Lecture 7

Recursion 1

- Binary Search
- Iterative Implementation: search1
- Recursive Implementation: search2
At the end of Lecture 6, we ran a *binary search*.

- A number between 1 and 100 was chosen.
- The initial guess was 50, which was too low.
- The next guess was 75, which was too low.
- The next guess was 87, which was too high.
- The next guess was 81, which was too high.
- The next guess was 84, which was too high.
- The next guess was 82, which is correct.

At each step, there is a range of indices that could contain the value, and we guess the middle of the range.

The same approach works whenever the data array is sorted. Furthermore, it allows us to either find a particular value, or deduce that it is not in the array.

In this lecture, we’ll implement binary search in two ways.
Iteration vs Recursion

In Computer Science term *iteration* is most closely associated with doing one thing at a time. The most commonly associated control structure is a loop.

The term recursion is most closely associated with splitting a task into one more more subtasks. The most commonly associated control structure is a function that calls itself.

<table>
<thead>
<tr>
<th><strong>iteration</strong></th>
<th><strong>recursion</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>/ˌɪtəˈreɪʃən/</td>
<td>/rəˈkərZhən/</td>
</tr>
<tr>
<td>noun</td>
<td>noun</td>
</tr>
<tr>
<td>repetition of a mathematical or computational procedure applied to the result of a previous application, typically as a means of obtaining successively closer approximations to the solution of a problem.</td>
<td><strong><a href="https://www.oxforddictionaries.com/definition/iteration">Oxford Dictionary's definition.</a> <a href="https://www.oxforddictionaries.com/definition/recursion">Oxford Dictionary's definition.</a> (Is this a joke?)</strong></td>
</tr>
<tr>
<td>a new version of a piece of computer hardware or software.</td>
<td><strong><a href="https://www.oxforddictionaries.com/definition/iteration">Oxford Dictionary's definition.</a> <a href="https://www.oxforddictionaries.com/definition/recursion">Oxford Dictionary's definition.</a> (Is this a joke?)</strong></td>
</tr>
<tr>
<td>plural noun: iterations</td>
<td>plural noun: recursions</td>
</tr>
</tbody>
</table>

Some problems can naturally be solved using iteration or recursion.

When recursion is possible, it is often (a) cleaner, and (b) more difficult conceptually (at first).
The Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, … Implemented iteratively (left) and recursively (right) [source].

Is the $n == 2$ base case needed? What about the $n == 1$ base case?

Which implementation is more efficient? What does the recursive function recalculate? Could this be avoided?
Starter Code for **BinarySearch.java**.

We will implement `search1` and `search2` using iteration and recursion, respectively.

Note: Updated `~/.nanorc` will be posted after today's lecture.
Iterative Implementation
Discussion: Binary Search 1 — Iterative Approach

Discuss how to implement the function `search1`, which is our iterative approach to binary search in `BinarySearch.java`.

Start by talking with your neighbor. Then we'll discuss this as a group. Finally, you’ll have time to write your own version.

- Which variables will you use?
- When is the search finished?
- How does division work in Java? (Does it round up or down, or give a floating point number?)
- Can you avoid off-by-one errors? (There is some tricky mathematics.)
// Searches for value f in array A and returns true or false.
// Implemented using iteration.
public static Boolean search1(int[] a, int f) {
    int left, right, middle;
    left = 0;
    right = a.length - 1;
    while (left <= right) {
        middle = left + (right - left)/2;
        if (a[middle] == f) {
            return true;
        } else if (f < a[middle]) {
            left = left;
            right = middle - 1;
        } else {
            left = middle + 1;
            right = right;
        }
    }
    return false;
}
Recursive Implementation
Discussion: Binary Search 2 — Recursive Approach

Discuss how to implement the function search2, which is our recursive approach to binary search in BinarySearch.java.

Start by talking with your neighbor. Then we’ll discuss this as a group. Finally, you’ll have time to write your own version.

Discuss your ideas with a neighbor for 3 minutes. Then you’ll have time to try writing search2.

- Which variables will you use?
- When is the search finished (i.e., what are the base cases)?
- Do you want to change the function signature of search2? Why?
- Is the recursive approach faster or slower? How much memory does it use?
// Searches for value f in array A and returns true or false.
// Implemented using recursion.
public static boolean search2(int[] a, int f) {
    return search2rec(a, f, 0, a.length-1);
}

// This function is used by search2.
private static boolean search2rec(int[] a, int f, int left, int right) {
    int middle;

    // Base case: The range is empty.
    if (left > right) return false;

    // Check the middle fo the range and recurse as needed.
    middle = left + (right - left)/2;
    if (a[middle] == f) {
        return true;
    } else if (f < a[middle]) {
        return search2rec(a, f, left, middle - 1);
    } else {
        return search2rec(a, f, middle + 1, right);
    }
}

Finished search2 which is a stub for search2rec.