size()];
        stack.push(start);
        while (!stack.isEmpty()) {
            int node = stack.pop();
            if (!visited[node]) {
```

```
            visited[node] = true;
            for (int neighbor : graph.get(node)) {
                stack.push(neighbor);
            }
        }
    }
}
public static void main(String[] args) {
    // Both use O(V) space, but iterative avoids stack overflow
    System.out.println("Both DFS versions: O(V) space complexity");
}
}
```

Q28. Analyze the complexity of the Sieve of Eratosthenes.

```
public class SieveComplexity {
    public static void main(String[] args) {
        int n = 1000000;
        boolean[] prime = new boolean[n + 1];
        Arrays.fill(prime, true);
        prime[0] = prime[1] = false;
        long operations = 0;
        for (int p = 2; p * p <= n; p++) {
            if (prime[p]) {
                for (int i = p * p; i <= n; i += p) {
                    prime[i] = false;
                    operations++;
                }
            }
        }
        // Time complexity: O(n log log n)
        System.out.println("Operations: " + operations + ", n log log n ≈ " + (n *
Math.log(Math.log(n))));
    }
}
```

Q29. Verify the Master Theorem with a custom recurrence.

```
public class MasterTheoremVerify {
    public static void main(String[] args) {
        // T(n) = 3T(n/4) + n
```

```java
        // a=3, b=4, f(n)=n
        // n^(log_b a) = n^(log4 3) ≈ n^0.79
        // f(n) = n = Ω(n^0.79 + ε) for ε=0.2
        // Case 3: T(n) = Θ(f(n)) = Θ(n)
        System.out.println("T(n) = 3T(n/4) + n => Θ(n)");
    }
}
```

Q30. Analyze the complexity of union-find with path compression.

```java
public class UnionFindComplexity {
    static int[] parent, rank;
    static int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]); // path compression
        }
        return parent[x];
    }
    static void union(int x, int y) {
        x = find(x); y = find(y);
        if (x == y) return;
        if (rank[x] < rank[y]) parent[x] = y;
        else if (rank[x] > rank[y]) parent[y] = x;
        else {
            parent[y] = x;
            rank[x]++;
        }
    }
    public static void main(String[] args) {
        // With union by rank and path compression: O(α(n)) per operation
        // α(n) is inverse Ackermann, <= 5 for any practical n
        System.out.println("Union-Find with optimizations: O(α(n)) per operation");
    }
}
```

TOPIC 14: Selection Sort, Insertion Sort

Easy MCQs (10)

1. Selection Sort works by:

A) Swapping adjacent elements
B) Selecting the smallest element and placing it at the beginning
C) Dividing the array
D) Using recursion
Answer: B) Selecting the smallest element and placing it at the beginning

2. Insertion Sort is efficient for:
A) Large unsorted arrays
B) Small or nearly sorted arrays
C) Reverse-sorted arrays
D) Arrays with duplicates only
Answer: B) Small or nearly sorted arrays

3. What is the time complexity of Selection Sort in all cases?
A) $O(n)$
B) $O(n \log n)$
C) $O(n^2)$
D) $O(\log n)$
Answer: C) $O(n^2)$

4. Insertion Sort is:
A) Not stable
B) Stable
C) Only stable for integers
D) Unstable by design
Answer: B) Stable

5. How many swaps does Selection Sort make in total?
A) $O(n^2)$
B) $O(n)$
C) $O(\log n)$
D) $O(1)$
Answer: B) $O(n)$

6. In Insertion Sort, elements are inserted into:
A) The end of the array
B) The correct position in the sorted portion
C) A new array
D) Random positions

Answer: B) The correct position in the sorted portion

7. Selection Sort is:
   A) Stable
   B) Not stable
   C) Conditionally stable
   D) Always stable
   Answer: B) Not stable

8. Which sort is adaptive?
   A) Selection Sort
   B) Insertion Sort
   C) Both
   D) Neither
   Answer: B) Insertion Sort

9. The best-case time complexity of Insertion Sort is:
   A) O(n)
   B) O(n log n)
   C) $O(n^2)$
   D) O(1)
   Answer: A) O(n)

10. Which algorithm uses less writes to memory?
   A) Bubble Sort
   B) Selection Sort
   C) Insertion Sort
   D) Quick Sort
   Answer: B) Selection Sort

Medium MCQs (10)

1. Why is Selection Sort not stable?
   A) It uses recursion
   B) It swaps non-adjacent elements
   C) It doesn't compare elements
   D) It uses extra space
   Answer: B) It swaps non-adjacent elements

2. In Insertion Sort, the inner loop shifts elements to:
   A) The left
   B) The right
   C) Random positions
   D) The end
   Answer: B) The right

3. What is the space complexity of both sorts?
   A) O(n)
   B) O(log n)
   C) O(1)
   D) O(n^2)
   Answer: C) O(1)

4. For an array of size n, how many comparisons does Selection Sort make?
   A) n
   B) n-1
   C) n(n-1)/2
   D) n^2
   Answer: C) n(n-1)/2

5. Insertion Sort performs well on:
   A) Random data
   B) Sorted data
   C) Reverse-sorted data
   D) All equally
   Answer: B) Sorted data

6. Which sort is preferred for small datasets in hybrid algorithms (e.g., Timsort)?
   A) Selection Sort
   B) Insertion Sort
   C) Bubble Sort
   D) Quick Sort
   Answer: B) Insertion Sort

7. In Selection Sort, after i iterations, the first i elements are:
   A) Sorted but not in final position
   B) Sorted and in final position
   C) Unsorted

D) Reversed
Answer: B) Sorted and in final position

8. The number of inversions in an array affects:
A) Selection Sort performance
B) Insertion Sort performance
C) Both equally
D) Neither
Answer: B) Insertion Sort performance

9. Which sort is online (can sort as data arrives)?
A) Selection Sort
B) Insertion Sort
C) Both
D) Neither
Answer: B) Insertion Sort

10. What is the worst-case time complexity of Insertion Sort?
A) $O(n)$
B) $O(n \log n)$
C) $O(n^2)$
D) $O(2^n)$
Answer: C) $O(n^2)$

Hard MCQs (10)

1. In a stable Selection Sort variant, what is the time complexity?
A) $O(n)$
B) $O(n \log n)$
C) $O(n^2)$
D) $O(n^3)$
Answer: C) $O(n^2)$
(Stable version shifts instead of swapping, but still $O(n^2)$)

2. The number of swaps in Insertion Sort equals:
A) Number of comparisons
B) Number of inversions
C) n-1
D) 0

Answer: B) Number of inversions

3. Which sort minimizes the number of writes to memory?
A) Insertion Sort
B) Selection Sort
C) Bubble Sort
D) Merge Sort
Answer: B) Selection Sort
(Exactly n-1 swaps)

4. For an array with k inversions, Insertion Sort runs in:
A) $O(k)$
B) $O(n + k)$
C) $O(nk)$
D) $O(k^2)$
Answer: B) $O(n + k)$

5. In Selection Sort, if all elements are equal, how many swaps occur?
A) 0
B) n-1
C) n(n-1)/2
D) 1
Answer: B) n-1
(It still performs swaps even if unnecessary)

6. Which property makes Insertion Sort suitable for real-time systems?
A) Low memory usage
B) Online nature and low overhead
C) Fast for large data
D) Parallelizable
Answer: B) Online nature and low overhead

7. The decision tree height for comparison-based sorting is $\Omega(n \log n)$, but Insertion Sort can be faster because:
A) It's not comparison-based
B) It's adaptive to input order
C) It uses hashing
D) It's not a sorting algorithm
Answer: B) It's adaptive to input order

8. In a cache-oblivious context, which sort has better locality?
A) Selection Sort
B) Insertion Sort
C) Both same
D) Neither
Answer: B) Insertion Sort
(Works on contiguous segments)

9. What is the best-case number of comparisons for Selection Sort?
A) O(n)
B) O(n log n)
C) $O(n^2)$
D) O(1)
Answer: C) $O(n^2)$
(Always scans the entire unsorted portion)

10. Which sort is used in Java's `Arrays.sort()` for small subarrays?
A) Selection Sort
B) Insertion Sort
C) Bubble Sort
D) Quick Sort
Answer: B) Insertion Sort

Coding Problems

Easy Coding (10)

Q1. Implement Selection Sort.

```
public class SelectionSort {
    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
```

```
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q2. Implement Insertion Sort.

```
public class InsertionSort {
    public static void sort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q3. Sort an array of strings using Insertion Sort.

```
public class StringInsertionSort {
    public static void sort(String[] arr) {
        for (int i = 1; i < arr.length; i++) {
            String key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j].compareTo(key) > 0) {
```

```
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
public static void main(String[] args) {
    String[] arr = {"banana", "apple", "cherry"};
    sort(arr);
    for (String s : arr) System.out.print(s + " ");
}
}
```

Q4. Count the number of swaps in Selection Sort.

```
public class SelectionSortSwaps {
    public static int sort(int[] arr) {
        int swaps = 0;
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            if (minIndex != i) {
                int temp = arr[minIndex];
                arr[minIndex] = arr[i];
                arr[i] = temp;
                swaps++;
            }
        }
        return swaps;
    }
    public static void main(String[] args) {
        int[] arr = {5, 4, 3, 2, 1};
        System.out.println("Swaps: " + sort(arr));
    }
}
```

Q5. Count the number of comparisons in Insertion Sort.

```
public class InsertionSortComparisons {
    public static int sort(int[] arr) {
        int comparisons = 0;
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0) {
                comparisons++;
                if (arr[j] > key) {
                    arr[j + 1] = arr[j];
                    j--;
                } else {
                    break;
                }
            }
            arr[j + 1] = key;
        }
        return comparisons;
    }
    public static void main(String[] args) {
        int[] arr = {5, 4, 3, 2, 1};
        System.out.println("Comparisons: " + sort(arr));
    }
}
```

Q6. Sort in descending order using Selection Sort.

```
public class SelectionSortDesc {
    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int maxIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] > arr[maxIndex]) {
                    maxIndex = j;
                }
            }
            int temp = arr[maxIndex];
```

```
            arr[maxIndex] = arr[i];
            arr[i] = temp;
        }
    }
    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q7. Sort in descending order using Insertion Sort.

```
public class InsertionSortDesc {
    public static void sort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] < key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q8. Check if an array is sorted using Insertion Sort logic.

