}

IT Skill Test GIC Myanmar

Duration: 30 Minutes Total Questions: 8

1. You are debugging a Java method that is supposed to find the maximum element in an array of integers. The current implementation is as follows:

```
public static int findMax(int[] arr) {
    int max = arr[0];
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] > max)
            max = arr[i];
    }
    return max;
}
```

However, when testing with the array {-5, -2, -8, -1}, the method returns 0 instead of -1. What is the most likely cause of this issue?

A.int max = arr[0];

B.int max = 0;

C.for (int i = 0; i < arr.length; i++)

D.if (arr[i] > max)

2. You are working on a Java method to calculate the factorial of a given number. The current implementation is as follows:

```
public static long factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

However, for large inputs, this method throws a StackOverflowError. What is the best way to fix this issue while maintaining the functionality?

A.
```
public static long factorial(int n) {
      long result = 1;
      for (int i = 1; i <= n; i++) {
          result *= i;
      }
      return result;
  }
```

B.
```
public static long factorial(int n) {
      if (n < 0) throw new IllegalArgumentException("n must be non-negative");
      return n == 0 ? 1 : n * factorial(n - 1);
  }
```

C.
```
public static long factorial(int n) {
      return IntStream.rangeClosed(1, n).reduce(1, (x, y) -> x * y);
  }
```

D.
```
public static long factorial(int n) {
      return LongStream.rangeClosed(1, n).reduce(1, (x, y) -> x * y);
  }
```

3. You are reviewing code for a method that should return the nth Fibonacci number. The current implementation is:

```
public static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

What is the main issue with this implementation for large values of n?

A.It will throw an ArrayIndexOutOfBoundsException for large n

B.It has exponential time complexity, making it inefficient for large n

C.It will return incorrect results for n > 46 due to integer overflow

D.It will cause a StackOverflowError for n > 1000

4. In a Java application, you need to implement a thread-safe singleton class. Which of the following is the best implementation?

```
A.public class Singleton {
      private static Singleton instance;
      private Singleton() {}
      public static synchronized Singleton getInstance() {
          if (instance == null) {
              instance = new Singleton();
          }
          return instance;
      }
  }
```

```
B.public class Singleton {
      private static Singleton instance = new Singleton();
      private Singleton() {}
      public static Singleton getInstance() {
          return instance;
      }
  }
```

```
C.public class Singleton {
      private static volatile Singleton instance;
      private Singleton() {}
      public static Singleton getInstance() {
          if (instance == null) {
              synchronized (Singleton.class) {
                  if (instance == null) {
                      instance = new Singleton();
                  }
              }
          }
          return instance;
      }
  }
```

```
D.public enum Singleton {
        INSTANCE;
        // methods here
    }
```

5. You are working on a Java method that needs to find all prime numbers up to a given number n. The current implementation is as follows:

```
public static List<Integer> findPrimes(int n) {
    List<Integer> primes = new ArrayList<>();
    for (int i = 2; i <= n; i++) {
        boolean isPrime = true;
        for (int j = 2; j < i; j++) {
            if (i % j == 0) {
                isPrime = false;
                break;
            }
        }
        if (isPrime) {
            primes.add(i);
        }
    }
    return primes;
}
```

Which of the following optimizations would most significantly improve the performance of this method?

A.Change the outer loop to: for (int i = 2; i <= Math.sqrt(n); i++)

B.Change the inner loop to: for (int j = 2; j <= Math.sqrt(i); j++)

C.Use a boolean array to implement the Sieve of Eratosthenes algorithm

D.Use parallel streams to check primality concurrently

6. You are reviewing code for a Java method that should return the longest common substring of two given strings. The current implementation is:

```
public static String longestCommonSubstring(String s1, String s2) {
    String longer = s1.length() > s2.length() ? s1 : s2;
    String shorter = s1.length() > s2.length() ? s2 : s1;
    String result = "";
    for (int i = 0; i < shorter.length(); i++) {
        for (int j = i + 1; j <= shorter.length(); j++) {
            String substring = shorter.substring(i, j);
            if (longer.contains(substring) && substring.length() >
result.length()) {
                result = substring;
            }
        }
    }
    return result;
}
```

What is the time complexity of this implementation?

A.O(n)

B.O(n log n)

C.O(n^2)

D.O(n^3)

7. You are working on a Java application that processes customer orders. The following method is supposed to calculate the total price of an order, including a discount if applicable. However, it's not working correctly for some orders. Review the code and identify the bug:

```
public double calculateTotalPrice(double basePrice, int quantity, boolean
isPreferredCustomer) {
    double total = basePrice * quantity;
    if (isPreferredCustomer || total > 1000) {
        total -= total * 0.1;
    } else if (total > 500) {
        total -= total * 0.05;
    }
    return total;
}
```

A.total -= total * 0.1;

B.if (isPreferredCustomer || total > 1000) {

C.double total = basePrice * quantity;

D.return total;

8. You are maintaining a legacy system that uses a custom logging mechanism. The senior developer asks you to refactor the following method to improve its performance and readability:

```
public void logMessage(String message, int level) {
    String logEntry = new Date().toString() + " - " + message;
    if (level == 1) {
        System.out.println("INFO: " + logEntry);
    } else if (level == 2) {
        System.out.println("WARNING: " + logEntry);
    } else if (level == 3) {
        System.out.println("ERROR: " + logEntry);
    } else {
        System.out.println("UNKNOWN: " + logEntry);
    }
}
```

Which of the following refactored versions would be most appropriate?

```
A.public void logMessage(String message, int level) {
      String[] levels = {"UNKNOWN", "INFO", "WARNING", "ERROR"};
      String logEntry = new Date().toString() + " - " + message;
      System.out.println(levels[Math.min(level, 3)] + ": " + logEntry);
```

```
  }
```

```
B.public void logMessage(String message, int level) {
      String logEntry = new Date().toString() + " - " + message;
      switch(level) {
          case 1: System.out.println("INFO: " + logEntry); break;
          case 2: System.out.println("WARNING: " + logEntry); break;
          case 3: System.out.println("ERROR: " + logEntry); break;
          default: System.out.println("UNKNOWN: " + logEntry);
      }
  }
```

```
C.public void logMessage(String message, int level) {
      System.out.println(new Date().toString() + " - " +
          (level == 1 ? "INFO" : level == 2 ? "WARNING" : level == 3 ? "ERROR" :
  "UNKNOWN") +
          ": " + message);
  }
```

```
D.public void logMessage(String message, int level) {
      String logEntry = new Date().toString() + " - " + message;
      if (level >= 1 && level <= 3) {
          String[] levels = {"INFO", "WARNING", "ERROR"};
          System.out.println(levels[level-1] + ": " + logEntry);
      } else {
          System.out.println("UNKNOWN: " + logEntry);
      }
  }
```