Review

- mousePressed()
- mouseReleased()
- mouseClicked()
- mouseMoved()
- mouseDragged()
- keyPressed()
- keyReleased()
- keyTyped()
- mouseButton
- mousePressed
- keyCode
- key

- Models of Interactivity

- OOP
- encapsulation
- inheritance

Hints for Assignment 7 – Part 2

- loadStrings() loads lines from a file and returns a String array
  - one for each line in the file
- split() takes a delimited String and returns a String array
  - one for each delimited substring

Must convert Strings that look like numbers to actual numbers before they can be used as numbers.

```
println( "1.2" + 3 );          →  ?

println( float("1.2") + 3 );   →  ?
```

Hints for Assignment 7 – Part 2

- RTF files are not plain text files
  - *RTF = Rich Text Formatted*

- Try…
  - Word/Writer | Save As | Plain Text (*.txt)
  - Excel/Calc | Save As | CSV (Comma Delimited) (*.csv)

- Wordpad vs. Notepad

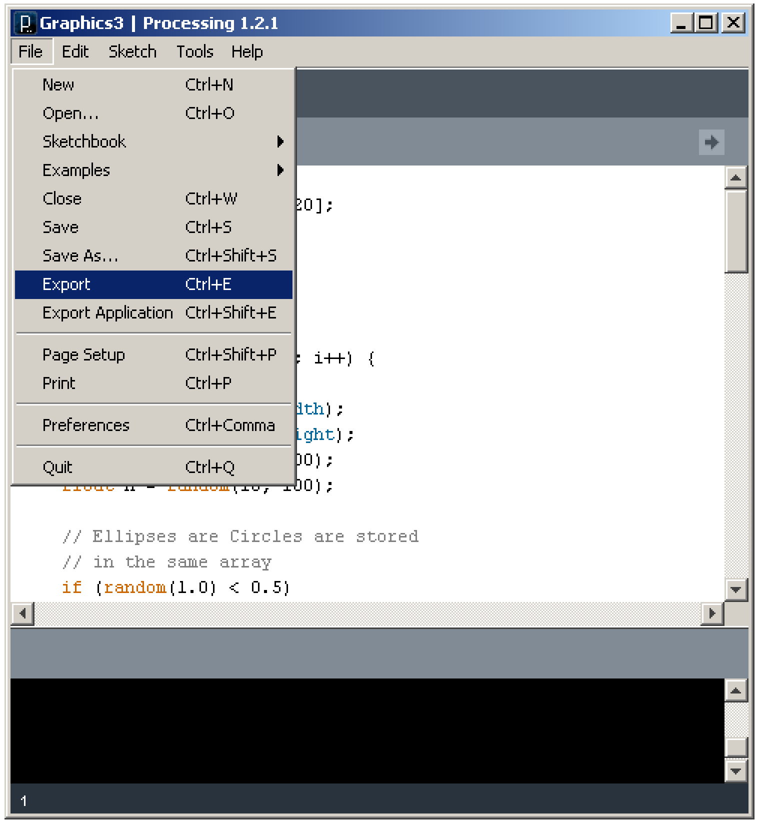
Processing == Java + Extras

Processing Adds:

- A handful of simplified functions
- Drawing functions
- Text and font manipulations
- Image and video
- 3D transformations
- An editor
- …

Best way to see what Processing becomes is to generate a Java program and compare

- File | Export
- Look in generated applet folder



```
20];
; i++) {
dth);
ight);
00);
float h = random(10, 100);

  // Ellipses are Circles are stored
  // in the same array
  if (random(1.0) < 0.5)
```

WinMerge

What's different?

- Import of core Processing libraries
- Import of numerous Java libraries
- Encapsulation of code in subclass of PApplet
- Explicit visibility statements : `public, private …`
- Number format changes

  `if (random(1.0) < 0.5) -> if (random(1.0`**`f`**`) < 0.5`**`f`**`)`
- color data types are converted to int
- Some functions are changed

  `numbers =` **`int`**`(snumbers); ->`

  `numbers =` **`PApplet.parseInt`**`(snumbers);`
- Added `public static void main() {…`

The smallest program in Processing

```
println("Hello World!");
```

The equivalent program in Java.

```
class HelloWorldApp {
    public static void main(String[] args) {
        // Display the string.
        System.out.println("Hello World!");
    }
}
```

