Search Algorithms
• Last time we looked at some different searching algorithms
  – Look for a value in a sorted array and return its index if it is there, -1 if it is not
• Linear Search
  – Start at the beginning of the list and examine each element until you find the one you want
• Binary Search
  – Start in the middle of the list, and keep bisecting until you find the one you want

And we discovered that these algorithms required a different number of comparison operations
• Linear Search
  • Takes up to n comparisons to find the value
• Binary Search
  • Takes up to \( \log(n) \) comparisons to find the value

This means that for an array with 1,000,000 elements
• Linear Search
  • Takes up to 1,000,000 comparisons
• Binary Search
  • Takes up to 20 comparisons

Obviously, there's a big difference here
• Your choice of algorithm can have a major impact on how fast your program runs.

In general, it is important to know which algorithms are faster and which are slower
• In particular, we want to know how many operations are required to do a particular algorithm on a given number of data items
• Some algorithms are very efficient, some are doable but slow, and some aren’t doable at all

Notice that
• The \( n \) and \( 2n \) columns grow at the same rate
  • Multiplying by a constant doesn't make much difference
• The \( \log(n) \) and \( n \) columns grow at very different rates
• The \( n \) and \( 2^n \) columns also grow at very different rates
• Different functions of \( n \) make a big difference
**Big O notation**

- Special notation for discussing the performance of algorithms
- Big O notation distills out the important information about how many operations are required by an algorithm
- \( O(f(n)) \) is read as 'order \( f(n) \)' or 'on the order of \( f(n) \)'.

**Big O notation**

- \( O(c \cdot f(n)) = O(f(n)) \) for any constant \( c \)
  - constants don’t matter
  - \( O(2n) = O(n) \)
- Common orders
  - \( O(1) \) – constant time
  - \( O(\log(n)) \) - logarithmic time
  - \( O(n) \) - linear time
  - \( O(2^n) \) - exponential time
- \( O(\log(n)) \ll O(n) \ll O(2^n) \)

**Putting this into Practice**

- Linear Search: \( O(n) \)
- Binary Search: \( O(\log(n)) \)
- Binary Search will generally take less time to execute than linear search
- Binary Search is a more efficient algorithm

**Putting this into Practice**

- Remember Selection Sort
  - Find smallest element in array and put it at the front
  - Find smallest element in rest of array and put it in second position
  - And so on for the rest of the array
- What is the order of selection sort?

**Order of Selection Sort**

- First time takes \( n - 1 \) comparisons, second time takes \( n - 2 \) comparisons, third time takes \( n - 3 \) comparisons, ...
- So, selection sort takes
  \[
  (n-1) + (n-2) + (n-3) + \ldots + 2 + 1 = \\
  n(n - 1)/2 = \\
  (n^2 - n) / 2 \rightarrow O(n^2)
  \]
- Quadratic order
- Not very efficient

**Other Sorting Algorithms**

- Selection Sort: \( O(n^2) \)
- Bubble Sort: \( O(n^2) \)
- Merge Sort: \( O(n \log(n)) \)
- Quick Sort: \( O(n \log(n)) \)
Arrays of other types

- Remember, the elements of an array can be of any type
- So far we have looked mostly at arrays of int
- Book has a couple of examples using other types
  - Sieve of Eratosthenes (page 160)
    - find prime numbers
    - uses an array of boolean
  - CountWord.java
    - uses array of char

Arrays and Types

- Remember, the elements of an array can be of any type
- So far we have looked mostly at arrays of int
- Book has a couple of examples using other types
  - Sieve of Eratosthenes (page 160)
    - find prime numbers
    - uses an array of boolean
  - CountWord.java (page 161)
    - uses array of char

CountWord.java

```java
//CountWord.java
import tio.*;
public class CountWord { 
  public static void main(String[] args) { 
    String input;
    char[] buffer;
    System.out.println("Type in a line ");
    input = Console.in.readLine();
    System.out.println(input);
    buffer = input.toCharArray();
    System.out.println("word count is "+
      wordCount(buffer));
  }
}
```

CountWord.java

```java
//words are separated by nonalphabetic characters
public static int wordCount( char[] buf ) {
  int position = 0, wc = 0;
  while (position < buf.length) {
    while (position < buf.length && !isAlpha( buf[position] ))
      position++;
    if (position < buf.length)
      wc++;
    while (position < buf.length && isAlpha( buf[position] ))
      position++;
  }
  return wc;
}
```