```
public class IsSortedInsertion {
    public static boolean isSorted(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < arr[i - 1]) {
                return false;
            }
        }
```

```java
        return true;
    }
    public static void main(String[] args) {
        System.out.println(isSorted(new int[]{1,2,3})); // true
    }
}
```

Q9. Sort only a subarray using Insertion Sort.

```java
public class SubarrayInsertionSort {
    public static void sort(int[] arr, int left, int right) {
        for (int i = left + 1; i <= right; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= left && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
    public static void main(String[] args) {
        int[] arr = {5, 4, 3, 2, 1};
        sort(arr, 1, 3); // sort indices 1 to 3
        for (int x : arr) System.out.print(x + " "); // 5 2 3 4 1
    }
}
```

Q10. Implement stable Selection Sort (using shifting).

```java
public class StableSelectionSort {
    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            // Shift elements instead of swapping
```

```
            int temp = arr[minIndex];
            for (int k = minIndex; k > i; k--) {
                arr[k] = arr[k - 1];
            }
            arr[i] = temp;
        }
    }
    public static void main(String[] args) {
        int[] arr = {4, 3, 2, 1};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Medium Coding (10)

Q11. Sort an array of objects using Insertion Sort (by a field).

```
class Student {
    String name; int marks;
    Student(String n, int m) { name = n; marks = m; }
}

public class ObjectInsertionSort {
    public static void sort(Student[] students) {
        for (int i = 1; i < students.length; i++) {
            Student key = students[i];
            int j = i - 1;
            while (j >= 0 && students[j].marks > key.marks) {
                students[j + 1] = students[j];
                j--;
            }
            students[j + 1] = key;
        }
    }
    public static void main(String[] args) {
        Student[] students = {
            new Student("Alice", 85),
            new Student("Bob", 75)
        };
```

```
        sort(students);
        for (Student s : students) {
            System.out.println(s.name + ": " + s.marks);
        }
    }
}
```

Q12. Implement Selection Sort with early termination.

```
public class EarlySelectionSort {
    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            boolean swapped = false;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                    swapped = true;
                }
            }
            if (swapped) {
                int temp = arr[minIndex];
                arr[minIndex] = arr[i];
                arr[i] = temp;
            }
        }
    }
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5}; // already sorted
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q13. Count inversions using Insertion Sort logic.

```
public class InversionCount {
    public static int count(int[] arr) {
        int inversions = 0;
        for (int i = 1; i < arr.length; i++) {
```

```
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
                inversions++;
            }
            arr[j + 1] = key;
        }
        return inversions;
    }
    public static void main(String[] args) {
        int[] arr = {2, 4, 1, 3, 5};
        System.out.println("Inversions: " + count(arr)); // 3
    }
}
```

Q14. Sort a 2D array row-wise using Insertion Sort.

```
public class RowWiseInsertionSort {
    public static void sort(int[][] matrix) {
        for (int[] row : matrix) {
            for (int i = 1; i < row.length; i++) {
                int key = row[i];
                int j = i - 1;
                while (j >= 0 && row[j] > key) {
                    row[j + 1] = row[j];
                    j--;
                }
                row[j + 1] = key;
            }
        }
    }
    public static void main(String[] args) {
        int[][] mat = {{3,1,2},{6,5,4}};
        sort(mat);
        for (int[] row : mat) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
```

```
    }
}
```

Q15. Implement binary insertion sort (use binary search for insertion point).

```
public class BinaryInsertionSort {
    public static void sort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int left = 0, right = i;
            // Binary search for insertion point
            while (left < right) {
                int mid = (left + right) / 2;
                if (arr[mid] > key) {
                    right = mid;
                } else {
                    left = mid + 1;
                }
            }
            // Shift elements
            for (int j = i; j > left; j--) {
                arr[j] = arr[j - 1];
            }
            arr[left] = key;
        }
    }
    public static void main(String[] args) {
        int[] arr = {5, 2, 4, 6, 1, 3};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q16. Sort with a custom comparator using Insertion Sort.

```
import java.util.*;
public class CustomInsertionSort {
    public static <T> void sort(T[] arr, Comparator<T> cmp) {
        for (int i = 1; i < arr.length; i++) {
            T key = arr[i];
            int j = i - 1;
```

```
            while (j >= 0 && cmp.compare(arr[j], key) > 0) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
    public static void main(String[] args) {
        String[] arr = {"banana", "apple", "cherry"};
        sort(arr, (a, b) -> b.compareTo(a)); // descending
        for (String s : arr) System.out.print(s + " ");
    }
}
```

Q17. Measure time taken by Selection Sort vs Insertion Sort.

```
import java.util.*;
public class SortTimeComparison {
    // Include SelectionSort.sort and InsertionSort.sort methods
    public static void main(String[] args) {
        int n = 10000;
        int[] arr1 = new int[n], arr2 = new int[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            arr1[i] = rand.nextInt();
            arr2[i] = arr1[i];
        }
        long start = System.nanoTime();
        SelectionSort.sort(arr1);
        long end = System.nanoTime();
        System.out.println("Selection Sort: " + (end - start) / 1e6 + " ms");

        start = System.nanoTime();
        InsertionSort.sort(arr2);
        end = System.nanoTime();
        System.out.println("Insertion Sort: " + (end - start) / 1e6 + " ms");
    }
}
```

Q18. Sort a partially sorted array efficiently with Insertion Sort.

```
public class PartialSort {
    public static void main(String[] args) {
        // Array is mostly sorted
        int[] arr = {1, 2, 3, 5, 4, 6, 7, 8, 10, 9};
        InsertionSort.sort(arr); // efficient for few inversions
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q19. Implement Selection Sort for a linked list (simulated with array).

```
public class LinkedListSelectionSort {
    // Simulate linked list with array indices
    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            if (minIndex != i) {
                int temp = arr[minIndex];
                arr[minIndex] = arr[i];
                arr[i] = temp;
            }
        }
    }
    public static void main(String[] args) {
        int[] arr = {5, 2, 8, 1};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q20. Sort an array of doubles using Insertion Sort.

```
public class DoubleInsertionSort {
    public static void sort(double[] arr) {
        for (int i = 1; i < arr.length; i++) {
```

```
            double key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
    public static void main(String[] args) {
        double[] arr = {3.2, 1.5, 4.8, 2.1};
        sort(arr);
        for (double x : arr) System.out.print(x + " ");
    }
}
```

Hard Coding (10)

Q21. Implement stable Selection Sort that preserves order of equal elements.

```
public class StableSelectionSortHard {
    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            // Find the first occurrence of the minimum
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            // Shift all elements from i to minIndex-1 right by 1
            int temp = arr[minIndex];
            for (int k = minIndex; k > i; k--) {
                arr[k] = arr[k - 1];
            }
            arr[i] = temp;
        }
    }
    public static void main(String[] args) {
```

```
        int[] arr = {4, 2, 2, 1}; // stable: first 2 should come before second 2
        sort(arr);
        for (int x : arr) System.out.print(x + " "); // 1 2 2 4
    }
}
```

Q22. Hybrid sort: use Insertion Sort for small subarrays in Selection Sort.

```
public class HybridSelectionInsertion {
    private static final int THRESHOLD = 10;

    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i += THRESHOLD) {
            int end = Math.min(i + THRESHOLD, n);
            // Sort subarray with Insertion Sort
            for (int j = i + 1; j < end; j++) {
                int key = arr[j];
                int k = j - 1;
                while (k >= i && arr[k] > key) {
                    arr[k + 1] = arr[k];
                    k--;
                }
                arr[k + 1] = key;
            }
        }
        // Now do Selection Sort on the "sorted" blocks
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
    public static void main(String[] args) {
```

```
        int[] arr = {64, 25, 12, 22, 11, 90, 88, 76, 50, 42};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q23. Sort with duplicate keys and count frequency using Insertion Sort.

```
import java.util.*;
public class SortWithFrequency {
    static class Element {
        int value, originalIndex;
        Element(int v, int i) { value = v; originalIndex = i; }
    }

    public static void main(String[] args) {
        int[] arr = {4, 2, 2, 8, 3, 3, 1};
        Element[] elements = new Element[arr.length];
        for (int i = 0; i < arr.length; i++) {
            elements[i] = new Element(arr[i], i);
        }
        // Stable Insertion Sort by value
        for (int i = 1; i < elements.length; i++) {
            Element key = elements[i];
            int j = i - 1;
            while (j >= 0 && elements[j].value > key.value) {
                elements[j + 1] = elements[j];
                j--;
            }
            elements[j + 1] = key;
        }
        // Count frequencies
        Map<Integer, Integer> freq = new LinkedHashMap<>();
        for (Element e : elements) {
            freq.put(e.value, freq.getOrDefault(e.value, 0) + 1);
        }
        System.out.println("Sorted: " + Arrays.toString(arr));
        System.out.println("Frequencies: " + freq);
    }
}
```

Q24. Implement Selection Sort with minimum number of writes (for flash memory).

```java
public class MinWriteSelectionSort {
    public static void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            if (minIndex != i) {
                // Only 2 writes per swap
                int temp = arr[minIndex];
                arr[minIndex] = arr[i];
                arr[i] = temp;
            }
        }
    }
    public static void main(String[] args) {
        int[] arr = {5, 1, 4, 2, 8};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q25. Sort a stream of integers using Insertion Sort (online algorithm).

```java
import java.util.*;
public class StreamInsertionSort {
    private List<Integer> sorted = new ArrayList<>();

    public void insert(int value) {
        int i = Collections.binarySearch(sorted, value);
        if (i < 0) i = -i - 1;
        sorted.add(i, value);
    }

    public List<Integer> getSorted() {
```

```
        return sorted;
    }

    public static void main(String[] args) {
        StreamInsertionSort stream = new StreamInsertionSort();
        int[] input = {5, 2, 8, 1};
        for (int x : input) {
            stream.insert(x);
            System.out.println("After " + x + ": " + stream.getSorted());
        }
    }
}
```

Q26. Analyze the number of cache misses in Selection Sort vs Insertion Sort.

```
public class CacheMissAnalysis {
    public static void main(String[] args) {
        // Theoretical analysis:
        // Selection Sort: poor locality (jumps to minIndex)
        // Insertion Sort: good locality (works on contiguous segment)
        System.out.println("Selection Sort: high cache misses");
        System.out.println("Insertion Sort: low cache misses");
    }
}
```

Q27. Sort an array where each element is at most k positions from its sorted position.

```
public class NearlySortedInsertion {
    public static void sort(int[] arr, int k) {
        // Insertion Sort is optimal for this
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
    public static void main(String[] args) {
```

```
        int[] arr = {2, 1, 3, 5, 4, 6}; // k=1
        sort(arr, 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q28. Implement Selection Sort that works on a file (external sort simulation).