All Java programs start at the special function …

```
public static void main() { …
```

This must be added to the generated Java program

PApplet

- The top-level class in Processing
- The class in which all your Processing code goes
- Implements most of Processing's core behavior
- Way down the Java hierarchy

```
java.lang.Object
    ↘ java.awt.Component
        ↘ java.awt.Container
            ↘ java.awt.Panel
                ↘ java.applet.Applet
                    ↘ processing.core.PApplet
```

http://processing.googlecode.com/svn/trunk/processing/build/javadoc/core/processing/core/PApplet.html

Nearly all of Java is available from Processing

http://download.oracle.com/javase/7/docs/api/

Exam 2 Review

Algorithms Review

- A well-defined set of instructions for solving a particular kind of problem.

1. Exhaustive (Linear) Search
2. Binary Search
3. Selection Sort
4. Bubble Sort

Selection Sort

1. Scan a list top to bottom and find the value that should come first.
   a) Set top item as tentative item to swap
   b) Scan and compare items lower on list to tentative swap
   c) If "better" item identified, reset it to tentative swap
2. When reaching bottom, swap that item with item at the top position.
3. Repeat scan starting at next lowest item in the list.
4. Stop swapping when reach bottom of list.

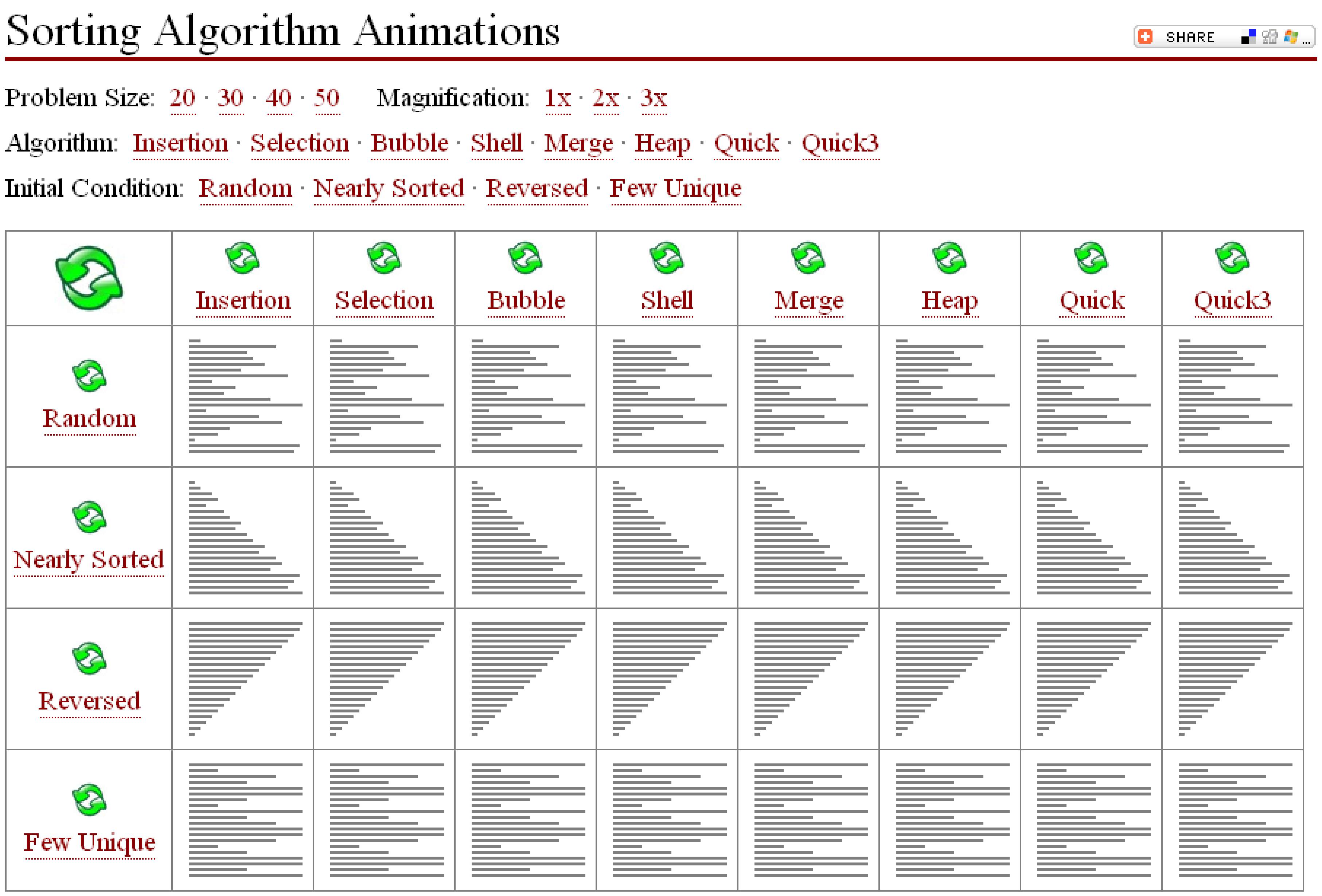
– Works best when swapping is expensive.

