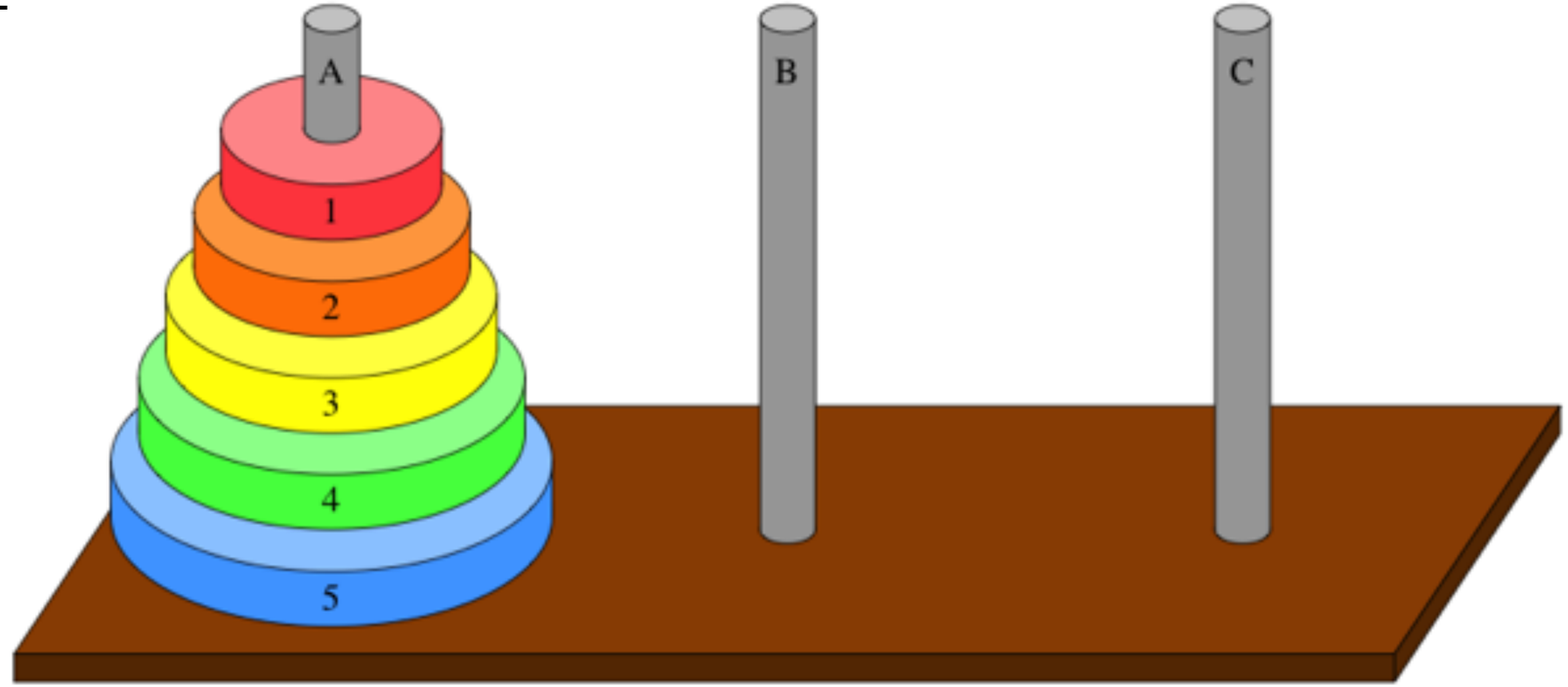
Recursion — Pt 2

cs206 — Oct 30

Towers of Hanoi



Extra Credit: +15 to homework grade

http://165.106.10.159/~gtowell/Towers/

Must be done by May 5

Must submit a believable time. +5 in-class final round.

Recursion

A method that calls itself, either directly or indirectly

Importantly, need a way to stop

Class Recurser

```
public void badRecurse(int c)
{
  System.out.println("A" + c);
  badRecurse(c-1);
}

public void goodRecurse(int c)
{
  System.out.println("B" + c);
  if (c<=0) return;
  goodRecurse(c-1);
}
```

Write a recursive function that given a number computes its integer base N log.
e.g.
baseNlog(2, 1100)==>10
baseNlog(10,1000) ==>3
baseNlog(10,9999) ==> 3
baseNlog(1,9)==>0
baseNlog(4,0) ==> 0