```
import java.io.*;
import java.util.*;

public class ExternalSelectionSort {
    public static void main(String[] args) throws Exception {
        // Simulate with list
        List<Integer> data = Arrays.asList(64, 25, 12, 22, 11);
        List<Integer> sorted = new ArrayList<>();
        List<Integer> working = new ArrayList<>(data);

        while (!working.isEmpty()) {
            int min = Collections.min(working);
            sorted.add(min);
            working.remove(Integer.valueOf(min));
        }
        System.out.println("Sorted: " + sorted);
    }
}
```

Q29. Sort with stability and custom object using Selection Sort.

```
class Product {
    String name; double price;
    Product(String n, double p) { name = n; price = p; }
    public String toString() { return name + ":" + price; }
}

public class StableObjectSelectionSort {
    public static void sort(Product[] products) {
        int n = products.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
```

```
                if (products[j].price < products[minIndex].price) {
                    minIndex = j;
                }
            }
            // Shift to maintain stability
            Product temp = products[minIndex];
            for (int k = minIndex; k > i; k--) {
                products[k] = products[k - 1];
            }
            products[i] = temp;
        }
    }
    public static void main(String[] args) {
        Product[] products = {
            new Product("A", 10.0),
            new Product("B", 10.0) // should come after A
        };
        sort(products);
        for (Product p : products) System.out.println(p);
    }
}
```

Q30. Compare the number of inversions before and after sorting.

```
public class InversionComparison {
    public static int countInversions(int[] arr) {
        int count = 0;
        for (int i = 0; i < arr.length; i++) {
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[i] > arr[j]) count++;
            }
        }
        return count;
    }
    public static void main(String[] args) {
        int[] arr = {3, 1, 2};
        int before = countInversions(arr);
        InsertionSort.sort(arr);
        int after = countInversions(arr);
        System.out.println("Before: " + before + ", After: " + after);
```

```
    }
}
```

TOPIC 15: Quick Sort, Merge Sort

Easy MCQs (10)

1. Merge Sort is based on:
   A) Divide and conquer
   B) Greedy approach
   C) Dynamic programming
   D) Backtracking
   Answer: A) Divide and conquer

2. Quick Sort's pivot selection affects:
   A) Space complexity
   B) Time complexity
   C) Stability
   D) All of the above
   Answer: B) Time complexity

3. What is the worst-case time complexity of Quick Sort?
   A) O(n)
   B) O(n log n)
   C) $O(n^2)$
   D) O(log n)
   Answer: C) $O(n^2)$

4. Merge Sort is:
   A) In-place
   B) Not in-place
   C) Unstable
   D) Online
   Answer: B) Not in-place

5. The best-case time complexity of Merge Sort is:
   A) O(n)
   B) O(n log n)
   C) $O(n^2)$

D) O(1)
Answer: B) O(n log n)

6. Quick Sort is:
A) Stable
B) Not stable
C) Always stable
D) Conditionally stable
Answer: B) Not stable

7. Merge Sort uses which data structure for merging?
A) Stack
B) Queue
C) Temporary array
D) Linked list
Answer: C) Temporary array

8. What is the average-case time complexity of Quick Sort?
A) O(n)
B) O(n log n)
C) $O(n^2)$
D) O(log n)
Answer: B) O(n log n)

9. Which sort is used in Java's `Arrays.sort()` for primitives?
A) Merge Sort
B) Quick Sort
C) Dual-Pivot Quick Sort
D) Heap Sort
Answer: C) Dual-Pivot Quick Sort

10. Merge Sort's space complexity is:
A) O(1)
B) O(log n)
C) O(n)
D) $O(n^2)$
Answer: C) O(n)

Medium MCQs (10)

1. Why is Merge Sort preferred for linked lists?
A) No random access needed
B) In-place sorting
C) Faster than Quick Sort
D) Uses less memory
Answer: A) No random access needed

2. In Quick Sort, the partition step ensures:
A) Pivot is in final position
B) Array is sorted
C) Elements are reversed
D) Duplicates are removed
Answer: A) Pivot is in final position

3. What is the recurrence for Merge Sort?
A) $T(n) = T(n-1) + O(n)$
B) $T(n) = 2T(n/2) + O(n)$
C) $T(n) = T(n/2) + O(1)$
D) $T(n) = 2T(n/2) + O(1)$
Answer: B) $T(n) = 2T(n/2) + O(n)$

4. Which pivot selection minimizes worst-case in Quick Sort?
A) First element
B) Last element
C) Median-of-three
D) Random element
Answer: C) Median-of-three

5. Merge Sort is stable because:
A) It uses extra space
B) Equal elements retain relative order during merge
C) It's recursive
D) It's not in-place
Answer: B) Equal elements retain relative order during merge

6. The depth of recursion in Quick Sort (average case) is:
A) $O(1)$
B) $O(\log n)$

C) O(n)
D) O(n log n)
Answer: B) O(log n)

7. Which algorithm is not comparison-based?
A) Quick Sort
B) Merge Sort
C) Counting Sort
D) Heap Sort
Answer: C) Counting Sort

8. In-place Merge Sort is:
A) Easy to implement
B) Complex and rarely used
C) More efficient than standard Merge Sort
D) Always stable
Answer: B) Complex and rarely used

9. What is the time complexity of merging two sorted arrays of size n/2?
A) O(1)
B) O(log n)
C) O(n)
D) $O(n^2)$
Answer: C) O(n)

10. Quick Sort is cache-friendly because:
A) It accesses memory sequentially
B) It uses less memory
C) It's recursive
D) It's stable
Answer: A) It accesses memory sequentially

Hard MCQs (10)

1. The information-theoretic lower bound for comparison-based sorting is:
A) O(n)
B) O(n log n)
C) $O(n^2)$
D) O(log n)

Answer: B) O(n log n)

2. Which variant of Quick Sort is used in Java 7+ for primitives?
   A) 3-way Quick Sort
   B) Dual-Pivot Quick Sort
   C) Randomized Quick Sort
   D) Intro Sort
   Answer: B) Dual-Pivot Quick Sort

3. What is the worst-case space complexity of Quick Sort?
   A) O(1)
   B) O(log n)
   C) O(n)
   D) O(n log n)
   Answer: C) O(n)
   (Due to recursion stack in worst-case)

4. In Merge Sort, the number of merge operations is:
   A) n
   B) log n
   C) n log n
   D) 1
   Answer: B) log n

5. Which sorting algorithm is used in `Collections.sort()`?
   A) Quick Sort
   B) Merge Sort
   C) Timsort (adaptive merge sort)
   D) Heap Sort
   Answer: C) Timsort (adaptive merge sort)

6. The probability of worst-case in randomized Quick Sort is:
   A) 1
   B) 1/n!
   C) $1/2^n$
   D) 0
   Answer: B) 1/n!

7. What is the time complexity of Merge Sort for linked lists?

A) O(n)
B) O(n log n)
C) O(n^2)
D) O(log n)
Answer: B) O(n log n)
(But space complexity is O(log n) due to recursion)

8. In 3-way Quick Sort (Dutch National Flag), the partitioning is:
A) Into 2 parts
B) Into 3 parts (less, equal, greater)
C) Into n parts
D) Not possible
Answer: B) Into 3 parts (less, equal, greater)

9. Which algorithm has guaranteed O(n log n) time?
A) Quick Sort
B) Merge Sort
C) Heap Sort
D) Both B and C
Answer: D) Both B and C

10. The cache complexity of Merge Sort is:
A) O(n)
B) O(n / B log n)
C) O(n log n / B)
D) O(n^2)
Answer: B) O(n / B log n)
(Where B is block size)

Coding Problems

Easy Coding (10)

Q1. Implement basic Merge Sort.

```
public class MergeSort {
    public static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, l, L, 0, n1);
```

```
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void sort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q2. Implement basic Quick Sort.

```
public class QuickSort {
    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
```

```
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void sort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q3. Sort an array of strings using Merge Sort.

```
public class StringMergeSort {
    public static void merge(String[] arr, int l, int m, int r) {
        int n1 = m - l + 1, n2 = r - m;
        String[] L = new String[n1], R = new String[n2];
        System.arraycopy(arr, l, L, 0, n1);
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i].compareTo(R[j]) <= 0) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void sort(String[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
```

```
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void main(String[] args) {
        String[] arr = {"banana", "apple", "cherry"};
        sort(arr, 0, arr.length - 1);
        for (String s : arr) System.out.print(s + " ");
    }
}
```

Q4. Count the number of comparisons in Merge Sort.

```
public class MergeSortComparisons {
    static long comparisons = 0;

    public static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, l, L, 0, n1);
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            comparisons++;
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void sort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
```

```
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        sort(arr, 0, arr.length - 1);
        System.out.println("Comparisons: " + comparisons);
    }
}
```

Q5. Count the number of swaps in Quick Sort.

```
public class QuickSortSwaps {
    static long swaps = 0;

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
                swaps++;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        swaps++;
        return i + 1;
    }

    public static void sort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }
```

```
    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        sort(arr, 0, arr.length - 1);
        System.out.println("Swaps: " + swaps);
    }
}
```

Q6. Sort in descending order using Merge Sort.

```
public class MergeSortDesc {
    public static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, l, L, 0, n1);
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i] >= R[j]) arr[k++] = L[i++]; // descending
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void sort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q7. Sort in descending order using Quick Sort.

```
public class QuickSortDesc {
    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] > pivot) { // descending
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void sort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q8. Check if an array is sorted using Merge Sort logic.

```
public class IsSortedMerge {
    public static boolean isSorted(int[] arr, int l, int r) {
        if (l >= r) return true;
```

```java
        int m = l + (r - l) / 2;
        if (!isSorted(arr, l, m)) return false;
        if (!isSorted(arr, m + 1, r)) return false;
        return arr[m] <= arr[m + 1];
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.println(isSorted(arr, 0, arr.length - 1));
    }
}
```

Q9. Sort only a subarray using Merge Sort.