Bubble Sort

1. Scan through a list from bottom to top.
2. Compare adjacent pairs of items.
3. If two items are out of order, swap them.
4. After one complete scan, the top item is in place (bubbles to top). Skip that item on subsequent scans.
5. Repeat scan until no changes are made (completely ordered).

– Works best when there are few items out of order.

Bubble-sort with Hungarian ("Csángó") folk dance
http://www.youtube.com/watch?v=lyZQPjUT5B4



Sorting Algorithm Animations

Problem Size: 20 · 30 · 40 · 50   Magnification: 1x · 2x · 3x

Algorithm: Insertion · Selection · Bubble · Shell · Merge · Heap · Quick · Quick3

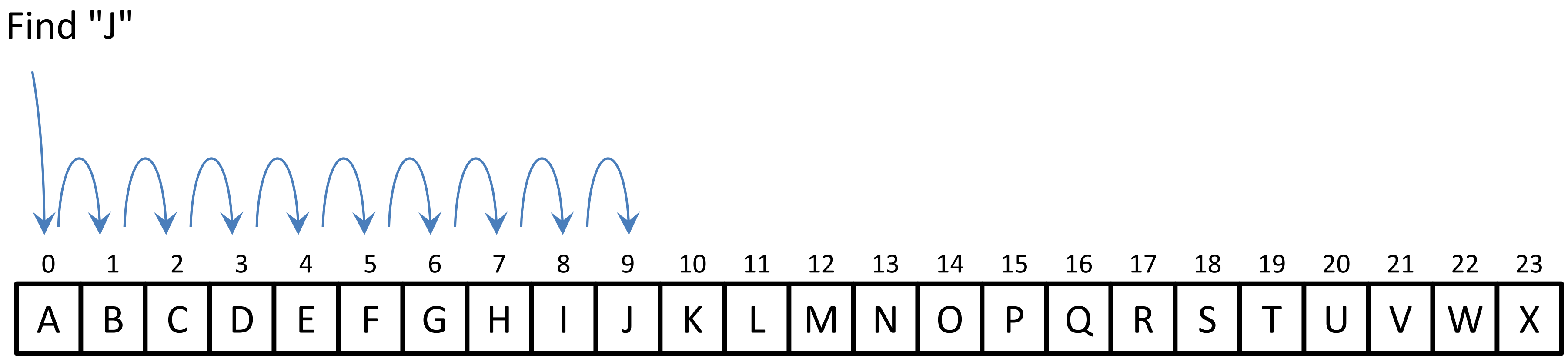
Initial Condition: Random · Nearly Sorted · Reversed · Few Unique

	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick	Quick3
Random								
Nearly Sorted								
Reversed								
Few Unique								

http://www.sorting-algorithms.com/

Exhaustive (Linear) Search

- Systematically enumerate all possible values and compare to value being sought.
  - For an array, iterate from the beginning to the end, and test each item in the array.
- Return the position of the item if found.
- Return nothing (null) if not found



