Recursion — Pt 2

cs206 — Oct 30

Back to Heaps

Remove Top

- Assume a min heap
- Let parent.loc = top of heap
- Repeat
  - If parent has no children STOP
  - Find the smallest child of the parent. Call it “bestchild”
  - If bestchild.value < parent.value
    - swap bestchild.value and parent.value
    - set parent.loc = bestchild.loc
  - ELSE STOP

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

```java
/**
 * 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);
}
```

Recursion — return values

```java
/**
 * Implement multiplication recursively using addition
 * For example, given the args 7 and 4 write a recusive function
 * that computes 7+7+7+7
 * @param i1 a number
 * @param i2 another number
 * @return i1*i2
 */
public int multiply(int i1, int i2) {

}
```

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 — returning values & private recursive functions

```
    private BigInteger fibonacciUtil(BigInteger fibNumA, BigInteger fibNumB, int
counter)
    {
        System.out.println(counter + " " + fibNumA + " " + fibNumB);
        if (counter==1)
            return fibNumA.add(fibNumB);
        return iFibonacci(fibNumB, fibNumA.add(fibNumB), counter-1);
    }

    public BigInteger fibonacci(int n) {
        if (n<=0) // make sure that the number being asked for is reasonable
            return BigInteger.valueOf(0);
        if (n<3)
            return BigInteger.valueOf(1);
        return iFibonacci(BigInteger.valueOf(1), BigInteger.valueOf(1), n-2);
    }
```

recursion practice

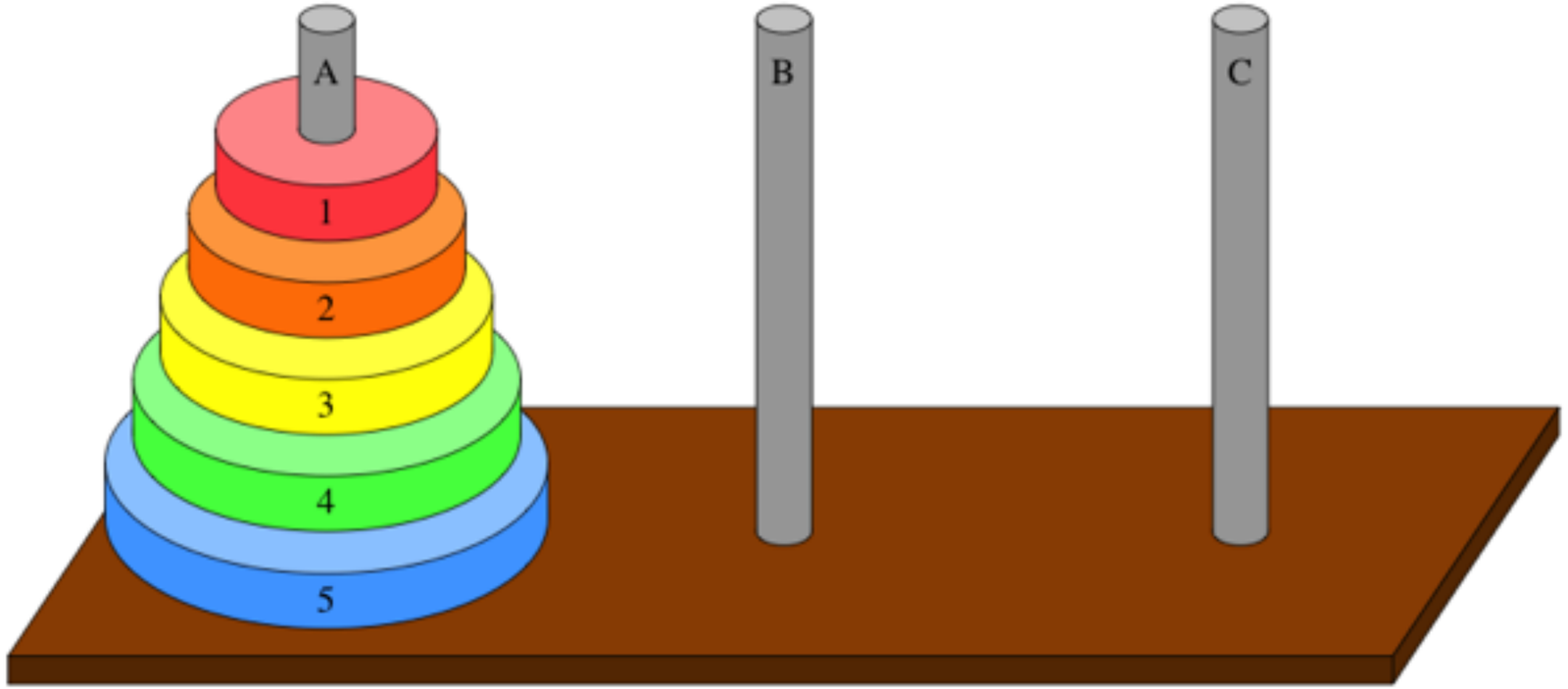
```java
/**
 * Write a recursive function to add all the values in the array
 * Hint, this method should not be recursive.  Rather make a
 * private recursive function and call that from here
 * @param array
 * @return the sum of the numbers in the array
 */
public int addArray(int[] array);
```

```java
/**
 * Return true iff the string is a palindrome.
 * @param s -- the string to be checked
 * @return true iff the provided string is a palindrome
 */
public boolean palindrome(String s);
```