```java
public class SubarrayMergeSort {
    // Include merge and sort methods from Q1
    public static void main(String[] args) {
        int[] arr = {5, 4, 3, 2, 1};
        MergeSort.sort(arr, 1, 3); // sort indices 1 to 3
        for (int x : arr) System.out.print(x + " "); // 5 2 3 4 1
    }
}
```

Q10. Implement stable Quick Sort (using extra space).

```java
public class StableQuickSort {
    public static int[] sort(int[] arr) {
        if (arr.length <= 1) return arr;
        int pivot = arr[arr.length / 2];
        java.util.List<Integer> less = new java.util.ArrayList<>();
        java.util.List<Integer> equal = new java.util.ArrayList<>();
        java.util.List<Integer> greater = new java.util.ArrayList<>();
        for (int x : arr) {
            if (x < pivot) less.add(x);
            else if (x == pivot) equal.add(x);
            else greater.add(x);
        }
        int[] result = new int[arr.length];
        int[] lessArr = sort(less.stream().mapToInt(i -> i).toArray());
        int[] greaterArr = sort(greater.stream().mapToInt(i -> i).toArray());
        System.arraycopy(lessArr, 0, result, 0, lessArr.length);
```

```
        System.arraycopy(equal.stream().mapToInt(i -> i).toArray(), 0, result,
lessArr.length, equal.size());
        System.arraycopy(greaterArr, 0, result, lessArr.length + equal.size(),
greaterArr.length);
        return result;
    }

    public static void main(String[] args) {
        int[] arr = {3, 1, 4, 1, 5};
        arr = sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Medium Coding (10)

Q11. Implement randomized Quick Sort.

```
public class RandomizedQuickSort {
    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return i + 1;
    }

    public static void randomizedQuickSort(int[] arr, int low, int high) {
        if (low < high) {
```

```
            // Randomize pivot
            int random = low + (int)(Math.random() * (high - low + 1));
            swap(arr, random, high);
            int pi = partition(arr, low, high);
            randomizedQuickSort(arr, low, pi - 1);
            randomizedQuickSort(arr, pi + 1, high);
        }
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        randomizedQuickSort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q12. Implement 3-way Quick Sort (Dutch National Flag).

```
public class ThreeWayQuickSort {
    public static void sort(int[] arr, int low, int high) {
        if (low >= high) return;
        int lt = low, gt = high;
        int pivot = arr[low];
        int i = low + 1;
        while (i <= gt) {
            if (arr[i] < pivot) {
                swap(arr, lt++, i++);
            } else if (arr[i] > pivot) {
                swap(arr, i, gt--);
            } else {
                i++;
            }
        }
        sort(arr, low, lt - 1);
        sort(arr, gt + 1, high);
    }

    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
```

```
        arr[j] = temp;
    }

    public static void main(String[] args) {
        int[] arr = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q13. Implement iterative Merge Sort (using stack).

```
import java.util.*;
public class IterativeMergeSort {
    public static void merge(int[] arr, int l, int m, int r) {
        // Same as Q1
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, l, L, 0, n1);
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void sort(int[] arr) {
        int n = arr.length;
        for (int currSize = 1; currSize <= n - 1; currSize = 2 * currSize) {
            for (int leftStart = 0; leftStart < n - 1; leftStart += 2 * currSize) {
                int mid = Math.min(leftStart + currSize - 1, n - 1);
                int rightEnd = Math.min(leftStart + 2 * currSize - 1, n - 1);
                if (mid < rightEnd) {
                    merge(arr, leftStart, mid, rightEnd);
                }
            }
        }
```

```
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        sort(arr);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q14. Implement iterative Quick Sort (using stack).

```
import java.util.*;
public class IterativeQuickSort {
    public static int partition(int[] arr, int low, int high) {
        // Same as Q2
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void sort(int[] arr) {
        Stack<Integer> stack = new Stack<>();
        stack.push(0);
        stack.push(arr.length - 1);

        while (!stack.isEmpty()) {
            int high = stack.pop();
            int low = stack.pop();
            if (low < high) {
```

```
            int pi = partition(arr, low, high);
            stack.push(low);
            stack.push(pi - 1);
            stack.push(pi + 1);
            stack.push(high);
        }
    }
}

public static void main(String[] args) {
    int[] arr = {10, 7, 8, 9, 1, 5};
    sort(arr);
    for (int x : arr) System.out.print(x + " ");
}
}
```

Q15. Sort a linked list using Merge Sort.

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class LinkedListMergeSort {
    public static ListNode sortList(ListNode head) {
        if (head == null || head.next == null) return head;
        ListNode mid = getMid(head);
        ListNode left = sortList(head);
        ListNode right = sortList(mid);
        return merge(left, right);
    }

    private static ListNode getMid(ListNode head) {
        ListNode midPrev = null;
        while (head != null && head.next != null) {
            midPrev = (midPrev == null) ? head : midPrev.next;
            head = head.next.next;
        }
        ListNode mid = midPrev.next;
```

```
        midPrev.next = null;
        return mid;
    }

    private static ListNode merge(ListNode list1, ListNode list2) {
        ListNode dummyHead = new ListNode(0);
        ListNode tail = dummyHead;
        while (list1 != null && list2 != null) {
            if (list1.val < list2.val) {
                tail.next = list1;
                list1 = list1.next;
            } else {
                tail.next = list2;
                list2 = list2.next;
            }
            tail = tail.next;
        }
        tail.next = (list1 != null) ? list1 : list2;
        return dummyHead.next;
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(4);
        head.next = new ListNode(2);
        head.next.next = new ListNode(1);
        head.next.next.next = new ListNode(3);
        head = sortList(head);
        while (head != null) {
            System.out.print(head.val + " ");
            head = head.next;
        }
    }
}
```

Q16. Implement Merge Sort with custom comparator.

```
import java.util.*;

public class CustomMergeSort {
    public static <T> void merge(T[] arr, int l, int m, int r, Comparator<T> cmp) {
```

```
        int n1 = m - l + 1, n2 = r - m;
        T[] L = Arrays.copyOfRange(arr, l, m + 1);
        T[] R = Arrays.copyOfRange(arr, m + 1, r + 1);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (cmp.compare(L[i], R[j]) <= 0) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static <T> void sort(T[] arr, int l, int r, Comparator<T> cmp) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m, cmp);
            sort(arr, m + 1, r, cmp);
            merge(arr, l, m, r, cmp);
        }
    }

    public static void main(String[] args) {
        String[] arr = {"banana", "apple", "cherry"};
        sort(arr, 0, arr.length - 1, (a, b) -> b.compareTo(a)); // descending
        for (String s : arr) System.out.print(s + " ");
    }
}
```

Q17. Measure time taken by Quick Sort vs Merge Sort.

```
import java.util.*;
public class SortTimeComparison {
    // Include QuickSort.sort and MergeSort.sort
    public static void main(String[] args) {
        int n = 100000;
        int[] arr1 = new int[n], arr2 = new int[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            arr1[i] = rand.nextInt();
            arr2[i] = arr1[i];
```

```
        }
        long start = System.nanoTime();
        QuickSort.sort(arr1, 0, n - 1);
        long end = System.nanoTime();
        System.out.println("Quick Sort: " + (end - start) / 1e6 + " ms");

        start = System.nanoTime();
        MergeSort.sort(arr2, 0, n - 1);
        end = System.nanoTime();
        System.out.println("Merge Sort: " + (end - start) / 1e6 + " ms");
    }
}
```

Q18. Sort an array of objects using Quick Sort (by a field).

```
class Student {
    String name; int marks;
    Student(String n, int m) { name = n; marks = m; }
}

public class ObjectQuickSort {
    public static int partition(Student[] students, int low, int high) {
        int pivot = students[high].marks;
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (students[j].marks < pivot) {
                i++;
                Student temp = students[i];
                students[i] = students[j];
                students[j] = temp;
            }
        }
        Student temp = students[i + 1];
        students[i + 1] = students[high];
        students[high] = temp;
        return i + 1;
    }

    public static void sort(Student[] students, int low, int high) {
        if (low < high) {
```

```
            int pi = partition(students, low, high);
            sort(students, low, pi - 1);
            sort(students, pi + 1, high);
        }
    }

    public static void main(String[] args) {
        Student[] students = {
            new Student("Alice", 85),
            new Student("Bob", 75)
        };
        sort(students, 0, students.length - 1);
        for (Student s : students) {
            System.out.println(s.name + ": " + s.marks);
        }
    }
}
```

Q19. Implement Merge Sort for a 2D array (row-wise).

```
public class RowWiseMergeSort {
    // Include MergeSort.sort method
    public static void sort(int[][] matrix) {
        for (int[] row : matrix) {
            MergeSort.sort(row, 0, row.length - 1);
        }
    }

    public static void main(String[] args) {
        int[][] mat = {{3,1,2},{6,5,4}};
        sort(mat);
        for (int[] row : mat) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

Q20. Sort with duplicate keys and maintain stability using Merge Sort.

```
public class StableMergeSort {
```

```java
    // Merge Sort is inherently stable
    public static void main(String[] args) {
        int[] arr = {4, 2, 2, 1}; // first 2 should come before second 2
        MergeSort.sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " "); // 1 2 2 4
    }
}
```

Hard Coding (10)

Q21. Implement Dual-Pivot Quick Sort (Java 7+ style).

```java
public class DualPivotQuickSort {
    public static void sort(int[] arr, int left, int right) {
        if (left < right) {
            if (arr[left] > arr[right]) {
                swap(arr, left, right);
            }
            int p = arr[left], q = arr[right];
            int l = left + 1, g = right - 1, k = l;
            while (k <= g) {
                if (arr[k] < p) {
                    swap(arr, k, l);
                    l++;
                } else if (arr[k] >= q) {
                    while (arr[g] > q && k < g) g--;
                    swap(arr, k, g);
                    g--;
                    if (arr[k] < p) {
                        swap(arr, k, l);
                        l++;
                    }
                }
                k++;
            }
            l--; g++;
            swap(arr, left, l);
            swap(arr, right, g);
            sort(arr, left, l - 1);
            sort(arr, l + 1, g - 1);
```

```
            sort(arr, g + 1, right);
        }
    }

    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public static void main(String[] args) {
        int[] arr = {5, 2, 8, 1, 9, 3};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q22. Implement in-place Merge Sort (complex, for educational purposes).

```
public class InPlaceMergeSort {
    public static void merge(int[] arr, int start, int mid, int end) {
        // This is a simplified version; true in-place merge is very complex
        // We'll use rotation-based merge (not efficient but in-place)
        if (start >= end) return;
        int i = start, j = mid + 1;
        while (i <= mid && j <= end) {
            if (arr[i] <= arr[j]) {
                i++;
            } else {
                int value = arr[j];
                int index = j;
                // Shift elements
                while (index != i) {
                    arr[index] = arr[index - 1];
                    index--;
                }
                arr[i] = value;
                i++;
                mid++;
                j++;
```

```
            }
        }
    }

    public static void sort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q23. Analyze the number of cache misses in Quick Sort vs Merge Sort.