Recursion — return values

```
/**
 * A recursive function to add two positive numbers
 * @param num1 one of the numbers
 * @param num2 another number
 * @return the sum of the two numbers
 */
public int rAdder(int num1, int num2) {
    if (num2<=0)
        return num1;
    return rAdder(num1+1, num2-1);
}
public int rAdderB(int num1, int num2) {
    if (num2<=0)
        return 0;
    return 1+rAdderB(num1, num2-1);
}
```

base 2 log

```java
public int base2log(int num) {
    if (num == 1)
        return 0;
    return 1 + base2log(num / 2);
}
```

```java
public int base2logB(int num) {
    if (num <= 0)
        return -1;
    if (num == 1)
        return 0;
    return base2logBUtil(num, 1);
}

private int base2logBUtil(int num, int upward) {
    if (upward > num)
        return -1; // stops 1 too late, so return -1
    return 1+base2logBUtil(num, upward * 2);
}
```

Counter the number of occurrences of a letter in a string

```
public int numOccur1(char ch, String str) {
     if (str == null || str.equals("")) {
         return 0;
     }
     int count = 0;
     if (str.charAt(0) == ch) {
         count++;
     }
     numOccur1(ch, str.substring(1));
     return count;
 }
```

What does this return on “a” , “abc”
Why?

Occurrence count v2

```
int acount = 0;

 public int numOccur2(char ch, String str) {
     if (str == null || str.equals("")) {
         return 0;
     }
     if (str.charAt(0) == ch) {
         acount++;
     }
     numOccur2(ch, str.substring(1));
     return acount;
 }
```

Correct answer, but a BAD solution

Occurrence count v3 and v4

```
public int numOccur3(char ch, String str) {
    if (str == null || str.equals("")) { return 0; }
    int count = 0;
    if (str.charAt(0) == ch) { count = 1; }
    return count + numOccur3(ch, str.substring(1));
}


public int numOccur4(char ch, String str) {
    return numOccur4Util(ch, str, 0);
}
private int numOccur4Util(char ch, String str, int count) {
    if (str == null || str.equals("")) { return count; }
    if (str.charAt(0) == ch) { count++; }
    return numOccur4Util(ch, str.substring(1), count);
}
```

v5 and v6

```
public int numOccur5(char ch, String str) {
        if (str == null || str.length()==0)
            return 0;
        return numOccur5Util(ch, str, 0, 0);
    }
    private int numOccur5Util(char ch, String str, int loc, int count) {
        if (loc >= str.length())
            return count;
        if (str.charAt(loc) == ch) { count++; }
        return numOccur5Util(ch, str, loc+1, count);
    }

public int numOccur6(char ch, String str) {
        if (str == null || str.length()==0)
            return 0;
        return numOccur6Util(ch, str, 0);
    }
    private int numOccur6Util(char ch, String str, int loc) {
        if (loc >= str.length())
            return 0;
        int cc = 0;
        if (str.charAt(loc) == ch) { cc=1; }
        return cc+numOccur6Util(ch, str, loc+1);
    }
```

recursion practice

```java
/**
 * Compute the sum of the components of the array
 */
public int addArray(int[] array);
```

```java
/**
  * Count the number of odd numbers in the
  * provided array
 **/
public int numOdd(ArrayList<Integer> intArrLis)
```

more returning values

```
public ArrayList<Integer> rAccmulate(int count)
 {
     if (count <= 0)
         return new ArrayList<Integer>();
     ArrayList<Integer> alAcc = rAccmulate(count - 1);
     alAcc.add(count);
     return alAcc;
 }
```

```
public ArrayList<Integer> rAccmulateB(int count) {
    ArrayList<Integer> ret = new ArrayList<>(count);
    rAccumulateUtil(count, ret);
    return ret;
}

private void rAccumulateUtil(int count, ArrayList<Integer> arrLis) {
    if (count <= 0)
        return;
    arrLis.add(count);
    rAccumulateUtil(count - 1, arrLis);
}
```

```
public static void main(String[] args) {
    System.out.println("AA " + (new AB()).rAccmulate(5));
    System.out.println("BB " + (new AB()).rAccmulateB(5));

}
```

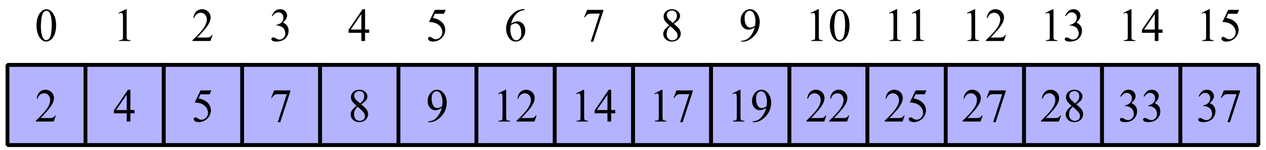
What is the output?

