Binary Search

The ***binary search*** algorithm is a very well-known algorithm in computer science. Given a ***sorted*** array (or ArrayList) of items, the algorithm is used to test whether a candidate item (the ***key***) is in the array or not. Often the algorithm is written in such a way that it returns the *index* within the array at which the key is located, rather than only a boolean value indicating whether key was found or not.

Recall that the *linear search* algorithm solves this same problem but does not assume the array is sorted. By beginning with a sorted array, binary search usually runs much faster than linear search.

Binary search begins by identifying the item in the middle of the array and comparing it against the key. If the middle item matches the key, then the index of the middle item is returned. If the middle item is larger than the key, the algorithm repeats itself on the sub-array to the left of the middle item; if the middle item is smaller than the key, the algorithm repeats on the sub-array to the right of the middle item. If the remaining sub-array to be searched ever becomes empty, we know the key is not in the array.

The "repetition" part of the algorithm can be implemented using iteration (a loop) or recursion. In either case, the algorithm maintains two variables to keep track of the current upper and lower indices for the portion of the array that could potentially contain the key.

We present the algorithm as a search over a sorted array of integers, though any data type can be used as long as the array is sorted in some fashion.

1. We are given
    a. an array `array` of size `n`, indexed from `0` to `n-1`
    b. an integer `key` to look for in the array
    c. an integer `low` that is the lowest index in the array that could contain the key
    d. an integer `high` that is the highest index in the array that could contain the key
2. If low > high, then the item is not found (return `-1`)
3. Compute the middle index in the array.
4. If the item at the middle index is the key, return that index.
5. If the item at the middle index is greater than the key, repeat from step 2, on the left sub-array.
6. If the item at the middle index is less than the key, repeat from step 2 on the right sub-array.

For steps 5 and 6, if using recursion, the "repeat" part is done by calling your binary search function with new argument values for low or high.

```
// Iterative binary search --- remember the array must be sorted!

public static int binarySearchIter(int[] array, int key) {
    int low = 0;                          // far left index
    int high = array.length - 1;          // far right index

    while (high >= low) {
        int mid = (low + high) / 2;       // find midpoint

        if (array[mid] > key) {           // if middle element > key
            high = mid - 1;               //   next time, look in left half of array
        }

        else if (array[mid] < key) {      // if middle element < key
            low = mid + 1;                //   next time, look in right half of array
        }

        else {                            // if middle element == key
            return mid;                   //   return this middle element index
        }
    }
    return -1; // key not found
}

// Recursive binary search.
public static int binarySearchRec(int[] array, int key) {
    return binarySearchRec(array, key, 0, array.length - 1);
}

// Helper method for above.
private static int binarySearchRec(int[] array, int key, int low, int high) {
    if (low > high) {                     // base case #1 - if item is not found
        return -1;
    }

    int mid = (low + high) / 2;

    if (array[mid] == key) {              // base case #2: if array[mid] == key
        return mid;
    }

    else if (array[mid] > key) {
        return binarySearchRec(array, key, low, mid - 1);
    }

    else {
        return binarySearchRec(array, key, mid + 1, high);
    }
}
```