Complexity of Search

- **Iteration to recursion**
- **Asymptotic complexity**
  - **What is it?**
  - **What is big O notation?**
  - **Justifying average case analysis**
- **Envalope classes**
  - **Use in i/o**

Where does the iteration go?

```
private static int binSearch(int low, int hi, int []
   a, int desired){
      if (hi == low +1) {return -1;}
      int mid = (hi+low)/2;
      if (desired == a[mid])  return mid;
      else if (desired < a[mid]) {
            return (binSearch(low, mid, a,desired));
            }
      else return(binSearch(mid, hi, a, desired));
```

Each time through the loop we halved the interval to be examined. Each time we call binSearch recursively, we half the interval.

Number of copies of binSearch needed == number of iterations.

Recursion in “Real” Life

- **Trivia question: What famous Dr. Seuss story has an example of recursion in it?**
- **Answer next week!**

Asymptotic Complexity

- **Trying to calculate how an algorithm behaves for large amounts of data**
  - **n or 2n comparisons versus n^2**
- **For 250 million people in US (2.5E8),**
  - **n is 2.5 E8; 2n is 5. E8; n^2 is 6.25 E16**
- **Clearly, for large n, 2n and n are *close* in value whereas n^2 is much larger!**

Asymptotic Complexity

- **Another comparison**

n versus $\log_2 n$ (use $\log_e n$ to approx)

log n grows much more slowly than n.

n	$\log_e n$
1.0	0.0
1.5	.41
2.0	.69
5.0	1.61
8.0	2.08
20	3.00
50	3.91
80	4.38
100	4.61

Asymptotic Complexity

- **Talk about how cost of an algorithm increases as problem size increases**
- **Try to find a function of problem size such that worst case behavior is bounded above by that function**
  - **O(j) (read this as big-O of j)**
  - **Means algorithm's performance in worst case is bounded above by j, a measure linear in the problem size (e.g., number of numbers to search).**
  - **Linear search is O(n); binary search is O(log n)**
  - **Constant time is O(1)**

Revisiting Linear Search Average Cost Analysis

- **Assume array holds j elements**
- **Assume about half the lookups fail (on average)**
- **Consider doing 2j lookups**
  - **j lookups find nothing and each costs j**
  - **j lookups find a match and each costs about j/2**
  - **Total cost of 2j lookups is:**

    $$j * j + j * (j/2) = 1.5\ j^2$$

  - **Expected cost for any one search is total cost / # searches = $1.5\ j^2 / (\ 2j\ ) = .75\ j$**

see next page

Validity of our Assumptions

- **Assume desired value is in the array of size j**
  - **Any position in array is equally likely to hold the value**
- **What's expected cost for a lookup that matches?**
  - **Find total cost of looking up each element**

    $1 + 2 + 3 + ... + (j-2) + (j-1) + j = ((j+1) * j) / 2$
  - **Number of lookups is j**
  - **Average cost:$((j +1)*(j/2))/j$ = $(j + 1)/ 2$ and $j/2$ is close enough to this value for large j**

Search Algorithm Complexities

Assume an array with n values.

	Linear Unordered	Linear Ordered	Binary
Best	O(1)	O(1)	O(1)
Worst	O(n)	O(n)	O(log n)
Average	O(3n/4)	O(n/2)	O(log n)

Envelope Classes

- **Needed because everything in Java is actually an object**
- **To get the primitive types into the language we need a some mechanism to obtain those kinds of values**
- **Envelope classes: *Integer, Double, Character, Boolean***
- **Methods in these envelope classes let us move between classes and primitive types**

Integer Class

- **Interface (partial)**

  ***Integer (int value)*; //creates an Integer object**

  ***int IntValue()*;//obtains int value from Integer receiver**

  ***Integer valueOf(String s)*;//class method which converts a String object to an Integer object**

```
Integer Iobj = new Integer (i);
System.out.println(Iobj.intValue());
```

What is this used for?

- **Input in standard Java**
  - **Input is a stream of substring tokens, separated by blanks, commas, or tabs**
  - **Can pass each token to the appropriate envelope class to convert it to an object of the correct type**
  - **Then convert to corresponding primitive value**
- **Have also seen class variables from Double**
  - **Double.POSITIVE_INFINITY**
  - **Double.NEGAIVE_INFINITY**

TokenStream Class

- **cs111.io package contains TokenStream class which uses StringTokenizer**
- ***TokenStream()* throws IOException**
  - **For keyboard input uses *InputStreamReader***
- ***TokenStream(String filename)* throws IOException**
  - **For file input uses *FileReader***
- **Allows for multiple input streams in use at at same time by creating multiple TokenStream objects**

StringTokenizer Class

- **Standard Java StringTokenizer class provides methods for reading substrings:**
  - ***StringTokenizer (String s)*;//constructor**
  - ***String nextToken()*;//returns next substring from StringTokenizer reciever**
  - ***boolean hasMoreTokens()*;//checks if StringTokenizer receiver has more tokens**

TokenStream Class Essentials

- **Similar to JavaGently Text Class**

```
public class TokenStream{
  private StringTokenizer t = null;
  private BufferedReader br = null;
  private String currentToken = "";
  private boolean keyboard = true;//reset to false if
      //file io is used
...
//2 forms of each read method, one for keyboard
//one for files (which don't use a prompt)
public int readInt() throws IOException();
public int readInt(String prompt) throws IOException();
//similarly for readDouble(), readString(), readChar()
```

```
/** reads a new line and establishes a tokenizer.
 * If reading from keyboard, prompt on each new line
 * @param prompt string used to prompt for input
 */
  private void refresh(String prompt) throws
  IOException {
    while ((t == null) || !(t.hasMoreTokens())) {
      if (keyboard) {
        System.out.print(prompt);
        System.out.flush();
      }
      String line = br.readLine();
      if (line == null) throw new EOFException();
      t = new StringTokenizer(line);
    } }
```

Java i/o package

StringTokenizer class

```
/**
 * reads an integer from the TokenStream
 * @param prompt string used to prompt for input
 * @return the next integer in the TokenStream
 */
  public int readInt(String prompt) throws
  IOException {
    while (true) {
      refresh(prompt);
      String item = nextToken();
      try {return
  (Integer.valueOf(item.trim())).intValue();}
        catch (NumberFormatException e) {
        System.out.println(item + " is an invalid ”
            “integer, try again.");
        System.out.flush();
        }
    }
  }
```

String method

Integer method

class method, class Integer