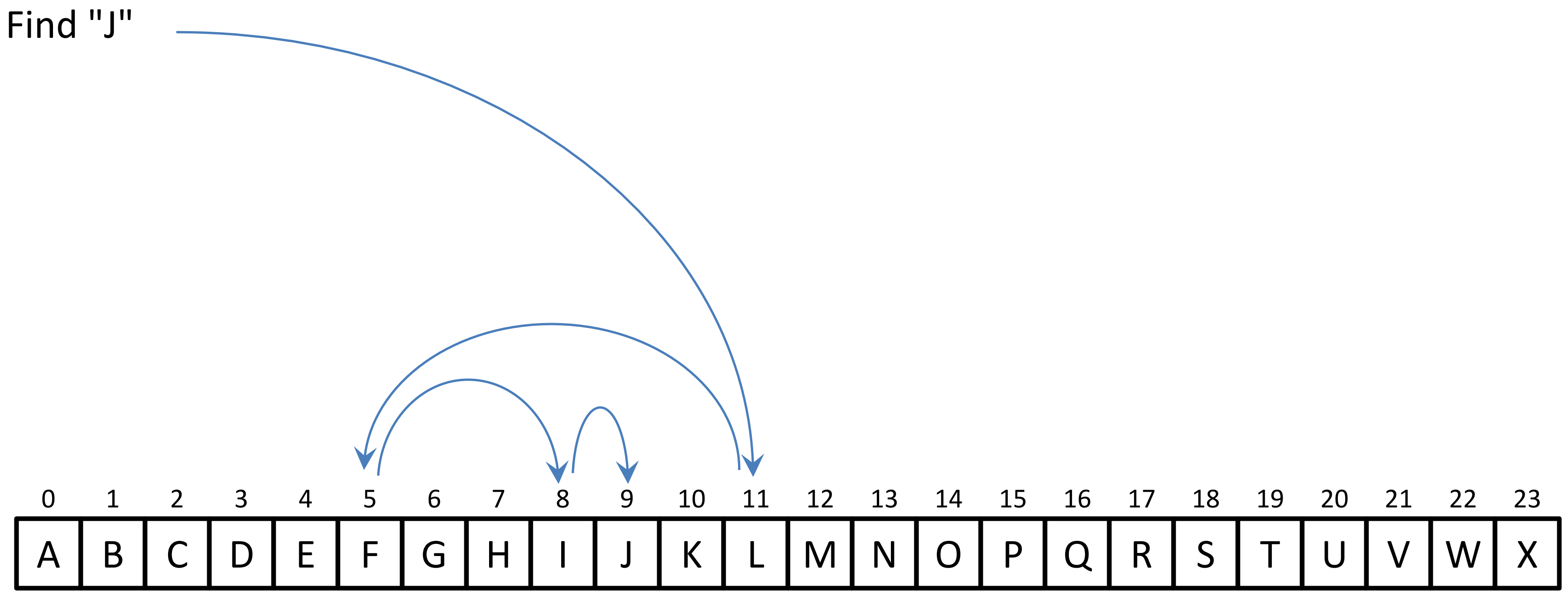
Binary Search

- Quickly find an item (val) in a sorted list.

1. Init **min** and **max** variables to lowest and highest index
2. Repeat while **min <= max**
    a. Compare item at the **middle** index with that being sought (**val**)
    b. If **item** at **middle** equals **val**, return **middle**
    c. If **val** comes before **middle,** then reset **max** to **middle-1**
    d. If **val** comes after **middle**, reset **min** to **middle+1**
3. If **min** > **max**, **val** not found