```
public class CacheMissAnalysis {
    public static void main(String[] args) {
        // Theoretical analysis:
        // Quick Sort: good locality (works on contiguous segments)
        // Merge Sort: poor locality in top levels (jumps between halves)
        System.out.println("Quick Sort: better cache performance for arrays");
        System.out.println("Merge Sort: better for linked lists");
    }
}
```

Q24. Implement parallel Merge Sort using threads.

```
public class ParallelMergeSort extends Thread {
    private int[] arr;
    private int low, high;

    public ParallelMergeSort(int[] arr, int low, int high) {
        this.arr = arr;
        this.low = low;
```

```
        this.high = high;
    }

    @Override
    public void run() {
        if (low < high) {
            int mid = low + (high - low) / 2;
            ParallelMergeSort left = new ParallelMergeSort(arr, low, mid);
            ParallelMergeSort right = new ParallelMergeSort(arr, mid + 1, high);
            left.start();
            right.start();
            try {
                left.join();
                right.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            MergeSort.merge(arr, low, mid, high);
        }
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        ParallelMergeSort sorter = new ParallelMergeSort(arr, 0, arr.length - 1);
        sorter.start();
        try {
            sorter.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q25. Implement Quick Sort with tail recursion optimization.

```
public class TailRecursiveQuickSort {
    public static int partition(int[] arr, int low, int high) {
        // Same as Q2
        int pivot = arr[high];
```

```
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void sort(int[] arr, int low, int high) {
        while (low < high) {
            int pi = partition(arr, low, high);
            // Recurse on smaller partition, iterate on larger
            if (pi - low < high - pi) {
                sort(arr, low, pi - 1);
                low = pi + 1;
            } else {
                sort(arr, pi + 1, high);
                high = pi - 1;
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q26. Sort a stream of integers using Merge Sort (external sort simulation).

```
import java.util.*;
public class ExternalMergeSort {
```

```
    public static List<Integer> sort(List<Integer> data) {
        if (data.size() <= 1) return data;
        int mid = data.size() / 2;
        List<Integer> left = new ArrayList<>(data.subList(0, mid));
        List<Integer> right = new ArrayList<>(data.subList(mid, data.size()));
        left = sort(left);
        right = sort(right);
        return merge(left, right);
    }

    private static List<Integer> merge(List<Integer> left, List<Integer> right) {
        List<Integer> result = new ArrayList<>();
        int i = 0, j = 0;
        while (i < left.size() && j < right.size()) {
            if (left.get(i) <= right.get(j)) {
                result.add(left.get(i++));
            } else {
                result.add(right.get(j++));
            }
        }
        while (i < left.size()) result.add(left.get(i++));
        while (j < right.size()) result.add(right.get(j++));
        return result;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(64, 25, 12, 22, 11);
        List<Integer> sorted = sort(data);
        System.out.println("Sorted: " + sorted);
    }
}
```

Q27. Implement Merge Sort that works on a file (external sort).

```
import java.io.*;
import java.util.*;
public class FileMergeSort {
    public static void main(String[] args) throws Exception {
        // Simulate with list
        List<Integer> data = Arrays.asList(64, 25, 12, 22, 11);
```

```
        List<Integer> sorted = ExternalMergeSort.sort(data);
        System.out.println("Sorted: " + sorted);
    }
}
```

Q28. Compare the number of inversions before and after sorting with Merge Sort.

```
public class InversionCountMergeSort {
    static long inversions = 0;

    public static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, l, L, 0, n1);
        System.arraycopy(arr, m + 1, R, 0, n2);
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k++] = L[i++];
            } else {
                arr[k++] = R[j++];
                inversions += (n1 - i); // Count inversions
            }
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void sort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            sort(arr, l, m);
            sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void main(String[] args) {
        int[] arr = {3, 1, 2};
        inversions = 0;
```

```
        sort(arr, 0, arr.length - 1);
        System.out.println("Inversions: " + inversions); // 2
    }
}
```

Q29. Implement Quick Sort that handles duplicate elements efficiently (3-way).

```
// Already covered in Q12 (ThreeWayQuickSort)
public class EfficientQuickSortDuplicates {
    public static void main(String[] args) {
        int[] arr = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
        ThreeWayQuickSort.sort(arr, 0, arr.length - 1);
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q30. Analyze the recursion depth of Quick Sort and Merge Sort.

```
public class RecursionDepthAnalysis {
    static int quickDepth = 0, mergeDepth = 0;
    static int maxQuickDepth = 0, maxMergeDepth = 0;

    public static void quickSort(int[] arr, int low, int high) {
        quickDepth++;
        maxQuickDepth = Math.max(maxQuickDepth, quickDepth);
        if (low < high) {
            int pi = QuickSort.partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
        quickDepth--;
    }

    public static void mergeSort(int[] arr, int l, int r) {
        mergeDepth++;
        maxMergeDepth = Math.max(maxMergeDepth, mergeDepth);
        if (l < r) {
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            MergeSort.merge(arr, l, m, r);
```

```
        }
        mergeDepth--;
    }

    public static void main(String[] args) {
        int[] arr1 = {10, 7, 8, 9, 1, 5};
        int[] arr2 = {12, 11, 13, 5, 6, 7};
        quickSort(arr1, 0, arr1.length - 1);
        mergeSort(arr2, 0, arr2.length - 1);
        System.out.println("Max Quick Sort depth: " + maxQuickDepth);
        System.out.println("Max Merge Sort depth: " + maxMergeDepth);
    }
}
```

TOPIC 16: Character Arrays vs Strings, String Creation, Immutability

Easy MCQs (10)

1. In Java, `String` objects are:
   A) Mutable
   B) Immutable
   C) Volatile
   D) Transient
   Answer: B) Immutable

2. Which is stored in the heap?
   A) String literals
   B) `new String("hello")`
   C) Both A and B
   D) Neither
   Answer: C) Both A and B
   (Literals in String Pool (part of heap since Java 7))

3. What is the output?

```
String s = "Hello";
s.concat(" World");
System.out.println(s);
```

A) `Hello World`
B) `Hello`
C) Compilation error
D) `null`
Answer: B) `Hello`

4. Character arrays are:
A) Immutable
B) Mutable
C) Final
D) Static
Answer: B) Mutable

5. Which method creates a new `String` object in heap (not pool)?
A) `String s = "hello";`
B) `String s = new String("hello");`
C) `String s = String.valueOf("hello");`
D) All of the above
Answer: B) `String s = new String("hello");`

6. The String Pool is part of:
A) Stack
B) Method Area (Java 6) / Heap (Java 7+)
C) Native Memory
D) Code Segment
Answer: B) Method Area (Java 6) / Heap (Java 7+)

7. What does `==` compare for `String` objects?
A) Content
B) Length
C) Reference
D) HashCode
Answer: C) Reference

8. Which is more secure for storing passwords?
A) `String`
B) `char[]`
C) Both same
D) `StringBuffer`

Answer: B) `char[]`

9. How many `String` objects are created?

String s1 = "Java";
String s2 = "Java";

A) 0
B) 1
C) 2
D) 3
Answer: B) 1

10. The `final` keyword in `String` class ensures:
A) The class cannot be extended
B) The object is immutable
C) Both A and B
D) Neither
Answer: C) Both A and B

Medium MCQs (10)

1. Why are `String` objects immutable?
A) For security and thread safety
B) To enable String Pool
C) To allow caching of hash codes
D) All of the above
Answer: D) All of the above

2. What is printed?

String s1 = new String("Hello");
String s2 = "Hello";
System.out.println(s1 == s2);

A) `true`
B) `false`
C) Compilation error
D) Runtime exception

Answer: B) `false`

3. After `s.intern()`, where is the string stored?
   A) Stack
   B) String Pool
   C) Native memory
   D) Register
   Answer: B) String Pool

4. Which statement about `char[]` is TRUE?
   A) It is immutable
   B) It can be cleared manually
   C) It is stored in String Pool
   D) It cannot be modified
   Answer: B) It can be cleared manually

5. What is the output?

```
char[] c = {'H', 'i'};
String s = new String(c);
c[0] = 'B';
System.out.println(s);
```

   A) `Hi`
   B) `Bi`
   C) `B`
   D) Compilation error
   Answer: A) `Hi`
   (String constructor copies the array)

6. Which is NOT a way to create a `String`?
   A) `String s = "abc";`
   B) `String s = new String("abc");`
   C) `String s = String.join("", "a", "b", "c");`
   D) `String s = (String) {'a','b','c'};`
   Answer: D) `String s = (String) {'a','b','c'};`

7. The `hashCode()` of a `String` is cached because:
   A) Strings are immutable

B) It’s required by JVM
C) For security
D) To save memory
Answer: A) Strings are immutable

8. What happens when you call `s.toUpperCase()` on a `String`?
A) Original string is modified
B) New `String` object is created
C) Exception is thrown
D) Returns `null`
Answer: B) New `String` object is created

9. Which is thread-safe?
A) `String`
B) `StringBuilder`
C) `StringBuffer`
D) Both A and C
Answer: D) Both A and C

10. Why should passwords not be stored in `String`?
A) Cannot be garbage collected
B) Remain in memory until GC, visible in dumps
C) Immutable, so cannot be cleared
D) Both B and C
Answer: D) Both B and C

Hard MCQs (10)

1. What is the output?

```
String s1 = "a" + "b";
String s2 = "ab";
System.out.println(s1 == s2);
```

A) `true`
B) `false`
C) Depends on JVM
D) Compilation error
Answer: A) `true`

(Compile-time constant concatenation)

2. What is the output?

```
String a = "hello";
String b = new String("hello");
String c = b.intern();
System.out.println(a == c);
```

