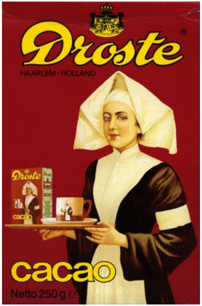
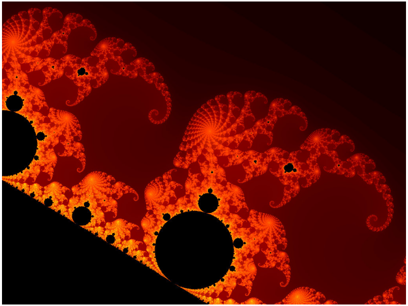
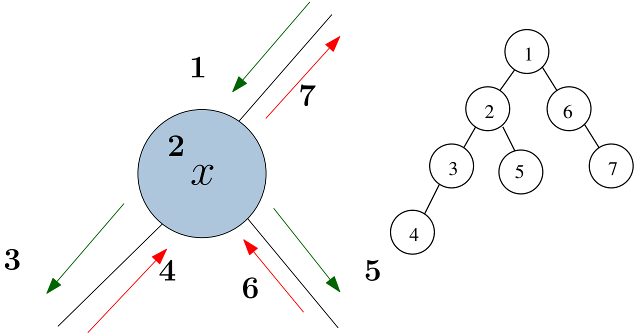
Recursion

Daniel Archambault





Previously in CS-115



Gotta see the trees through the forest!

Previously in CS-115

- What is a binary tree?

Previously in CS-115

- What is a binary tree?
- What are the attributes of a node in a binary tree?

Previously in CS-115

- What is a binary tree?
- What are the attributes of a node in a binary tree?
- What is the difference between in order, pre order, and post order traversals?

Previously in CS-115

- What is a binary tree?
- What are the attributes of a node in a binary tree?
- What is the difference between in order, pre order, and post order traversals?
- What is a binary search tree?

Previously in CS-115

- What is a binary tree?
- What are the attributes of a node in a binary tree?
- What is the difference between in order, pre order, and post order traversals?
- What is a binary search tree?
- What is so special about an in order traversal of a binary search tree?

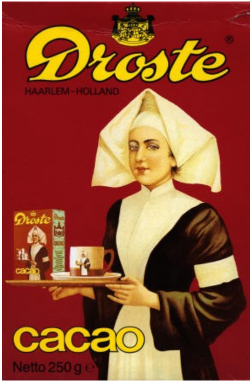
Previously in CS-115

- To understand recursion... you need to understand recursion - 1!

Recursion

What is Recursion?

- "A method for defining functions in which the function being defined is applied within its own definition."
  - –Wikipedia
- In many ways, it is an alternative to iterative solutions
  - Solutions involving loops



Motivation

- Why learn recursion?

Motivation

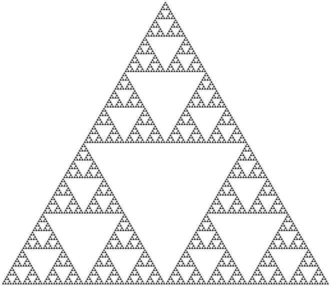
- Why learn recursion?
  - Because it's fun. :)

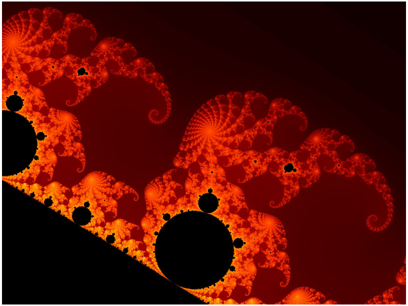
Motivation

- Why learn recursion?
  - Because it's fun. :)
  - Many problems exist in computer science where:
    - a recursive solution is easier to implement and more efficient
  - You have already seen some recursive algorithms!
    - quicksort is recursive
    - binary search is recursive (review today)

Fractals

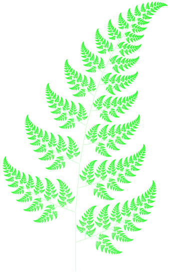
- Recursion can be used to draw nice mathematical pictures
  - Both Serpinski Gasket and Mandelbrot Set are from Wikipedia