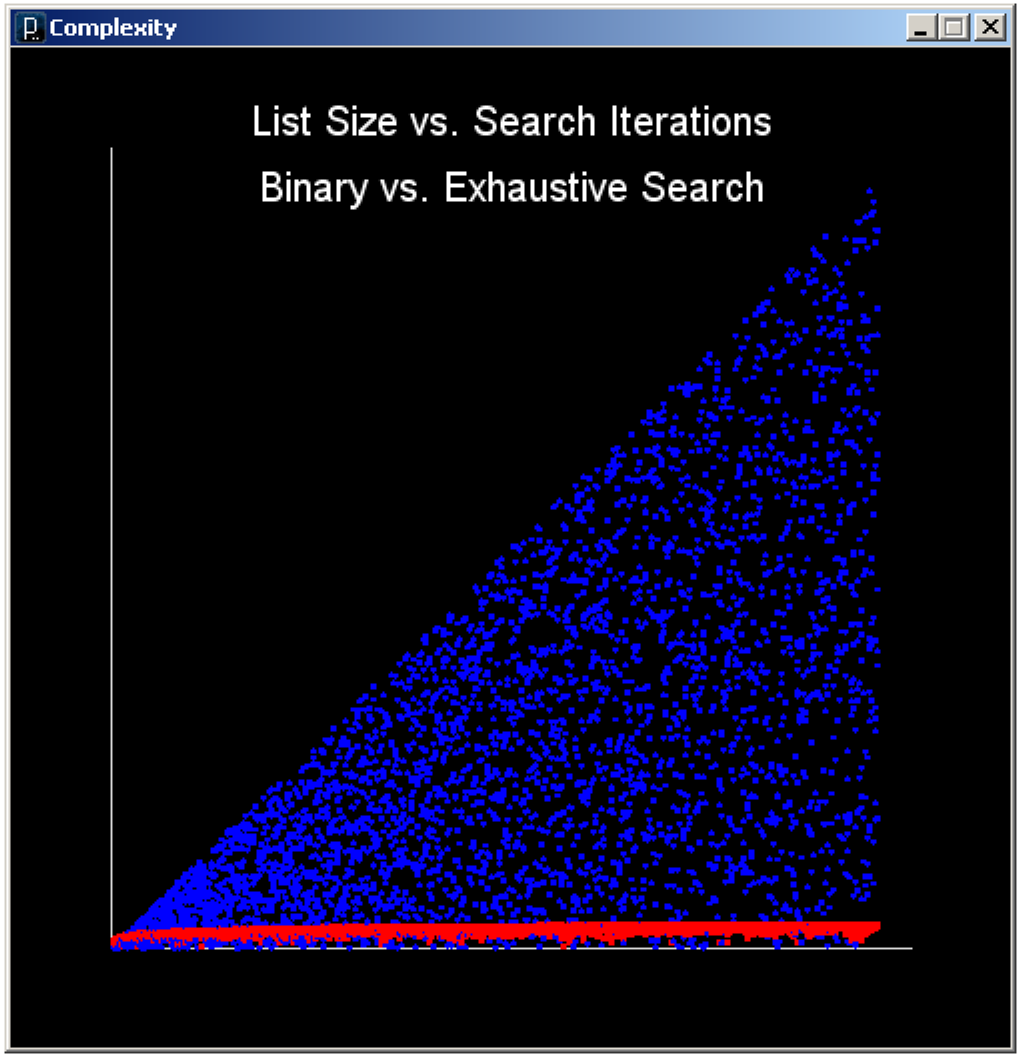
The most efficient way to play "guess the number" …

Binary Search



An Experiment - Exhaustive vs. Binary Search

- For names (Strings) in arrays of increasing size…
  - Select 10 names at random from the list
  - Search for each name using Binary and Exhaustive Search
  - Count the number of iterations it takes to find each name
  - Plot number of iterations for each against list size
- Start with an array of 3830+ names (Strings)



Wow! That's fast!

Worst Case Running Time

- **Exhaustive (Linear) Search**

  N items in a list

  Worst case: Number of iterations = N (we must look at every item)

  It is said that Linear Search executes in O(N) time.

- **Binary Search**

  After 1st iteration, N/2 items remain ($N/2^1$)

  After 2nd iteration, N/4 items remain ($N/2^2$)

  After 3rd iteration, N/8 items remain ($N/2^3$)