Finding a data item

- Suppose you have an array (or ArrayList) of N items. How do you determine if the array contains a particular item?
  - Does the form of the array matter?
    - Unsorted
    - Sorted
  - What is the complexity of finding an item?

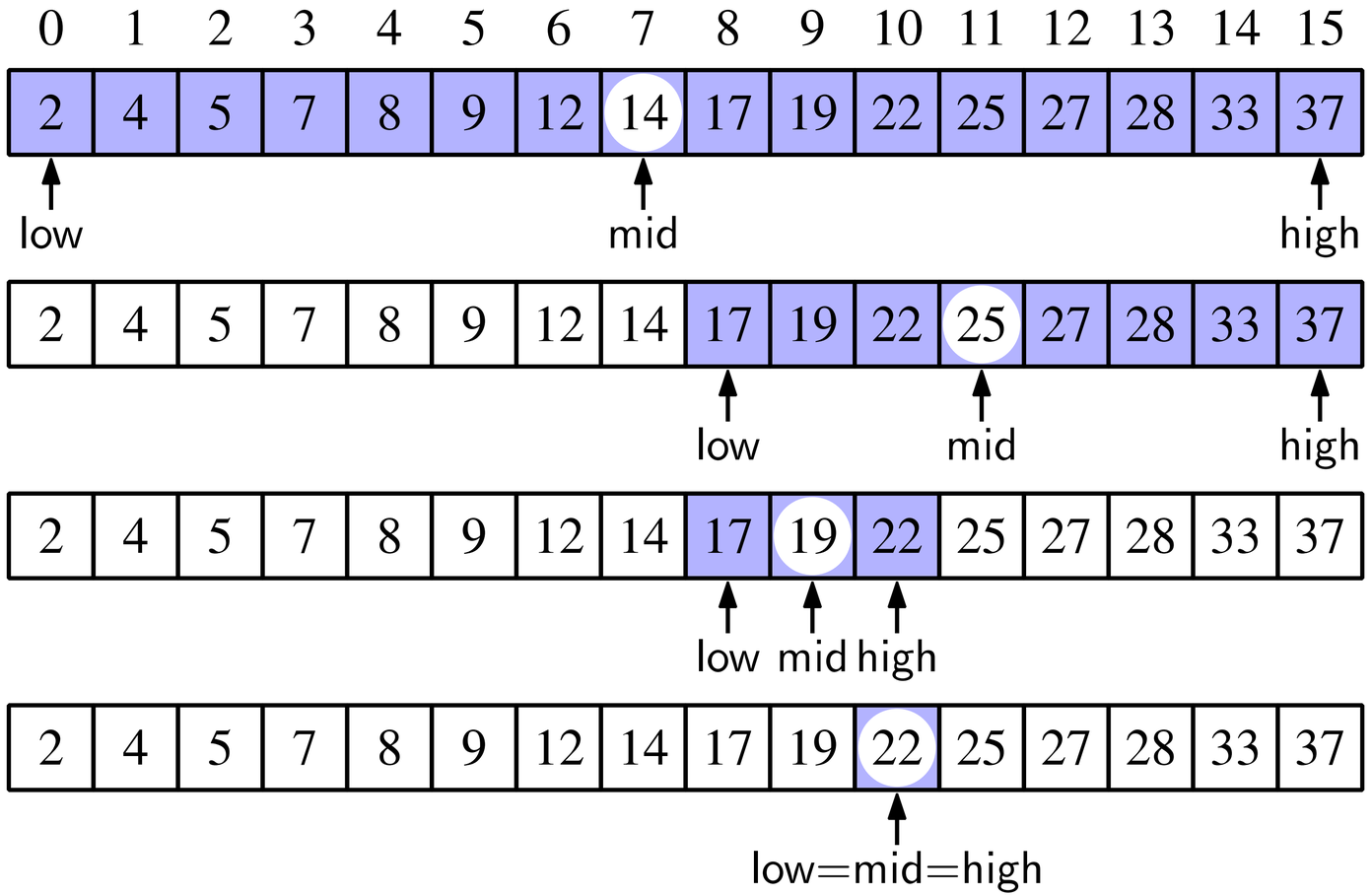
Binary Search

- Search for an integer (22) in an ordered list



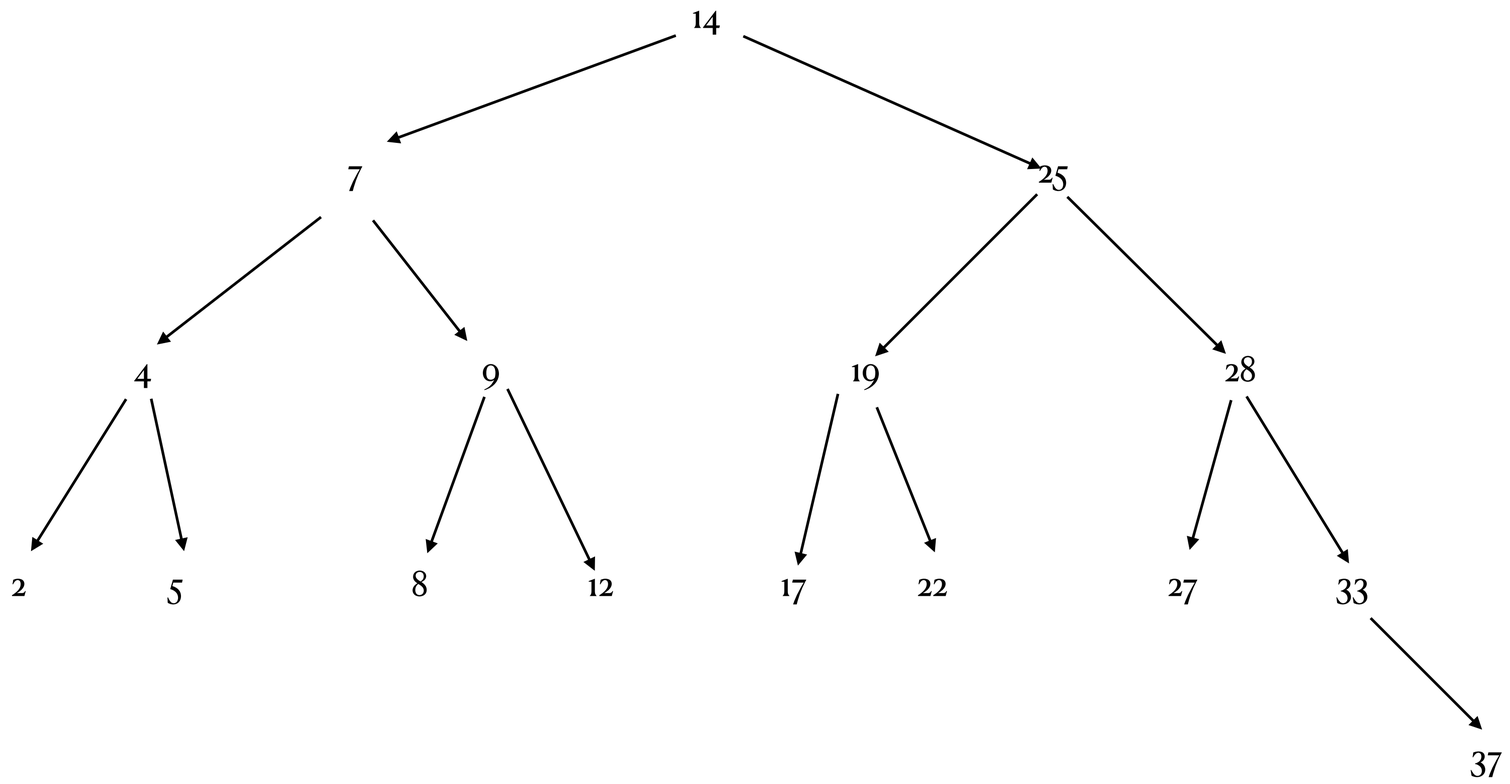
- $mid = \left\lfloor \frac{low + high}{2} \right\rfloor = \left\lfloor \frac{0 + 15}{2} \right\rfloor = 7$
  - `target==data[mid]`, found
  - `target>data[mid]`, recur on second half
  - `target<data[mid]`, recur on first half

target = 22


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low mid high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low=mid=high

View the data as a binary tree

“Binary Search Tree”



Binary Search Code

```
/**
* The public facing call to array search
* The array to be searched is a private instance variable
* @param target the value being searched for
* @return true if the value is in known, false otherwise
*/
public boolean contains(int target) {
    if (data==null)
        return 0;
    return searchUtil(target, 0, data.length-1, 0);
}
```

Suppose change the storer of the data to ArrayList?

