Building Java Programs

Chapter 13
Searching

reading: 13.3

Binary search (13.1)

- **binary search**: Locates a target value in a *sorted* array/ list by successively eliminating half of the array from consideration.
  - How many elements will it need to examine? **O(log N)**
  - Can be implemented with a loop or recursively
  - Example: Searching the array below for the value **42**:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

min mid max

Binary search code

```
// Returns the index of an occurrence of target in a,
// or a negative number if the target is not found.
// Precondition: elements of a are in sorted order
public static int binarySearch(int[] a, int target) {
    int min = 0;
    int max = a.length - 1;

    while (min <= max) {
        int mid = (min + max) / 2;
        if (a[mid] < target) {
            min = mid + 1;
        } else if (a[mid] > target) {
            max = mid - 1;
        } else {
            return mid;    // target found
        }
    }

    return -(min + 1);     // target not found
}
```

Recursive binary search (13.3)

- Write a recursive `binarySearch` method.
  - If the target value is not found, return its negative insertion point.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

```
int index  = binarySearch(data, 42);  // 10
int index2 = binarySearch(data, 66);  // -14
```

Exercise solution

```
// Returns the index of an occurrence of the given value in
// the given array, or a negative number if not found.
// Precondition: elements of a are in sorted order
public static int binarySearch(int[] a, int target) {
    return binarySearch(a, target, 0, a.length - 1);
}

// Recursive helper to implement search behavior.
private static int binarySearch(int[] a, int target,
                                int min, int max) {
    if (min > max) {
        return -1;          // target not found
    } else {
        int mid = (min + max) / 2;
        if (a[mid] < target) {          // too small; go
  right
            return binarySearch(a, target, mid + 1, max);
        } else if (a[mid] > target) {  // too large; go left
            return binarySearch(a, target, min, mid - 1);
        } else {
            return mid;    // target found; a[mid] == target
        }
    }
}
```