CountWord.java

```java
public static boolean isAlpha(char c) {
  return (c >= 'a' && c <= 'z') ||
    (c >= 'A' && c <= 'Z');
}
```

Arrays of non-primitive Types

- Array elements can be of any type
- Remember that String is a non-primitive type
- So, we can have an array of String

```java
String[] hello;
hello = new String[2];
hello[1] = "Hello ";
hello[2] = "Brian!";
```

Or you can initialize the array this way

```java
String[] hello = {"Hello ", "Brian!"};
```
**Array of String: Example**

```java
// StringArray.java - uses a string array initializer
class StringArray {
    public static void main(String[] args) {
        String[] myStringArray = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};
        for (int i = 0; i < myStringArray.length; i++)
            System.out.println(myStringArray[i]);
    }
}
```

**Main (String[] args )**

- Now we know what 'String[] args' means
- It's an array of String that is passed to the method main as a parameter
- These are called the *command-line arguments*
  - The parameters passed to the program on the command line
  - `java myProgram alpha beta gamma`
    - passes "alpha", "beta", and "gamma" to the program in args

**CommandLine.java**

```java
// CommandLine.java - print command line arguments
class CommandLine {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

**Multi-dimensional Arrays**

- The elements of an array can be any type
  - Including an array type
- So
  - `int 2D[][];` declares an array of arrays of int
- Two dimensional arrays are useful for representing tables of data with rows and columns
- Three dimensional arrays are useful for representing volume data with three coordinates x, y, and z

**Two-Dimensional Arrays**

- A 2 dimensional array is declared and allocated like this:
  - `int a[][] = new int[3][4];`
    
  - `a` is an array of 3 arrays of int
  - `a[0]` is an array of 4 ints
  - `a[1]` is an array of 4 ints
  - `a[2]` is an array of 4 ints

**Two-Dimensional Array Indexing**

- Array elements are access by providing a value for all subscripts.
  
  a[0][0]  a[0][1]  a[0][2]  a[0][3]  
  a[1][0]  a[1][1]  a[1][2]  a[1][3]  

  - The first subscript represents the row
  - The second subscript represents the column
Two-Dimensional Array Example

```java
// Multiplication table
class Mult {
    public static void main(String[] args) {
        int[][] data = new int[10][10];
        for(int i = 0; i < data.length; i++) {
            for(int j = 0; j < data[i].length; j++) {
                data[i][j] = i * j;
            }
        }
        for(int i = 0; i < data.length; i++) {
            for(int j = 0; j < data[i].length; j++) {
                System.out.print(data[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Two-Dimensional Array Initializer

- Remember, we can initialize a single dimensional array
  ```java
  int temperature[] = {32, 35, 38, 36, 38, 35, 40};
  ```
- We can also initialize two-dimensional arrays
  ```java
  int temperature[][] = {{32, 58}, {35, 60},
                        {38, 59}, {36, 63}, {38, 61},
                        {35, 65}, {40, 68}};
  ```

Two-Dimensional Arrays

- 2-dimensional arrays don't have to be 'rectangular'
  ```java
  int a[][] = {{2, 5}, {3, 4, 0}, {3}, {4, 5}};
  ```

```
  a[0]  array of 2 ints
  a[1]  array of 3 ints
  a[2]  array of 1 int
  a[3]  array of 2 ints
```

Distributed Pair Programming

- Pair Programming has been shown to provide many benefits
- However, there is at least one major drawback
  - You have to be collocated
- I am developing a tool to support distributed pair programming
  - Supports pair programming from 2 separate locations

Seeking Volunteers

- I am looking for some students who would like to try to use this tool
  - You can use it on your homework assignments
- What do I want?
  - Your honest feedback regarding your experience using the tool
    - Did it work?
    - How did distributed pairing compare to collocated pairing?
    - What would you do to improve the tool?

Requirements

- You must have:
  - A PC running Windows
  - A sound card
  - A high speed Internet connection (DSL, campus network, cable modem)
    - Dial-up is probably not sufficient, although I don't know for sure
- You also need to have or set up:
  - SSH (secure shell)
  - AOL Instant Messenger
Next Time

- More about arrays
  - Read book sections about two-dimensional arrays and the ‘Game of Life’
- Sample questions for quiz 4
- Enjoy your 3-day weekend