Solution to Practice Problems: Searching Arrays

0. What must be true of an array before it can be searched using binarySearch?

1. What built-in Java command can sort an array?

2. Tracing Algorithms
For each call to the binarySearch method below, write which elements the search procedure visits.

array x:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>
</tr>
<tr>
<td>-19</td>
<td>-12</td>
<td>4</td>
<td>9</td>
<td>21</td>
<td>22</td>
<td>45</td>
<td>51</td>
<td>99</td>
<td>103</td>
</tr>
</tbody>
</table>

```java
int pos = Arrays.binarySearch(x, 21);
```

1) it first searches the middle point between positions 0 and 9, or position 4 (21).
2) It sees that the element at position 4 is exactly equal to the key, so it stops and returns that position (4).

```java
int pos = Arrays.binarySearch(x, 51);
```

1) it first searches the middle point between positions 0 and 9, or position 4 (21).
2) It sees that the element at position 4 is less than the key (21<51), so it rules out all the elements in positions 0 to 4.
3) Next, it checks the midway point between position 5 and 9, or position 7.
4) It sees that the element at position 7 is equal to the key, so it returns the position, 7.

You can write these steps as a sequence of positions, like this:
4 → 7, return 7

```java
int pos = Arrays.binarySearch(x, 9);
```

4 → 1 → 2 → 3, return 3

```java
int pos = Arrays.binarySearch(x, -15);
```

4 → 1 → 0, return -1 (because it can't find -15)
array y:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;abba&quot;</td>
<td>&quot;ccr&quot;</td>
<td>&quot;elvis&quot;</td>
<td>&quot;gomez&quot;</td>
<td>&quot;juno&quot;</td>
<td>&quot;mogwai&quot;</td>
<td>&quot;prince&quot;</td>
<td>&quot;rem&quot;</td>
<td>&quot;u2&quot;</td>
<td>&quot;who&quot;</td>
</tr>
</tbody>
</table>

int pos = Arrays.binarySearch(y, "juno");
4, return 4

int pos = Arrays.binarySearch(y, "prince");
4 → 7 → 5 → 6, return 6

int pos = Arrays.binarySearch(y, "who");
4 → 7 → 8 → 9, return 9

int pos = Arrays.binarySearch(y, "beirut");
4 → 1 → 0, return -1 (can't find "beirut")

### 3. Writing short methods involving search and sort

a. Translate the binary search algorithm into your own binarySearch() method.

```java
public static int binarySearch(int[] x, int key) {
    int left = 0;
    int right = x.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (x[mid] < key) {
            left = mid + 1; // discard everything between 0 and mid
        } else if (x[mid] > key) {
            right = mid - 1; // discard everything between mid and x.length-1
        } else {
            return mid;
        }
    }
    return -1; // couldn't find the key anywhere
}
```
b. Write a method that takes an int array X as an argument. It should return the median value of the array. The median of a set of numbers is defined as the number in the middle position, when the numbers are arranged from smallest to largest.

```java
public static int median(int[] X)
{
    Arrays.sort(X); // first, arrange the elements of X in ascending order
    int mid = X.length / 2;
    return X[mid]; // return the number in the middle position

    // technically, if there are an even number of elements in X,
    // the median should be an average between the two middle elements.
    // can you figure out how to modify this method to make that happen?
}
```

c. Write a method that takes an int array X as an argument. It should return true if 0 is in the array, and false otherwise.

```java
public static boolean containsZero(int[] X)
{
    // need to sort before searching!
    Arrays.sort(X);

    int pos = Arrays.binarySearch(X, 0);
    return (pos >= 0);
}
```