A) `true`
B) `false`
C) `null`
D) Exception
Answer: A) `true`

3. In Java 8, where is the String Pool located?
A) PermGen
B) Metaspace
C) Heap
D) Native memory
Answer: C) Heap

4. Which statement is FALSE?
A) `String` uses UTF-16 internally
B) `char[]` can be modified after `String` creation
C) `String` objects can be garbage collected
D) `String` literals are created at runtime
Answer: D) `String` literals are created at runtime
(They are created at class loading time)

5. What is the memory layout of a `String` object (Java 9+)?
A) Always `char[]`
B) `byte[]` with coder field (LATIN1 or UTF16)
C) `int[]`
D) `String` is not an object
Answer: B) `byte[]` with coder field (LATIN1 or UTF16)

6. After `System.gc()`, can a `String` in the pool be collected?
A) Yes, if no references

B) No, pool strings are permanent
C) Only if interned
D) Never
Answer: A) Yes, if no references
(Since Java 7, pool is in heap and subject to GC)

7. What is the output?

```
final char[] c = {'H', 'i'};
String s = new String(c);
c[0] = 'B';
System.out.println(s);
```

A) `Hi`
B) `Bi`
C) `B`
D) Compilation error
Answer: A) `Hi`
(String constructor copies the array; `final` prevents reassignment, not mutation)

8. Which is the most efficient for building strings in a loop?
A) `String` concatenation
B) `StringBuffer`
C) `StringBuilder`
D) `char[]`
Answer: C) `StringBuilder`

9. What does `String.valueOf(null)` return?
A) `"null"`
B) `null`
C) Throws `NullPointerException`
D) Empty string
Answer: C) Throws `NullPointerException`

10. The `compactStrings` JVM option (Java 9+) affects:
A) `String` memory usage for LATIN1 strings
B) `StringBuffer` performance
C) `char[]` allocation
D) String Pool size

Answer: A) `String` memory usage for LATIN1 strings

Coding Problems

Easy Coding (10)

Q1. Demonstrate String immutability.
```
public class StringImmutability {
    public static void main(String[] args) {
        String s = "Hello";
        s.concat(" World");
        System.out.println(s); // Hello
    }
}
```

Q2. Create a String from a character array.
```
public class StringFromCharArray {
    public static void main(String[] args) {
        char[] c = {'J', 'a', 'v', 'a'};
        String s = new String(c);
        System.out.println(s); // Java
    }
}
```

Q3. Compare two Strings using `equals()` and `==`.
```
public class StringComparison {
    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = new String("Hello");
        System.out.println(s1 == s2); // false
        System.out.println(s1.equals(s2)); // true
    }
}
```

Q4. Use `intern()` to reuse String from pool.
```
public class StringIntern {
    public static void main(String[] args) {
        String s1 = new String("Java");
        String s2 = s1.intern();
```

```
        String s3 = "Java";
        System.out.println(s2 == s3); // true
    }
}
```

Q5. Clear a password stored in `char[]`.

```
public class ClearPassword {
    public static void main(String[] args) {
        char[] password = {'s', 'e', 'c', 'r', 'e', 't'};
        // Use password
        Arrays.fill(password, '\0'); // Clear immediately
        System.out.println("Password cleared");
    }
}
```

Q6. Create a String using `String.valueOf()`.

```
public class StringValueOf {
    public static void main(String[] args) {
        int num = 123;
        String s = String.valueOf(num);
        System.out.println(s); // "123"
    }
}
```

Q7. Check if two Strings are equal ignoring case.

```
public class CaseInsensitiveEquals {
    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = "HELLO";
        System.out.println(s1.equalsIgnoreCase(s2)); // true
    }
}
```

Q8. Get the length of a String.

```
public class StringLength {
    public static void main(String[] args) {
        String s = "Java";
        System.out.println(s.length()); // 4
    }
```

```
}
```

Q9. Convert a String to a character array.

```
public class StringToCharArray {
    public static void main(String[] args) {
        String s = "Hello";
        char[] c = s.toCharArray();
        System.out.println(Arrays.toString(c)); // [H, e, l, l, o]
    }
}
```

Q10. Create a String from a byte array.

```
public class StringFromByteArray {
    public static void main(String[] args) {
        byte[] b = {72, 101, 108, 108, 111}; // ASCII for "Hello"
        String s = new String(b);
        System.out.println(s); // Hello
    }
}
```

Medium Coding (10)

Q11. Demonstrate that modifying `char[]` after String creation doesn’t affect String.

```
public class CharArrayIndependence {
    public static void main(String[] args) {
        char[] c = {'H', 'i'};
        String s = new String(c);
        c[0] = 'B';
        System.out.println(s); // Hi (unchanged)
    }
}
```

Q12. Securely handle a password using `char[]` and clear it.

```
import java.util.Arrays;
public class SecurePasswordHandling {
    public static boolean authenticate(char[] password) {
        char[] expected = {'p', 'a', 's', 's'};
        boolean result = Arrays.equals(password, expected);
```

```
        Arrays.fill(password, '\0'); // Clear input
        Arrays.fill(expected, '\0'); // Clear expected
        return result;
    }

    public static void main(String[] args) {
        char[] input = {'p', 'a', 's', 's'};
        System.out.println("Authenticated: " + authenticate(input));
    }
}
```

Q13. Compare performance of `String` vs `StringBuilder` for concatenation.

```
public class ConcatenationPerformance {
    public static void main(String[] args) {
        long start = System.nanoTime();
        String s = "";
        for (int i = 0; i < 10000; i++) {
            s += "a"; // Inefficient
        }
        long end = System.nanoTime();
        System.out.println("String: " + (end - start) / 1e6 + " ms");

        start = System.nanoTime();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 10000; i++) {
            sb.append("a");
        }
        end = System.nanoTime();
        System.out.println("StringBuilder: " + (end - start) / 1e6 + " ms");
    }
}
```

Q14. Show that `String` literals are reused from pool.

```
public class StringPoolReuse {
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Java";
        System.out.println(s1 == s2); // true
    }
```

```
}
```

Q15. Create a String with compile-time concatenation.

```
public class CompileTimeConcatenation {
    public static void main(String[] args) {
        String s1 = "Hello" + " World"; // One object in pool
        String s2 = "Hello World";
        System.out.println(s1 == s2); // true
    }
}
```

Q16. Demonstrate that `new String("literal")` creates two objects.

```
public class TwoStringObjects {
    public static void main(String[] args) {
        // "Java" in pool, new String("Java") in heap
        String s1 = "Java";
        String s2 = new String("Java");
        System.out.println(s1 == s2); // false
    }
}
```

Q17. Use `String` in a hash map and show immutability safety.

```
import java.util.*;
public class StringAsKey {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        String key = "name";
        map.put(key, 10);
        key = key + "2"; // Creates new String, original key unchanged
        System.out.println(map.get("name")); // 10 (still works)
    }
}
```

Q18. Convert `StringBuilder` to `String` and show immutability.

```
public class StringBuilderToString {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Hello");
        String s = sb.toString(); // New immutable String
        sb.append(" World");
        System.out.println(s); // Hello (unchanged)
```

```
    }
}
```

Q19. Show that `String` hash code is cached.

```
public class StringHashCodeCaching {
    public static void main(String[] args) {
        String s = "Hello";
        long start = System.nanoTime();
        int h1 = s.hashCode();
        long mid = System.nanoTime();
        int h2 = s.hashCode();
        long end = System.nanoTime();
        System.out.println("First: " + (mid - start) + " ns, Second: " + (end - mid) + " ns");
        // Second call is faster due to caching
    }
}
```

Q20. Create a String from a subset of a character array.

```
public class SubstringFromCharArray {
    public static void main(String[] args) {
        char[] c = {'H', 'e', 'l', 'l', 'o'};
        String s = new String(c, 1, 3); // from index 1, length 3
        System.out.println(s); // "ell"
    }
}
```

Hard Coding (10)

Q21. Implement a custom immutable string class.

```
public final class MyString {
    private final char[] value;

    public MyString(char[] value) {
        this.value = value.clone(); // Defensive copy
    }

    public MyString(MyString other) {
        this.value = other.value.clone();
    }
```

```java
    public char charAt(int index) {
        return value[index];
    }

    public int length() {
        return value.length;
    }

    public MyString concat(MyString other) {
        char[] result = new char[this.length() + other.length()];
        System.arraycopy(this.value, 0, result, 0, this.length());
        System.arraycopy(other.value, 0, result, this.length(), other.length());
        return new MyString(result);
    }

    @Override
    public String toString() {
        return new String(value);
    }

    public static void main(String[] args) {
        MyString s1 = new MyString(new char[]{'H', 'i'});
        MyString s2 = s1.concat(new MyString(new char[]{'!'}));
        System.out.println(s1); // Hi
        System.out.println(s2); // Hi!
    }
}
```

Q22. Analyze memory usage of `String` vs `char[]` for large text.

```java
public class MemoryUsageAnalysis {
    public static void main(String[] args) {
        Runtime rt = Runtime.getRuntime();
        // String
        long before = rt.totalMemory() - rt.freeMemory();
        String s = "A".repeat(1000000);
        long after = rt.totalMemory() - rt.freeMemory();
        System.out.println("String memory: " + (after - before) / 1024.0 + " KB");
```

```java
        // char[]
        before = rt.totalMemory() - rt.freeMemory();
        char[] c = new char[1000000];
        Arrays.fill(c, 'A');
        after = rt.totalMemory() - rt.freeMemory();
        System.out.println("char[] memory: " + (after - before) / 1024.0 + " KB");
        // char[] uses less memory (no object overhead)
    }
}
```

Q23. Demonstrate security risk of `String` for passwords in heap dump.

```java
public class PasswordSecurityRisk {
    public static void main(String[] args) throws Exception {
        String password = "secret123";
        // Simulate processing
        Thread.sleep(10000); // During this time, password is in memory
        // In a real system, heap dump would reveal "secret123"
        System.out.println("Password was in memory for 10 seconds");
    }
}
```

Q24. Implement a string pool manually using `WeakHashMap`.

```java
import java.util.*;

public class ManualStringPool {
    private static final Map<String, WeakReference<String>> pool = new
WeakHashMap<>();

    public static String intern(String s) {
        WeakReference<String> ref = pool.get(s);
        if (ref != null) {
            String existing = ref.get();
            if (existing != null) return existing;
        }
        pool.put(s, new WeakReference<>(s));
        return s;
    }

    public static void main(String[] args) {
```

```
        String s1 = new String("Java");
        String s2 = new String("Java");
        String i1 = intern(s1);
        String i2 = intern(s2);
        System.out.println(i1 == i2); // true
    }
}
```

