Binary search, plus more programming with methods, objects, and arrays

1. Binary search

a. Write a method that creates an array of 1000 random integers between 0 and 99, and returns the array.

```java
// need to have this at top of the class:
// import java.util.Random;
public static int[] createRandomArray()
{
    Random rand = new Random();
    int[] ret = new int[1000];
    for(int i=0; i<ret.length; i++) {
        ret[i] = rand.nextInt(100);
    }
    return ret;
}
```

b. Write a method that creates an array of `num` random integers between 0 and `max`, and returns the array.

```java
// need to have this at top of the class:
// import java.util.Random;
public static int[] createRandomArray(int num, int max)
{
    Random rand = new Random();
    int[] ret = new int[num];
    for(int i=0; i<ret.length; i++) {
        ret[i] = rand.nextInt(max);
    }
    return ret;
}
```
c. Write a method that takes an integer array as a parameter. The method should ask the user for 5 integers between 0 and 99, and print out the position of each one in the array (-1 if it can’t find it). Use a linear search method.

```java
// need to have this at the top of the class:
// import java.util.Scanner;
public static void find5Numbers(int [] array) {
    Scanner kb = new Scanner(System.in);
    for(int i=0; i<5; i++) {
        System.out.println("Please enter an integer (0-99)");
        int key = kb.nextInt();
        System.out.println(linearSearch(array, key));
    }
}

public static int linearSearch(int [] D, int key) {
    for(int i=0; i<D.length; i++) {
        if(D[i]==key) {
            return i;
        }
    }
    return -1;
}
```

d. Now modify the method so that it uses binary search instead.

```java
// need to have this at the top of the class:
// import java.util.Scanner;
// import java.util.Arrays;
public static void find5Numbers(int [] array) {
    Arrays.sort(array);
    Scanner kb = new Scanner(System.in);
    for(int i=0; i<5; i++) {
        System.out.println("Please enter an integer (0-99)");
        int key = kb.nextInt();
        System.out.println(Arrays.binarySearch(array, key));
    }
}
e. What needs to happen before you can run binary search? What do you need to import in order for this to happen?

need: import java.util.Arrays; need to sort the array before running binary search (using Arrays.sort()).

f. Which method will be faster, the linear search method or the binary search method? Why?

Binary search is faster than linear search. However, in order to use binary search, you first need to sort once. So is sorting once + binary search 5 times faster or slower than sorting zero times + linear search 5 times? It’s hard to tell. There’s no clear answer to this, unless you have more information (how big the array is, how slow/fast the sort algorithm is, etc.).

g. Look at the example array below. For each key, indicate the positions in the array (the indexes, not the values) that a binary search would visit if it was searching for that key.

<table>
<thead>
<tr>
<th></th>
<th>-20</th>
<th>-12</th>
<th>-9</th>
<th>1</th>
<th>4</th>
<th>16</th>
<th>21</th>
<th>67</th>
<th>75</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Key:  4   Positions visited during binary search:  4
Key:  16  Positions visited during binary search:  4, 7, 5
Key:  -25  Positions visited during binary search:  4, 1, 0
Key:  101 Positions visited during binary search:  4, 7, 8, 9
Key:  45  Positions visited during binary search:  4, 7, 5, 6
Key:  -9  Positions visited during binary search:  4, 1, 2
2. More programming with methods
   a. Write a method called isEven that returns whether an integer is even or not.

   ```java
   public static boolean isEven(int N) {
       return (N % 2 == 0);
   }
   ```

   b. **Write a method called isPerfect that returns whether an integer is a perfect number or not.** A perfect number is one that is equal to the sum of its factors (not including the number itself as a factor). For instance, 6 is a perfect number because its factors are 1, 2, and 3, and 1+2+3=6. 28 is the next perfect number, because 1+2+4+7+14=28.

   ```java
   public static boolean isPerfect(int N) {
       int factorSum = 0;
       for (int possibleFactor = 1; possibleFactor < N; possibleFactor++) {
           if (N % possibleFactor == 0) { // actual factor!
               factorSum += possibleFactor;
           }
       }
       return (factorSum == N);
   }
   ```

   c. Write a loop that finds the next perfect number after 28. Use the isPerfect method from above (you don’t have to solve (b) to do this one, just pretend that you have a correct version of isPerfect() defined).

   ```java
   public static int perfectNumAfter28() {
       for (int maybePerfect = 29; true; maybePerfect++) {
           if (isPerfect(maybePerfect)) { // found one!
               return maybePerfect;
           }
       }
   }
   ```
d. Write a method that returns the average value of a 2D array of doubles.

```java
public static double averageOf2DArray(double [][] array) {
    double sum = 0.0;
    int numElements = 0;
    for(int row=0; row<array.length; row++) {
        for(int col=0; col<array[row].length; col++) {
            sum += array[row][col];
            numElements++;
        }
    }
    return sum / numElements;
}
```

e. Write a method that returns the “average” Point from an array of Points. The “average” Point has an x coordinate that is the average of all the x coordinates in the array, and a y coordinate that is the average of all the y coordinates in the array.

```java
public static Point averagePoint(Point [] points) {
    int sumX = 0;
    int sumY = 0;
    for(int i=0; i<points.length; i++) {
        sumX += points[i].x;
        sumY += points[i].y;
    }
    int avgX = sumX / points.length;
    int avgY = sumY / points.length;
    return new Point(avgX, avgY);
}
```

f. Write a method that returns the Point object in an array that is farthest from (new Point(0,0)) -- the origin.

```java
public static Point farthestPoint(Point [] points) {
    Point origin = new Point();
    Point farthest = points[0];
    for(int i=1; i<points.length; i++) {
        if( points[i].distance(origin) > 
            farthest.distance(origin))
        {
            farthest = points[i];
        }
    }
    return farthest;
}
```
g. Write a class that calls each of the methods you defined above.

```java
import java.awt.Point;
import java.util.Arrays;

public class Miscellaneous {
    // assume definitions of isEven, isPerfect, etc.
    // are written here

    public static void main(String[] args) {
        int perfectAfter28 = perfectNumAfter28();
        System.out.println("first perfect num after 28 =");
        System.out.println(perfectAfter28);
        System.out.println("is it even?");
        System.out.println(isEven(perfectAfter28));

        double[][] arr2D = {{7.0, 2.9},
                            {-12.5, 2.2, 3.7},
                            {2.1, 1.5}};
        System.out.println(Arrays.toString(arr2D));
        System.out.println("avg = "+averageOf2DArray(arr2D));

        Point[] ps = {new Point(0,1), new Point(2,-5),
                      new Point(-5,-4), new Point(3,7),
                      new Point(2,21), new Point(15,13)};
        System.out.println(Arrays.toString(ps));
        System.out.println("avg point = "+averagePoint(ps));
        System.out.println("avg point = "+farthestPoint(ps));
    }
}
```