Lecture 7

Recursion 1

- Binary Search
- Iterative Implementation: `search1`
- Recursive Implementation: `search2`

Binary Search

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 use 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.

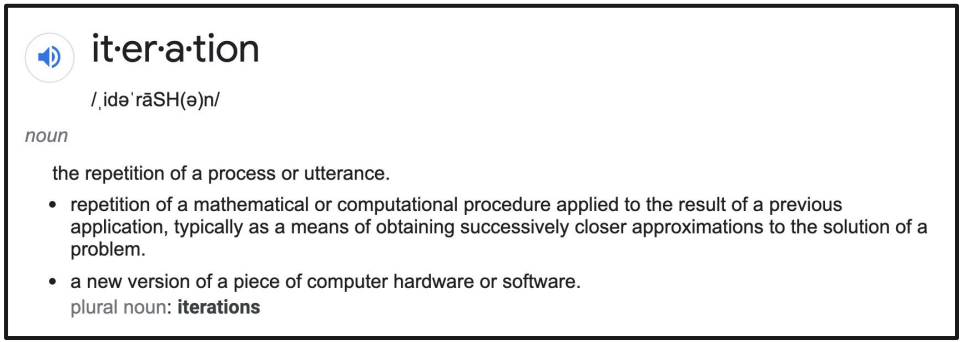
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Running binary search on 1 – 100
for the secret number 82.

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.



it·er·a·tion

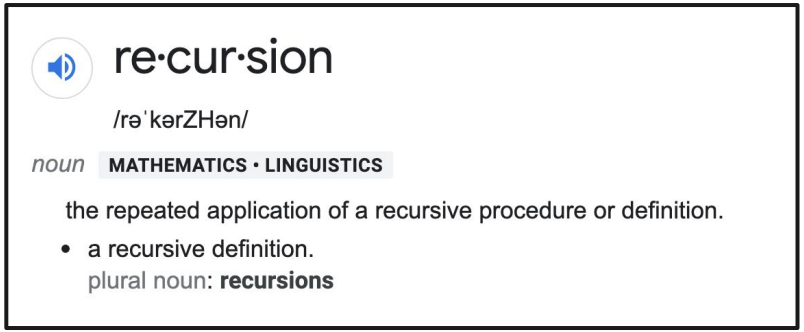
/ˌidəˈrāSH(ə)n/

noun

the repetition of a process or utterance.

- 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.
- a new version of a piece of computer hardware or software.
  plural noun: **iterations**

Oxford Dictionary's definition.



re·cur·sion

/rəˈkərZHən/

noun MATHEMATICS · LINGUISTICS

the repeated application of a recursive procedure or definition.

- a recursive definition.
  plural noun: **recursions**

Oxford Dictionary's definition.
(Is this a joke?)

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).

GNU nano 4.8 Iterative.java

```java
// Source: https://www.guru99.com/fibonacci-series-java.html

// Using For Loop
public class FibonacciExample {

    public static void main(String[] args)
    {
        // Set it to the number of elements you want in the Fibo
        int maxNumber = 10;
        int previousNumber = 0;
        int nextNumber = 1;

        System.out.print("Fibonacci Series of "+maxNumber+" numb

        for (int i = 1; i <= maxNumber; ++i)
        {
            System.out.print(previousNumber+" ");
            /* On each iteration, we are assigning second number
            * to the first number and assigning the sum of last
            * numbers to the second number
            */

            int sum = previousNumber + nextNumber;
            previousNumber = nextNumber;
            nextNumber = sum;
        }
    }
}
```

GNU nano 4.8 Recursive.java

```java
// Source: https://www.guru99.com/fibonacci-series-java.html

// Using Recursion
public class FibonacciCalc{
    public static int fibonacciRecursion(int n){
        // Base Cases.
        if(n == 0){
            return 0;
        }
        if(n == 1 || n == 2){
            return 1;
        }

        // Add the previous two numbers.
        return fibonacciRecursion(n-2) + fibonacciRecursion(n-1);
    }

    public static void main(String args[]) {
    int maxNumber = 10;
    System.out.print("Fibonacci Series of "+maxNumber+" numbers: ");
    for(int i = 0; i < maxNumber; i++){
            System.out.print(fibonacciRecursion(i) +" ");
        }
    }
}
```

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?

The Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, …
Implemented iteratively (left) and recursively (right) [source].

```
 GNU nano 4.8                BinarySearch.java
import java.util.Random;

public class BinarySearch {

    // Codes for printing color in the console.
    // Not supported in every shell.
    public static final String ANSI_RESET = "\u001B[0m";
    public static final String ANSI_RED = "\u001B[31m";

    // Searches for value f in array A and returns true or false.
    // Implemented using iteration.
    public static Boolean search1(int[] a, int f) {
        return false;
    }

    // Searches for value f in array A and returns true or false.
    // Implemented using recursion.
    public static Boolean search2(int[] a, int f) {
        return false;
    }

    public static void main(String[] args) {
        final int CAPACITY = 50;
        int[] array;
        int value, target;
        Random r;
        Boolean found1, found2;

        // Create array and fill it with random sorted positive ints.
        // Each successive value is included with probability 50%.
        r = new Random();
        array = new int[CAPACITY];
        value = 0;
        for (int i = 0; i < CAPACITY; i++) {
            value += r.nextInt(2) + 1; // Increment by 1 or 2.
            array[i] = value;
        // Create a random target between 1 and array's max value.
        target = r.nextInt(value); // value is array's max value.

        // Print the array and highlight the target if present.
        // ANSI_RED and ANSI_RESET may not work in your terminal.
        for (int i = 0; i < CAPACITY; i++) {
            if (array[i] == target) {
                System.out.print(ANSI_RED + array[i] + ANSI_RESET);
            } else {
                System.out.print(array[i]);
            }
            System.out.print(" ");
        }
        System.out.println("");

        // Binary search for the target.
        // We should expect to find it 50% of the time.
        found1 = search1(array, target);
        found2 = search2(array, target);

        // Print out the results of the searches.
        System.out.printf("search1 for %d: %b%n", target, found1);
        System.out.printf("search2 for %d: %b%n", target, found2);
    }
}
```

```
-> java BinarySearch
2 4 5 6 8 9 11 13 14 15 16 18 20 21 22 24 25 27 28 30 32 34 35 36 37 38 39
40 41 43 44 46 48 50 52 53 55 57 58 59 60 61 63 64 66 68 69 71 73 75
search1 for 18: false
search2 for 18: false
->
```

Starter Code for `BinarySearch.java`.

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

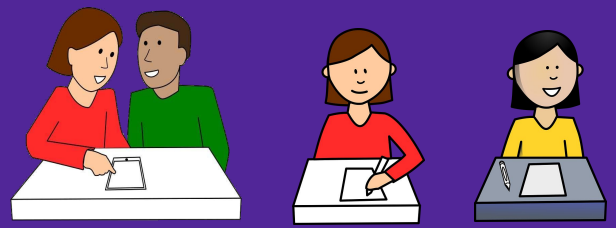
Note: Updated `~/.nanorc` will be posted after today's lecture.

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.



Discuss your ideas with a neighbor for 3 minutes.
Then you'll have time to try writing `search1`.

- 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.)


SPOILER
ALERT

```java
// 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;
}
```

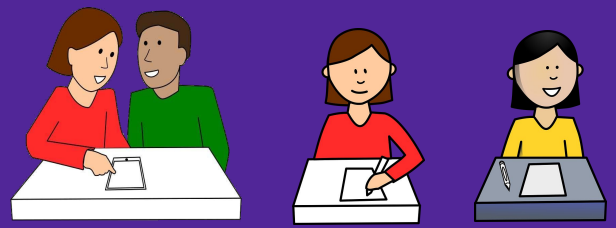
Finished `search1`.

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?


SPOILER
ALERT

```java
// 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`.