

Question 1

```
public class MyThread {
    public static void main(String[] args) {
        thread.setSequence();
        for(int i = 1; i <= 10; i++) {
            Thread t = new Thread(new thread(i));
            t.setName(i + "");
            t.start();
        }
    }
}
```

```
class thread implements Runnable {
    private static HashMap< String, String> sequence = new HashMap<String, String>();
```

```
    public static final Object lock = new Object();
    public static String turn = "1";
    private int startValue = 0;
    private AtomicInteger counter = new AtomicInteger(1);
```

```
    public thread(int startValue){
        this.startValue = startValue;
    }
```

@Override

```
    public void run() {
        while (!counter.equals(10)){
            synchronized (lock) {
                if(Thread.currentThread().getName().equals(turn)){
                    System.out.print(startValue + " ");
                    startValue += 10;
                    counter.incrementAndGet();
                    turn = getNextTurn(turn);
                    try {
                        this.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
```

```

    }
    else{
        try {
            this.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    this.notifyAll();
}
}
}

```

```

public static void setSequence(){
    for (int i = 1; i <= 10; i++)
        if (i == 10)
            sequence.put(i + "", 1 + "");
        else
            sequence.put(i + "", (i + 1) + "");
}

```

```

public static String getNextTurn(String currentTurn){
    return sequence.get(currentTurn);
}
}

```

Question 2

```
import java.util.Scanner;

public class CountDivisorsUsingThreads {

    private final static int MAX = 100000;
    private volatile static int maxDivisorCount = 0;
    private volatile static int intWithMaxDivisorCount;
    synchronized private static void report(int maxCountFromThread,
        int intWithMaxFromThread) {
        if (maxCountFromThread > maxDivisorCount) {
            maxDivisorCount = maxCountFromThread;
            intWithMaxDivisorCount = intWithMaxFromThread;
        }
    }

    private static class CountDivisorsThread extends Thread {
        int min, max;
        public CountDivisorsThread(int min, int max) {
            this.min = min;
            this.max = max;
        }
        public void run() {
            System.out.println("Thread " + this + " testing range " + min + " to " + max);
            long startTime = System.currentTimeMillis();
            int maxDivisors = 0;
            int whichInt = 0;
            for (int i = min; i < max; i++) {
                int divisors = countDivisors(i);
                if (divisors > maxDivisors) {
                    maxDivisors = divisors;
                    whichInt = i;
                }
            }
        }
    }

    private static void countDivisorsWithThreads(int numberOfThreads) {
        System.out.println("\nCounting divisors using " +
            numberOfThreads + " threads...");
        long startTime = System.currentTimeMillis();
        CountDivisorsThread[] worker = new CountDivisorsThread[numberOfThreads];
        int integersPerThread = MAX/numberOfThreads;
        int start = 1; // Starting point of the range of ints for first thread.
```

```

int end = start + integersPerThread - 1; // End point of the range of ints.
for (int i = 0; i < numberOfThreads; i++) {
    if (i == numberOfThreads - 1) {
        end = MAX; // Make sure that the last thread's range goes all
                    // the way up to MAX. Because of rounding, this
                    // is not automatic.
    }
    worker[i] = new CountDivisorsThread( start, end );
    start = end+1; // Determine the range of ints for the NEXT thread.
    end = start + integersPerThread - 1;
}
maxDivisorCount = 0;
for (int i = 0; i < numberOfThreads; i++)
    worker[i].start();
for (int i = 0; i < numberOfThreads; i++) {
    // Wait for each worker thread to die, because the results
    // are not complete until all threads have completed and
    // reported their results.
    while (worker[i].isAlive()) {
        try {
            worker[i].join();
        }
        catch (InterruptedException e) {
        }
    }
}
long elapsedTime = System.currentTimeMillis() - startTime;
System.out.println("\nThe largest number of divisors " +
    "for numbers between 1 and " + MAX + " is " + maxDivisorCount);
System.out.println("An integer with that many divisors is " +
    intWithMaxDivisorCount);
System.out.println("Total elapsed time: " +
    (elapsedTime/1000.0) + " seconds.\n");
}

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int numberOfThreads = 0;
    while (numberOfThreads < 1 || numberOfThreads > 10) {
        System.out.print("How many threads do you want to use (1 to 10) ? ");
        numberOfThreads = in.nextInt();
        if (numberOfThreads < 1 || numberOfThreads > 10)
            System.out.println("Please enter a number from 1 to 10 !");
    }
    countDivisorsWithThreads(numberOfThreads);
}

```

```

    }
    public static int countDivisors(int N) {
        int count = 0;
        for (int i = 1; i <= N ; i++) {
            if ( N % i == 0 )
                count ++;
        }
        return count;
    }
}

```

Question 3

```

/**
 * @param {string} s
 * @return {boolean}
 */
//create a map which relates left and right parentheses
var map = {
    "(": ". ",
    "[": ". ]",
    "{": ". "
}
var isValid = function(s) {
    var stack = [];
    for (var i = 0; i < s.length; i++) {
        var item = s[i];
        //item is a variable whose value is the element at position i of the string
        if (map[item]) {
            //if item is a value, add that value i.e. '}' to the stack
            stack.push(map[item]);
        } else {
            if (item !== stack.pop()) {
                //if item is not equal to the first element of the stack return false
                return false;
            }
        }
    }
    //return the stack when its elements have been exhausted
    return stack.length === 0;
};

```

Question 4

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
<pre>
<script>
  // var n = prompt("number of rows in the christmas tree?", "8");
  var n = 4;

  if(n<3){
    document.write("Too small")
  }
  else if(n<=5){
    christmasTree(n);

    for(a=1; a<=2; a++)
    {
      for(b=0; b<(n*2-4)/2; b++)
      {
        document.write(" ")
      }
      for(b=0;b<=2;b++)
      {
        document.write("0");
      }
      document.write("<br>");
    }
  }
  else{
    christmasTree(n);
    tree(n);

  }

  function christmasTree(n) {
    for(i=1; i<=n; i++)
    {
      for(j=i; j<n; j++)
```

```

        {
            document.write(" ");
        }
        for(j=1; j<=(2*i-1); j++)
        {
            if(i==1){
                document.write("***");
            }
            else{
                document.write("0");
            }
        }

        document.write("<br>");
    }
}

function tree(n) {
    for(a=1; a<=3; a++)
    {
        for(b=0; b<=(n*2-5)/2; b++)
        {
            document.write(" ")
        }
        for(b=0;b<=2;b++)
        {
            document.write("0");
        }
        document.write("<br>");
    }
}

</script>
</pre>
</body>
</html>

```