more returning values

```java
/**
     * Build up an ArrayList containing the numbers 1..count
     * This code does not handle negative numbers, at all.
     * @param count the max number in the returned ArrayList
     * @return an ArrayList containing the numbers 1..count
     */
    public ArrayList<Integer> rAccmulate(int count)
    {
        if (count <= 0)
            return new ArrayList<Integer>();
        ArrayList<Integer> alAcc = rAccmulate(count-1);
        alAcc.add(count);
        return alAcc;
    }
```

Towers of Hanoi



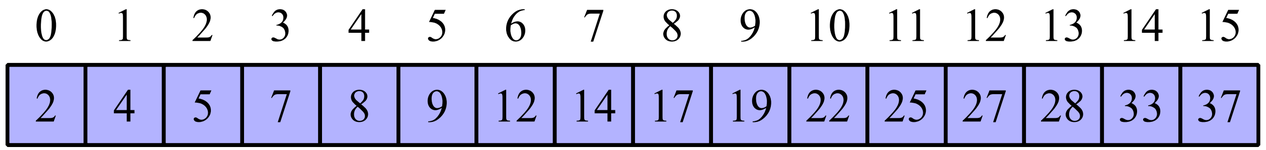
Complexity Analysis: $O(2^n)$!!!!

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
    - Heap
  - What is the complexity of finding an item?