Q25. Show that `String` literals are created at class loading.

```
public class StringLiteralTiming {
    static {
        System.out.println("Class loading: creating String literals");
    }

    private static final String LITERAL = "Created at class load";

    public static void main(String[] args) {
        System.out.println(LITERAL);
    }
}
```

Q26. Compare `String`, `StringBuffer`, and `StringBuilder` performance.

```
public class StringBufferVsBuilder {
    public static void main(String[] args) {
        int n = 100000;
        // StringBuilder
        long start = System.nanoTime();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < n; i++) sb.append("a");
        long end = System.nanoTime();
        System.out.println("StringBuilder: " + (end - start) / 1e6 + " ms");

        // StringBuffer
        start = System.nanoTime();
        StringBuffer sbf = new StringBuffer();
        for (int i = 0; i < n; i++) sbf.append("a");
        end = System.nanoTime();
        System.out.println("StringBuffer: " + (end - start) / 1e6 + " ms");
        // StringBuilder is faster (no synchronization)
```

```
    }
}
```

Q27. Demonstrate that `String` objects can be garbage collected.

```
public class StringGCDemo {
    public static void main(String[] args) throws Exception {
        String s = new String("Temporary");
        s = null;
        System.gc();
        Thread.sleep(100); // Allow GC to run
        System.out.println("String object may be collected");
    }
}
```

Q28. Implement a method that safely converts `null` to empty string.

```
public class NullSafeString {
    public static String safeToString(String s) {
        return s == null ? "" : s;
    }

    public static void main(String[] args) {
        System.out.println(safeToString(null)); // ""
        System.out.println(safeToString("Hello")); // "Hello"
    }
}
```

Q29. Show the effect of `-XX:+CompactStrings` on memory (Java 9+).

```
public class CompactStringsDemo {
    public static void main(String[] args) {
        // For LATIN1 strings, Java 9+ uses byte[] instead of char[]
        String latin = "ABC"; // 1 byte per char
        String utf16 = "A\u00A9B"; // 2 bytes per char for non-LATIN1
        System.out.println("LATIN1 and UTF16 strings use compact representation in Java
9+");
    }
}
```

Q30. Create a secure string class that zeros out memory on finalize (educational).

```
public class SecureString {
```

```
    private char[] data;

    public SecureString(String s) {
        this.data = s.toCharArray();
    }

    public String toString() {
        return new String(data);
    }

    public void clear() {
        if (data != null) {
            Arrays.fill(data, '\0');
            data = null;
        }
    }

    @Override
    protected void finalize() throws Throwable {
        clear();
        super.finalize();
    }

    public static void main(String[] args) {
        SecureString s = new SecureString("secret");
        System.out.println(s);
        s.clear(); // Explicit clear
    }
}
```

TOPIC 17: String Methods, StringBuffer, StringBuilder, `toString()`, StringTokenizer & Practice Problems

Easy MCQs (10)

1. Which method returns the length of a `String`?
   A) `size()`
   B) `length()`
   C) `len()`

D) `count()`
Answer: B) `length()`

2. `StringBuilder` is:
A) Thread-safe
B) Not thread-safe
C) Immutable
D) Final
Answer: B) Not thread-safe

3. What does `s.trim()` do?
A) Removes all spaces
B) Removes leading and trailing whitespace
C) Converts to lowercase
D) Splits the string
Answer: B) Removes leading and trailing whitespace

4. The `toString()` method is defined in:
A) `String` class
B) `Object` class
C) `System` class
D) `Class` class
Answer: B) `Object` class

5. Which class is synchronized?
A) `StringBuilder`
B) `StringBuffer`
C) `String`
D) `StringTokenizer`
Answer: B) `StringBuffer`

6. What is the output?

```
String s = "Hello";
System.out.println(s.substring(1, 4));
```

A) `ell`
B) `ello`
C) `Hel`

D) `llo`
Answer: A) `ell`

7. `StringTokenizer` is in which package?
A) `java.io`
B) `java.util`
C) `java.lang`
D) `java.text`
Answer: B) `java.util`

8. Which method checks if a string starts with a prefix?
A) `startsWith()`
B) `prefix()`
C) `beginWith()`
D) `hasPrefix()`
Answer: A) `startsWith()`

9. What does `s.replace('a', 'b')` do?
A) Replaces first 'a' with 'b'
B) Replaces all 'a' with 'b'
C) Throws exception
D) Returns `null`
Answer: B) Replaces all 'a' with 'b'

10. The default capacity of `StringBuilder` is:
A) 8
B) 16
C) 32
D) 64
Answer: B) 16

Medium MCQs (10)

1. What is the output?

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
System.out.println(sb);
```

A) `Hello`
B) `Hello World`
C) Compilation error
D) `World`
Answer: B) `Hello World`

2. Which method splits a string into an array?
A) `tokenize()`
B) `split()`
C) `divide()`
D) `break()`
Answer: B) `split()`

3. What does `StringTokenizer.hasMoreTokens()` return?
A) Next token
B) `true` if more tokens exist
C) Token count
D) `false` always
Answer: B) `true` if more tokens exist

4. Why override `toString()`?
A) To enable string concatenation
B) To provide meaningful object representation
C) To improve performance
D) To make object immutable
Answer: B) To provide meaningful object representation

5. What is the time complexity of `StringBuilder.append()`?
A) O(1) amortized
B) O(n)
C) O(log n)
D) $O(n^2)$
Answer: A) O(1) amortized

6. Which is faster for single-threaded string building?
A) `String` concatenation
B) `StringBuffer`
C) `StringBuilder`
D) `StringTokenizer`

Answer: C) `StringBuilder`

7. What does `s.indexOf("abc")` return if not found?
A) `0`
B) `-1`
C) `null`
D) Exception
Answer: B) `-1`

8. `StringBuffer` and `StringBuilder` are:
A) Immutable
B) Mutable
C) Final
D) Abstract
Answer: B) Mutable

9. What is the output?

```
StringTokenizer st = new StringTokenizer("a,b,c", ",");
System.out.println(st.countTokens());
```

A) 1
B) 2
C) 3
D) 4
Answer: C) 3

10. Which method converts primitive to `String`?
A) `parse()`
B) `valueOf()`
C) `toString()`
D) Both B and C
Answer: D) Both B and C

Hard MCQs (10)

1. What is the internal structure of `StringBuilder`?
A) `char[]`
B) `byte[]`

C) `String`
D) `List<Character>`
Answer: A) `char[]`

2. After `sb.ensureCapacity(50)`, what happens if current capacity is 16?
A) Capacity becomes 50
B) Capacity becomes 34 (16*2+2)
C) No change
D) Exception
Answer: A) Capacity becomes 50

3. What does `s.split("\\.")` do?
A) Splits on literal dot
B) Splits on any character
C) Throws exception
D) Returns empty array
Answer: A) Splits on literal dot
(Dot is regex metacharacter; must escape)

4. Which is thread-safe for multi-threaded string building?
A) `StringBuilder`
B) `StringBuffer`
C) `String`
D) Both B and C
Answer: D) Both B and C

5. What is the output?

```
class Person {
   String name = "John";
   public String toString() { return name; }
}
System.out.println(new Person());
```

A) `Person@12345`
B) `John`
C) Compilation error
D) `null`
Answer: B) `John`

6. `StringTokenizer` is:
   A) Legacy class
   B) Preferred over `split()`
   C) Immutable
   D) Part of `java.lang`
   Answer: A) Legacy class
   (Java docs recommend `split()` or `Scanner`)

7. What is the capacity after `new StringBuilder("Hello")`?
   A) 5
   B) 16
   C) 21
   D) 32
   Answer: C) 21
   (16 + 5 = 21)

8. Which method is NOT in `String` class?
   A) `reverse()`
   B) `substring()`
   C) `toLowerCase()`
   D) `trim()`
   Answer: A) `reverse()`
   (Available in `StringBuilder`/`StringBuffer`)

9. What does `s.matches("a*b")` check?
   A) Starts with "a*b"
   B) Entire string matches regex "a*b"
   C) Contains "a*b"
   D) Ends with "b"
   Answer: B) Entire string matches regex "a*b"

10. Overriding `toString()` affects:
    A) `System.out.println(obj)`
    B) String concatenation (`obj + ""`)
    C) Debugging output
    D) All of the above
    Answer: D) All of the above

Coding Problems

Easy Coding (10)

Q1. Use `substring()` to extract middle name.

```
public class MiddleName {
    public static void main(String[] args) {
        String fullName = "John Michael Smith";
        String[] parts = fullName.split(" ");
        if (parts.length >= 3) {
            System.out.println(parts[1]); // Michael
        }
    }
}
```

Q2. Reverse a string using `StringBuilder`.

```
public class ReverseString {
    public static void main(String[] args) {
        String s = "Hello";
        String reversed = new StringBuilder(s).reverse().toString();
        System.out.println(reversed); // olleH
    }
}
```

Q3. Count words using `StringTokenizer`.

```
import java.util.StringTokenizer;
public class WordCount {
    public static void main(String[] args) {
        String text = "Java is fun";
        StringTokenizer st = new StringTokenizer(text);
        System.out.println(st.countTokens()); // 3
    }
}
```

Q4. Override `toString()` in a `Person` class.

```
class Person {
    String name; int age;
```

```
    Person(String n, int a) { name = n; age = a; }
    public String toString() {
        return name + " (" + age + ")";
    }
}

public class ToStringDemo {
    public static void main(String[] args) {
        System.out.println(new Person("Alice", 30)); // Alice (30)
    }
}
```

Q5. Replace all vowels with '*'.

```
public class ReplaceVowels {
    public static void main(String[] args) {
        String s = "Hello";
        s = s.replaceAll("[aeiouAEIOU]", "*");
        System.out.println(s); // H*ll*
    }
}
```

Q6. Check if a string is a palindrome (ignore case).

```
public class PalindromeCheck {
    public static void main(String[] args) {
        String s = "Madam";
        s = s.toLowerCase();
        String reversed = new StringBuilder(s).reverse().toString();
        System.out.println(s.equals(reversed)); // true
    }
}
```

Q7. Convert integer to binary string.