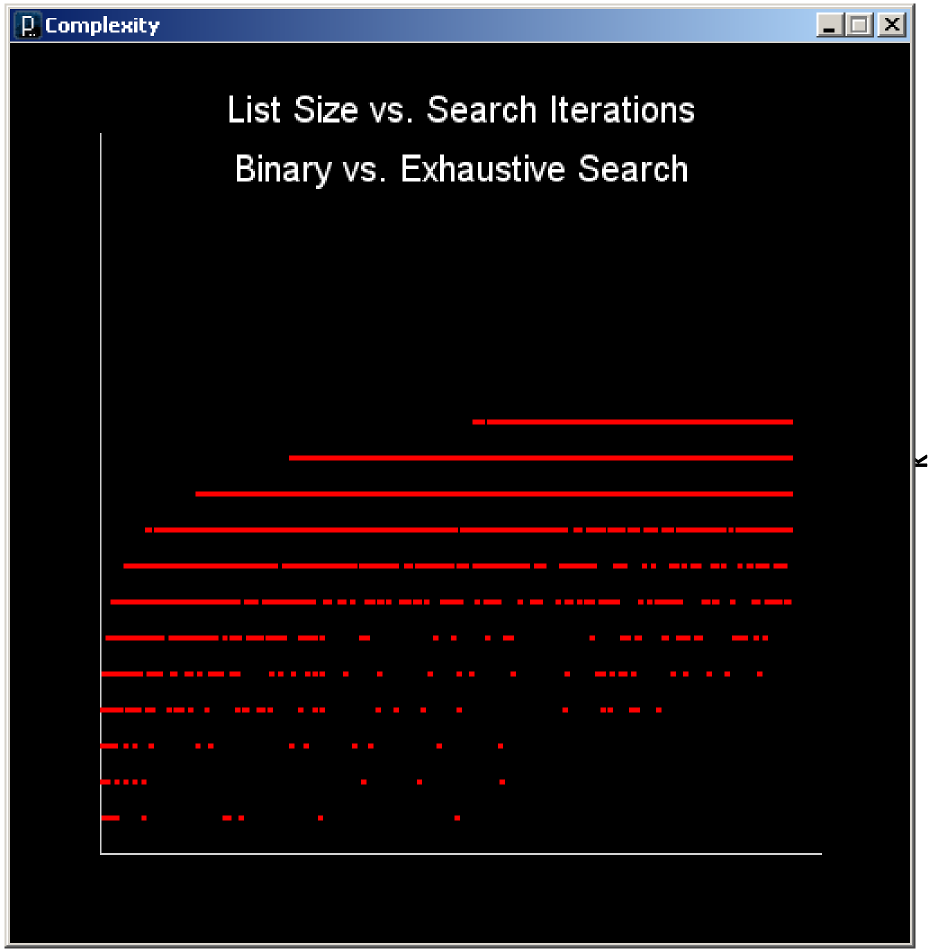
  …

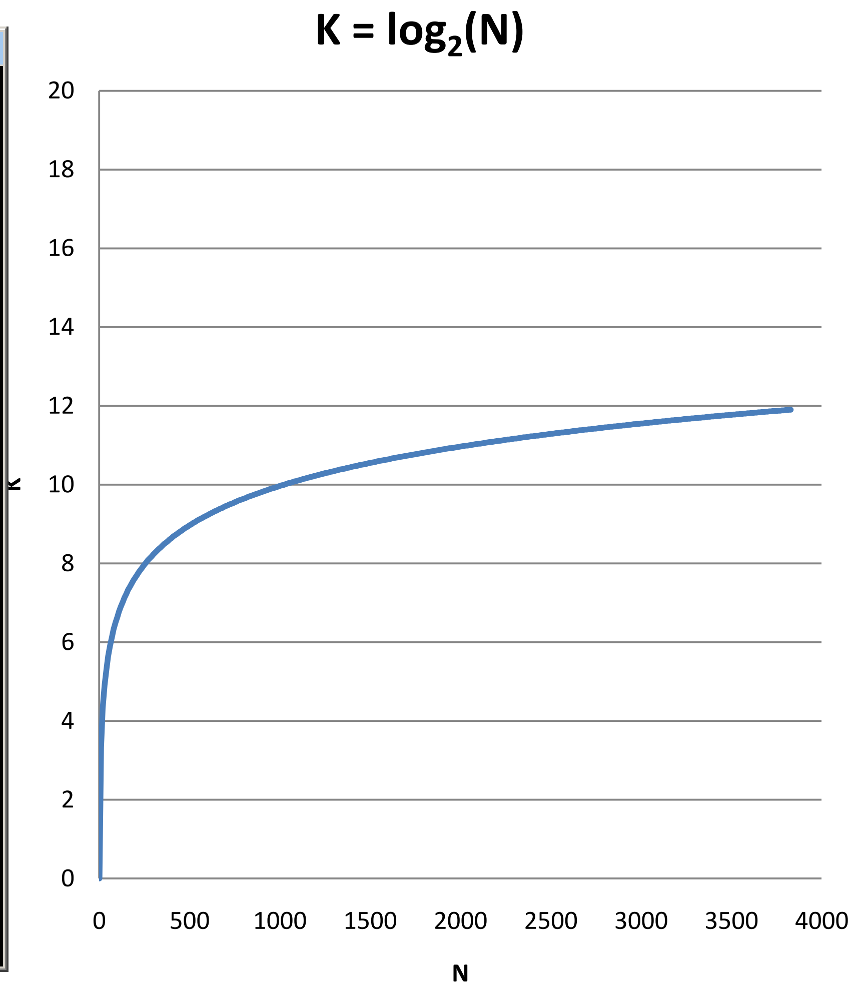
  Search stops when items to search ($N/2^K$) $\rightarrow$ 1