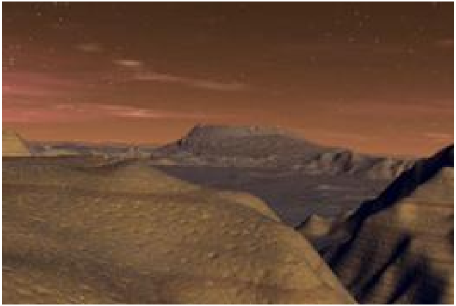




Modelling Plants and Terrain

- In computer graphics, plants and terrain can be modelled recursively
- L-systems have a natural recursive definition
  - Brandy's Fern from Wikipedia
- Fractals can be used to enhance realism of terrain
  - SimPlanet Project's Fractal Mars





Haskell

- Some languages don't even have iteration!
  - Believe that better programming style does not involve assignment to variables
  - Assignment required for loops

```
quicksort [] = []
quicksort (s:xs) =
quicksort [x|x <- xs,x < s] ++ [s] ++
quicksort [x|x <- xs,x >= s]
```

We kinda have already seen recursion...

- In our data structures, we have seen recursion a bit
  - A link list consists of the current link and a link to the next link
  - A binary tree consists of the current node and two links to its children which are also nodes
- You can also do this with function calls

Phone Book Example

Iterative Solution

```
PhoneNumber itSearch (PhoneBookEntry book[] , String name)
{
   for (int i = 0; i < book.length(); i++)
   {
      if (book[i].getName ().equals (name))
         return book[i].getPhoneNumber ();
   }
}
```

Recursive Solution

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Elements of a Recursive Program

- Base Case
- Divide Input into Parts
- Recursive Case
- Synthesis and Return

Base Case

- Case where solution is known or easy to compute
- Tells recursive program where to stop
- Can be several

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Divide Input into Parts

- Problem is divided into one or more smaller instances
- Parts must “head towards” base case. Otherwise, infinite recursion!

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Recursive Case

- Algorithm is reapplied to the smaller instances

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Synthesise and Return

- Synthesise solution from result(s) of recursive call(s)
  - In the binary search, we just return the name
  - In many other recursive algorithms, not always the case.

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Tracing Binary Search

- We are nearly ready to trace this code

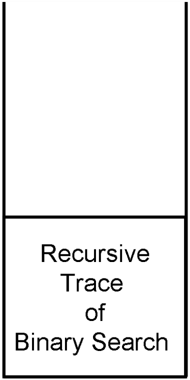
```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Tracing Binary Search

- We are nearly ready to trace this code
  - But first... Let's push this on the stack.

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

The Call Stack



callStack.push (Trace Binary Search)

Recursion?

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{

    return recSearch (book, name, start, middle - 1);
```

Recursion?

- Is actually....

```
 PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   callStack.push (local variables)
   int middle = (2 + 6)/2;
   ....
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
      callStack.pop ()
```

Recursion?

- Is actually....
- True for any function call in Java (and most languages)

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   callStack.push (local variables)
   int middle = (2 + 6)/2;
   ....
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
      callStack.pop ()
```

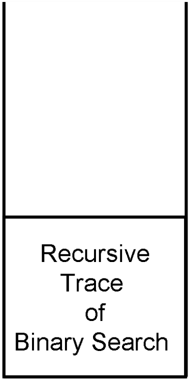
Recursion?

- Is actually....
- True for any function call in Java (and most languages)

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   callStack.push (local variables)
   int middle = (2 + 6)/2;
   ....
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
      callStack.pop ()
```

- Can view (and usually trace) recursion with stacks.

The Call Stack



Lecture = callStack.peek (); callStack.pop ();

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
 PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
 PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Trace Binary Search

- Suppose we have sorted list on the board and looking for Raul

```
PhoneNumber recSearch (PhoneBookEntry book[], String name,
int start, int end)
{
   int middle = (start + end)/2;
   if (book[middle].getName ().equals (name))
      return book[middle].getPhoneNumber ();
   if (name.compareTo (book[middle].getName()) < 0)
      return recSearch (book, name, start, middle - 1);
   else
      return recSearch (book, name, middle + 1, end);
}
```

Problem 1

- Consider the following program

```
int factorial (int n)
{
   if (n <= 1)
      return 1;
   return n*factorial (n - 1);
}
```

- Identify the recursive program elements
- Trace the execution of factorial (5);
  - Follow along by drawing the stack