```
public class ToBinary {
    public static void main(String[] args) {
        int num = 10;
        String binary = Integer.toBinaryString(num);
        System.out.println(binary); // 1010
    }
}
```

Q8. Use `StringBuffer` to append strings.
```
public class StringBufferAppend {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        sb.append("Hello").append(" ").append("World");
        System.out.println(sb.toString()); // Hello World
    }
}
```

Q9. Split a CSV string using `split()`.
```
public class CSVSplit {
    public static void main(String[] args) {
        String csv = "apple,banana,cherry";
        String[] items = csv.split(",");
        for (String item : items) {
            System.out.println(item);
        }
    }
}
```

Q10. Trim and convert to uppercase.
```
public class TrimAndUpper {
    public static void main(String[] args) {
        String s = "  hello  ";
        s = s.trim().toUpperCase();
        System.out.println(s); // HELLO
    }
}
```

Medium Coding (10)

Q11. Tokenize a sentence with multiple delimiters using `StringTokenizer`.
```
import java.util.StringTokenizer;

public class MultiDelimiterTokenize {
    public static void main(String[] args) {
        String text = "apple,banana;cherry:orange";
        StringTokenizer st = new StringTokenizer(text, ",;:");
```

```
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
}
```

Q12. Build a large string efficiently using `StringBuilder`.

```
public class EfficientStringBuilding {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        for (int i = 1; i <= 1000; i++) {
            sb.append("Line ").append(i).append("\n");
        }
        System.out.println("Built " + sb.length() + " characters");
    }
}
```

Q13. Implement a method that formats a person’s name (override `toString()`).

```
class Employee {
    String firstName, lastName;
    Employee(String f, String l) { firstName = f; lastName = l; }
    public String toString() {
        return lastName.toUpperCase() + ", " + firstName;
    }
}

public class NameFormatter {
    public static void main(String[] args) {
        System.out.println(new Employee("John", "Doe")); // DOE, John
    }
}
```

Q14. Remove duplicate characters from a string.

```
import java.util.LinkedHashSet;

public class RemoveDuplicates {
    public static void main(String[] args) {
```

```
        String s = "programming";
        LinkedHashSet<Character> set = new LinkedHashSet<>();
        for (char c : s.toCharArray()) {
            set.add(c);
        }
        StringBuilder sb = new StringBuilder();
        for (char c : set) sb.append(c);
        System.out.println(sb.toString()); // progamin
    }
}
```

Q15. Count occurrences of a substring.

```
public class SubstringCount {
    public static void main(String[] args) {
        String s = "abababa";
        String target = "aba";
        int count = 0, fromIndex = 0;
        while ((fromIndex = s.indexOf(target, fromIndex)) != -1) {
            count++;
            fromIndex++;
        }
        System.out.println(count); // 3
    }
}
```

Q16. Convert a string to title case.

```
public class TitleCase {
    public static void main(String[] args) {
        String s = "hello world";
        String[] words = s.split(" ");
        StringBuilder sb = new StringBuilder();
        for (String word : words) {
            if (!word.isEmpty()) {
                sb.append(Character.toUpperCase(word.charAt(0)))
                  .append(word.substring(1).toLowerCase())
                  .append(" ");
            }
        }
        System.out.println(sb.toString().trim()); // Hello World
```

```
    }
}
```

Q17. Use `StringTokenizer` to parse key-value pairs.

```
import java.util.StringTokenizer;
public class ParseKeyValue {
    public static void main(String[] args) {
        String input = "name=Alice;age=30;city=NYC";
        StringTokenizer st = new StringTokenizer(input, ";");
        while (st.hasMoreTokens()) {
            String pair = st.nextToken();
            String[] kv = pair.split("=");
            if (kv.length == 2) {
                System.out.println(kv[0] + " -> " + kv[1]);
            }
        }
    }
}
```

Q18. Compare performance of `StringBuffer` vs `StringBuilder`.

```
public class BufferVsBuilder {
    public static void main(String[] args) {
        int n = 100000;
        // StringBuilder
        long start = System.nanoTime();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < n; i++) sb.append("a");
        long end = System.nanoTime();
        System.out.println("StringBuilder: " + (end - start) / 1e6 + " ms");

        // StringBuffer
        start = System.nanoTime();
        StringBuffer sbf = new StringBuffer();
        for (int i = 0; i < n; i++) sbf.append("a");
        end = System.nanoTime();
        System.out.println("StringBuffer: " + (end - start) / 1e6 + " ms");
    }
}
```

Q19. Implement a method that returns object state as JSON-like string.

```
class Point {
    int x, y;
    Point(int x, int y) { this.x = x; this.y = y; }
    public String toString() {
        return "{x:" + x + ", y:" + y + "}";
    }
}

public class JSONLikeToString {
    public static void main(String[] args) {
        System.out.println(new Point(10, 20)); // {x:10, y:20}
    }
}
```

Q20. Extract all numbers from a string.

```
public class ExtractNumbers {
    public static void main(String[] args) {
        String s = "abc123def456ghi";
        s = s.replaceAll("[^0-9]", " ");
        String[] numbers = s.trim().split("\\s+");
        for (String num : numbers) {
            if (!num.isEmpty()) System.out.println(num);
        }
    }
}
```

Hard Coding (10)

Q21. Implement a thread-safe string builder using `StringBuffer`.

```
public class ThreadSafeStringBuilder {
    private final StringBuffer buffer = new StringBuffer();

    public synchronized void append(String s) {
        buffer.append(s);
    }

    public synchronized String toString() {
        return buffer.toString();
```

```java
    }

    public static void main(String[] args) throws InterruptedException {
        ThreadSafeStringBuilder tsb = new ThreadSafeStringBuilder();
        Thread t1 = new Thread(() -> tsb.append("Hello"));
        Thread t2 = new Thread(() -> tsb.append("World"));
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println(tsb.toString()); // HelloWorld (order may vary)
    }
}
```

Q22. Parse a complex CSV with quoted fields using `StringTokenizer` (simplified).

```java
import java.util.StringTokenizer;
public class CSVParser {
    public static void main(String[] args) {
        // Simplified: assumes no commas inside quotes
        String csv = "\"John, Doe\",30,\"New York\"";
        StringTokenizer st = new StringTokenizer(csv, "\",");
        while (st.hasMoreTokens()) {
            String token = st.nextToken().trim();
            if (!token.isEmpty()) System.out.println(token);
        }
    }
}
```

Q23. Create a custom `toString()` that handles null fields.

```java
class SafeToString {
    String name; Integer age;
    SafeToString(String n, Integer a) { name = n; age = a; }
    public String toString() {
        return "SafeToString{name='" + (name != null ? name : "null") +
              "', age=" + (age != null ? age : "null") + "}";
    }

    public static void main(String[] args) {
        System.out.println(new SafeToString(null, null));
```

```
    }
}
```

Q24. Implement a string reverser that preserves word order but reverses letters.

```
public class WordLetterReverser {
    public static void main(String[] args) {
        String s = "Hello World";
        String[] words = s.split(" ");
        StringBuilder result = new StringBuilder();
        for (String word : words) {
            result.append(new StringBuilder(word).reverse()).append(" ");
        }
        System.out.println(result.toString().trim()); // olleH dlroW
    }
}
```

Q25. Build a query string from a map using `StringBuilder`.

```
import java.util.*;
public class QueryStringBuilder {
    public static void main(String[] args) {
        Map<String, String> params = new LinkedHashMap<>();
        params.put("name", "Alice");
        params.put("age", "30");
        StringBuilder sb = new StringBuilder("?");
        boolean first = true;
        for (Map.Entry<String, String> entry : params.entrySet()) {
            if (!first) sb.append("&");
            sb.append(entry.getKey()).append("=").append(entry.getValue());
            first = false;
        }
        System.out.println(sb.toString()); // ?name=Alice&age=30
    }
}
```

Q26. Tokenize and reassemble a sentence with custom delimiters.

```
import java.util.StringTokenizer;
public class ReassembleSentence {
    public static void main(String[] args) {
        String text = "apple,banana;cherry";
```

```
        StringTokenizer st = new StringTokenizer(text, ",;");
        StringBuilder sb = new StringBuilder();
        while (st.hasMoreTokens()) {
            sb.append(st.nextToken()).append(" ");
        }
        System.out.println(sb.toString().trim()); // apple banana cherry
    }
}
```

Q27. Implement a method that formats a stack trace as a string.

```
public class StackTraceToString {
    public static String getStackTrace(Throwable t) {
        StringBuilder sb = new StringBuilder();
        sb.append(t.toString()).append("\n");
        for (StackTraceElement element : t.getStackTrace()) {
            sb.append("\tat ").append(element.toString()).append("\n");
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        try {
            throw new RuntimeException("Test");
        } catch (Exception e) {
            System.out.println(getStackTrace(e));
        }
    }
}
```

Q28. Create a mutable string wrapper with `StringBuilder` and custom `toString()`.

```
public class MutableString {
    private final StringBuilder sb = new StringBuilder();

    public MutableString append(String s) {
        sb.append(s);
        return this;
    }

    public MutableString reverse() {
```

```
        sb.reverse();
        return this;
    }

    public String toString() {
        return sb.toString();
    }

    public static void main(String[] args) {
        MutableString ms = new MutableString();
        ms.append("Hello").reverse();
        System.out.println(ms); // olleH
    }
}
```

Q29. Parse a log line using `StringTokenizer` and extract fields.

```
import java.util.StringTokenizer;
public class LogParser {
    public static void main(String[] args) {
        String log = "2024-06-15 INFO User logged in";
        StringTokenizer st = new StringTokenizer(log, " ");
        if (st.countTokens() >= 3) {
            String date = st.nextToken();
            String level = st.nextToken();
            String message = "";
            while (st.hasMoreTokens()) {
                message += st.nextToken() + " ";
            }
            System.out.println("Date: " + date);
            System.out.println("Level: " + level);
            System.out.println("Message: " + message.trim());
        }
    }
}
```

Q30. Implement a secure `toString()` that hides sensitive fields.

```
class SecureUser {
    String username;
    String password; // sensitive
```

```
    SecureUser(String u, String p) { username = u; password = p; }

    public String toString() {
        return "SecureUser{username='" + username + "', password='[HIDDEN]'}";
    }

    public static void main(String[] args) {
        System.out.println(new SecureUser("admin", "secret"));
    }
}
```