  i.e. $N = 2^K$, $\log_2(N) = K$

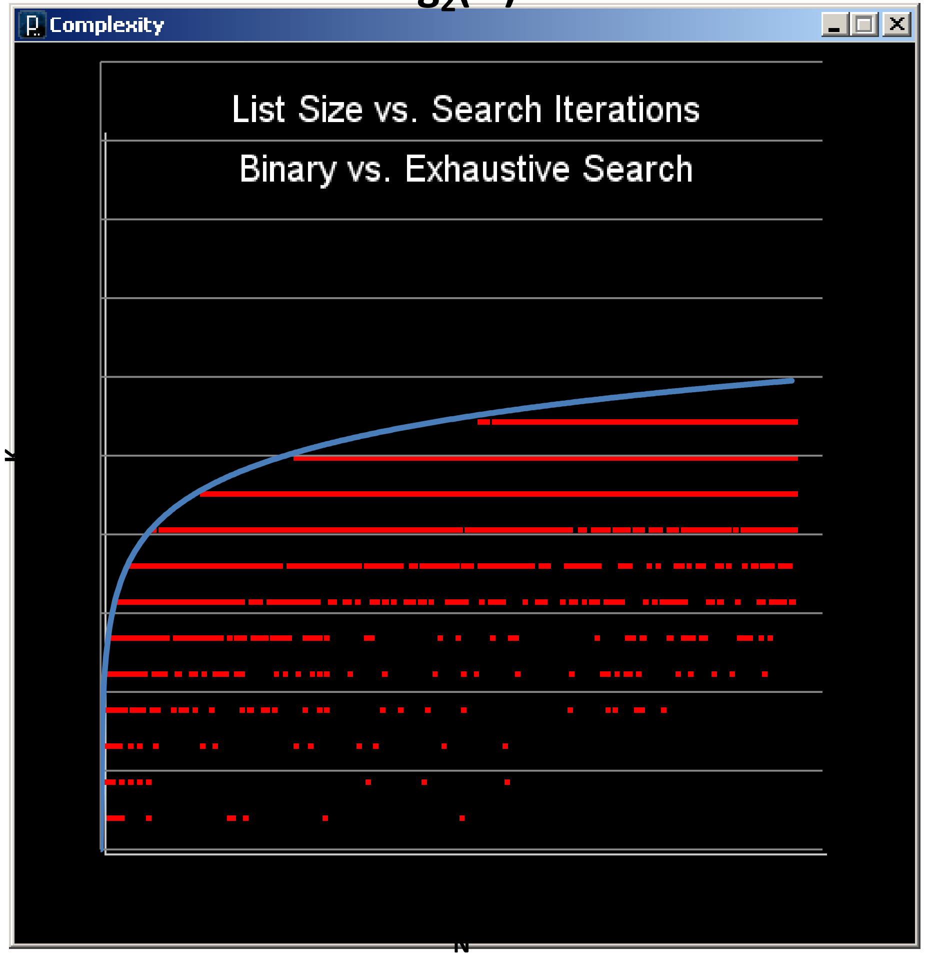
  Worst case: Number of iterations is $\log_2(N)$

It is said that Binary Search is a logarithmic algorithm and executes in O(logN) time.


Complexity
List Size vs. Search Iterations
Binary vs. Exhaustive Search


K = $\log_2(N)$
K
N
0
2
4
6
8
10
12
14
16
18
20
0
500
1000
1500
2000
2500
3000
3500
4000

$K = \log_2(N)$


Complexity
List Size vs. Search Iterations
Binary vs. Exhaustive Search
K
N

Consider the following array:

```
float[] vals = new float[]{ 1, 3, 6, 8, 9, 13, 19, 23, 32, 40 };
```

We could use the following code to determine whether the value x is in the array:

```
float x = 10;
boolean containsValue = false;
for (int i=0; i < vals.length; i++) {
    if (vals[i] == x) {  // comparison
        containsValue = true;
    }
}
```

However, in the worst case, this method requires vals.length (10) comparisons.

5.1 (10 pts) Describe *in detail* how the binary search algorithm would find whether x is in the array. Make certain to describe how the algorithm works.

5.2 (5 pts) How many comparisons would binary search take to solve the same problem? Justify your answer if you're not certain.