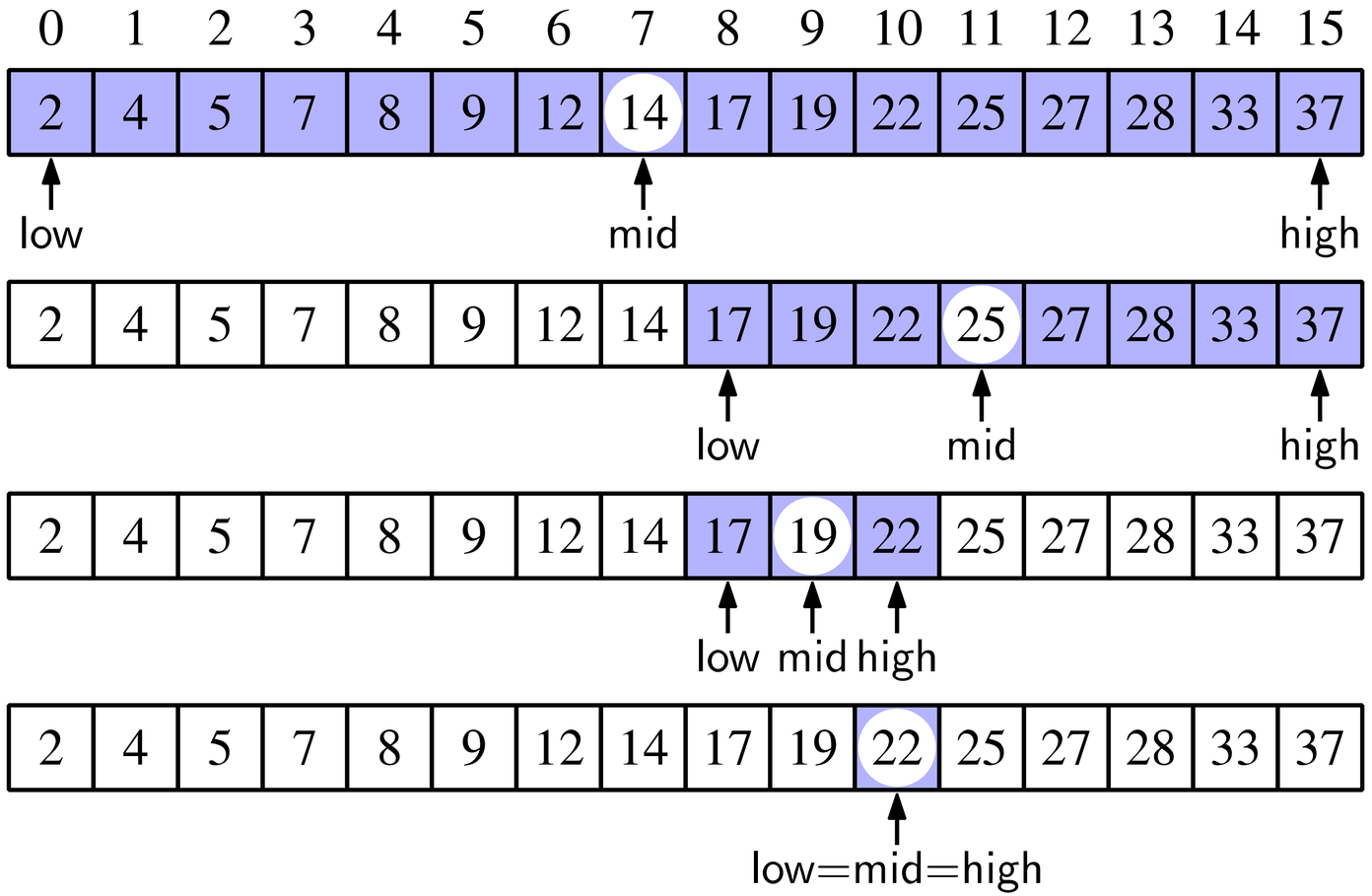
Binary Search

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



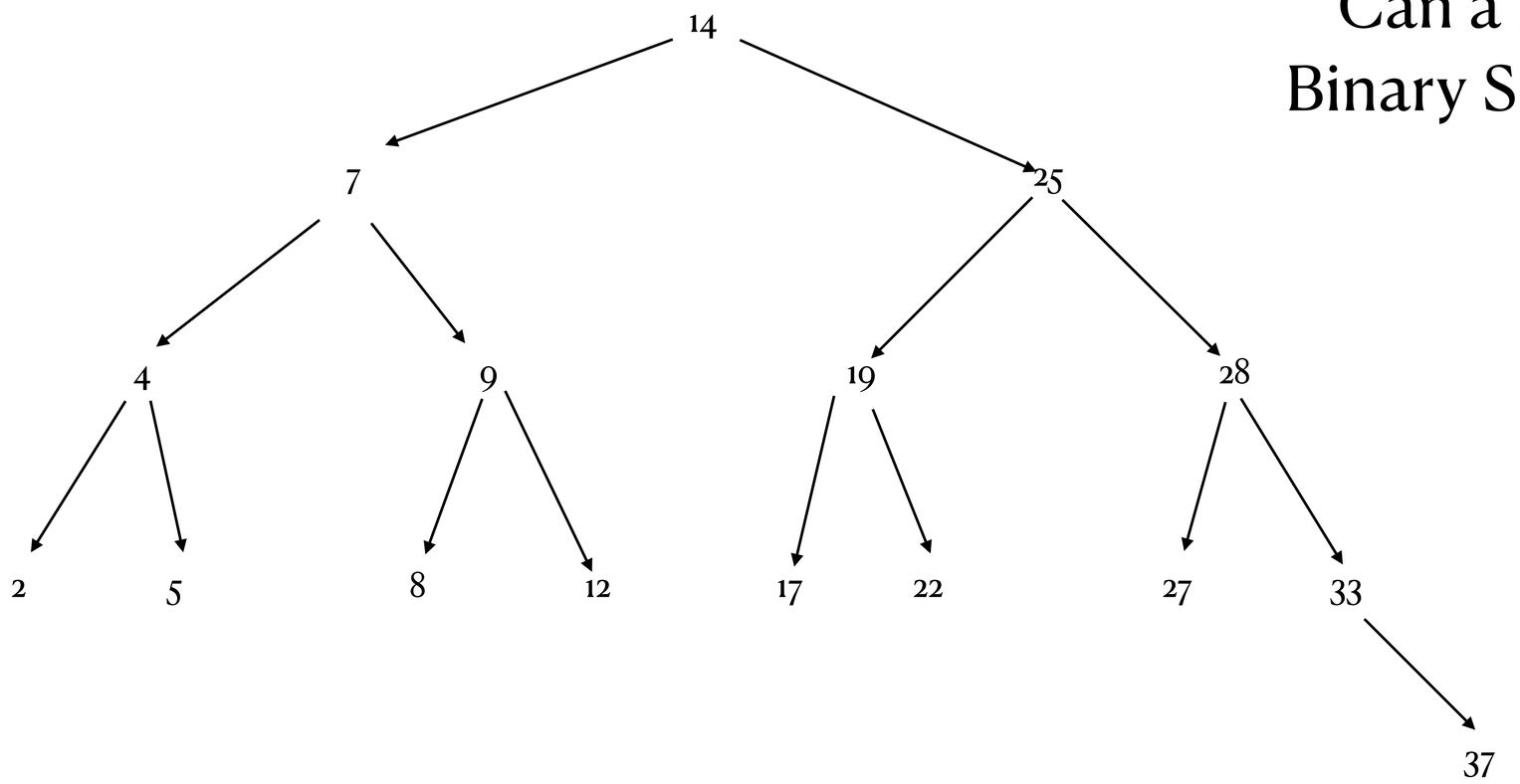
- $mid = \left\lfloor \frac{low + high}{2} \right\rfloor = \left\lfloor \frac{0 + 15}{2} \right\rfloor = 7$
  - `target==data[mid]`, found
  - `target>data[mid]`, recur on second half
  - `target<data[mid]`, recur on first half

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”



Is this a heap?
Can a heap be a
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 iSearch(target, 0, data.length-1, 0);
}
```

Suppose change instance variable 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 iSearch(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 iSearch(target, mid+1, hi, steps+1);
    else
        return iSearch(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
- $O(\log_2 n)$

Reimplement Binary search with iteration

What parameters does the iterative method need?
Does a separate private method even make sense?

Backtracking with Recursion

- Previous examples all progressed linearly to success/failure
- So consider doing binary like search on an unsorted array
  - Need to backtrack and try other directions on failure.
  - Backtracking is when recursion really shines

Backtracker

```
/** Binary-like search, but will work on sorted or unsorted lists
  * because it can do backtracking.
  */
private boolean iSearch(int target, int lo, int hi, int depth)
     {
    if (lo>hi) { return false; }
    int mid = (lo+hi)/2;
    System.out.println(" " + target + " " + data[mid] + " " + lo + "
" + depth);
    if (data[mid]==target) return true;
    if (iSearch(target, mid+1, hi, depth+1))
       return true;
    return iSearch(target, lo, mid-1, depth+1);
}
```