Binary Search Code

```
    /**
     * Binary search, recursively on sorted internal array of ints
     * @param target the item to be found
     * @param lo the bottom of the range being searched
     * @param hi the top of the range being searched
     * @param steps the number of steps the search has taken
     * @return true if the target was found
     */
private boolean searchUtil(int target, int lo, int hi, int steps) {
    if (lo>hi) return false;
    int mid = (lo+hi)/2;
    System.out.println(target + " " + data[mid] + " " + lo + " " + hi + " " + steps
    if (data[mid]==target) return true;
    if (data[mid]<target)
        return searchUtil(target, mid+1, hi, steps+1);
    else
        return searchUtil(target, lo, mid-1, steps+1);
     }
```

Binary Search Analysis

- Each recursive call divides the array in half
- If the array is of size n, it divides (and searches) at most $\log_2 n$ times before the current half is of size 1
- $O(\log_2 n)$

LAB

- For a binary search over an array containing the numbers 1..20, what is the sequence of recursive function calls to find 11
  - show all of the arguments to each recursive call

Recursion and Backtracking

- All problems considered so far progress steadily towards an answer.
- Consider a maze. Sometimes you need to “backtrack”.
- Idea:
  - somehow make a copy of where you are,
  - try going forward using your copy,
  - If that fails back up and go some other direction using your original

- Alternately
  - when backing up, undo your change
- Twiddle
  - especially with mazes mark places you have been so you do not retry failed paths

Example: Word Reduction

- Problem: given an english word can you remove one letter and still have an english word.
  - Can you do this repeatedly until only a 2 letter word remains?
  - Consider the word “cored” .. core, ore, or!!!
  - Lets suppose:
    - boolean isInEnglish(String s)
      - return true iff s is an English word
    - String removeNchar(int n, String s)
      - removes the nth character of the string. So removeNchar(0, “dour”) is “our”

Word Reducer

```
public Worder() {
        words = new HashMap<>();
        // this file exists on Mac and Linux.  It does not exist on Windows
        try (BufferedReader br = new BufferedReader(new FileReader("/usr/share/dict/words"))) {
            String l;
            while (null != (l=br.readLine())) {
                words.put(l.trim(), 1);
            }
        }
        catch (Exception ee) {
            ee.printStackTrace();
        }
    }

    public boolean isEnglish(String s) {
        return words.containsKey(s);
    }

    String removeNchar(int n, String s) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < s.length(); i++) {
            if (i != n) {
                sb.append(s.charAt(i));
            }
        }
        return sb.toString();
    }
```

Base Cases for Word reducer

WordReducer

recursion

- base case:
  - if word length <=2 return isEnglish()
- base case 2
  - if not english return false;
- loop
  - make a copy of the string with a letter removed
  - recur on the copy

```
public boolean reducable(String s) {
        System.out.println(s);
        if (s.length()<=2) { return isEnglish(s); }
        if (!isEnglish(s)) return false;
        for (int i=0; i<s.length(); i++) {
            if (reducable(removeNchar(i, s)))
                return true;
        }
        return false;
    }
```

N Queens problem

https://sites.fas.harvard.edu/~cscie119/lectures/recursion.pdf

- Place N queens on an NxN chessboard such that no queen can take another
- Strategy:
  - on row N
    - move across columns trying a spot for OK
      - if OK, then recur with N+1
    - if have checked everything in a column and there is no place that is OK
      - backtrack
        - undo placement of queen in row N-1 and continue across that row

N Queens

setup

- board just a 2d array of chars
- will do recursion with a private utility function

```
public class NQueens {
    private char[][] board;
    private int size = 0;

    public NQueens(int siz) {
        size = siz;
        board = new char[siz][siz];
        for (int i = 0; i < siz; i++) {
            for (int j = 0; j < siz; j++) {
                board[i][j] = '.';
            }
        }
    }

    private void showBoard() {
        for (int r = 0; r < size; r++) {
            for (int c = 0; c < size; c++) {
                System.out.print(board[r][c]);
            }
            System.out.print("\n");
        }
    }


    public void doQueens() {
        doQueensUtil(0);
    }
```

N Queens

recursion

- base case:
  - the row being asked to consider is off board
    - return true;
- in the row
  - go across every column
    - put queen in a column
      - check if that is OK
        - if it is, go to recur to next row
        - if found solution return true;
      - if NOT OK, remove queen from column
  - if cannot find a place to put a queen, return false

```
private boolean doQueensUtil(int roww) {
    if (roww >= size)
        return true;
    for (int col = 0; col < size; col++) {
        board[roww][col] = 'Q';
        if (OKBoard()) {
            boolean v = doQueensUtil(roww + 1);
            if (v)
                return true;
        }
        board[roww][col] = '-';
    }
    return false;
}
```