CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

5: Analysis of Algorithms

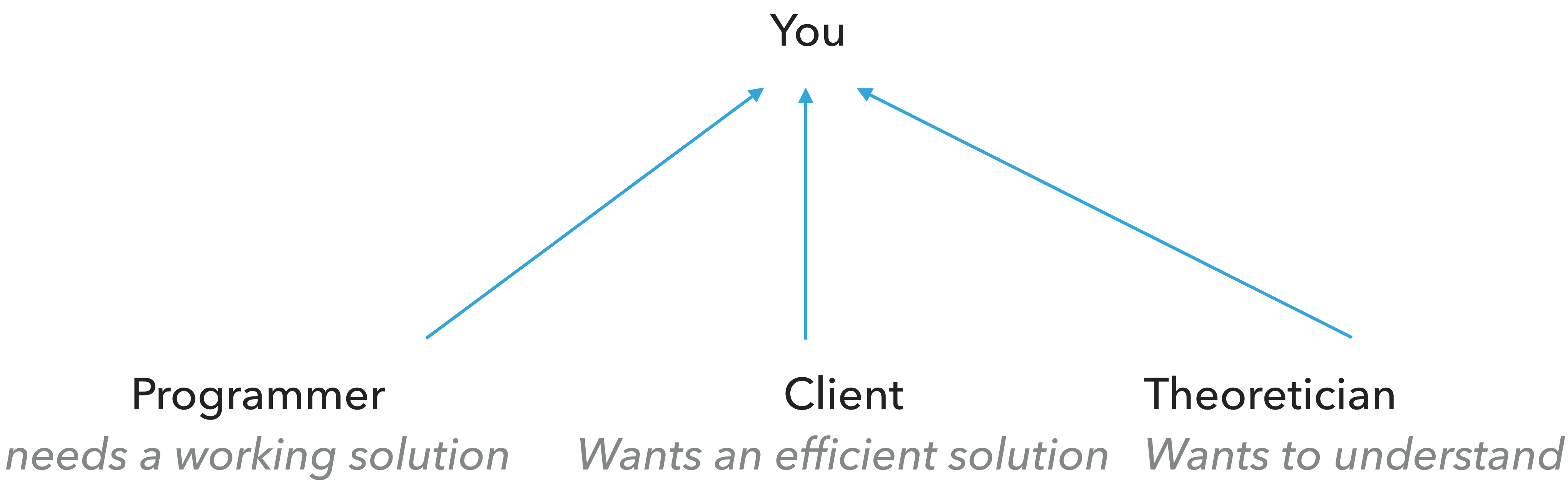
Tom Yeh
he/him/his

Lecture 5: Analysis of Algorithms

- Introduction
- Experimental Analysis of Running Time
- Mathematical Models of Running Time
- Order of Growth Classification
- Analysis of Memory Consumption

Some slides adopted from Algorithms 4th Edition or COS226

Different Roles



Why analyze algorithmic efficiency?

- Predict performance.
- Compare algorithms that solve the same problem.
- Provide guarantees.
- Understand theoretical basis.
- **Avoid performance bugs.**

Why is my program so slow?
Why does it run out of memory?

We can use a combination of experiments and mathematical modeling.

Lecture 5: Analysis of Algorithms

- Introduction
- Experimental Analysis of Running Time
- Mathematical Models of Running Time
- Order of Growth Classification
- Analysis of Memory Consumption

- 3-SUM: Given n distinct numbers, how many unordered triplets sum to 0?

- Input: 30 -40 -20 -10 40 0 10 5
- Output: 4
  - 30 -40 10
  - 30 -20 -10
  - -40 40 0
  - -10 0 10

- 3-SUM: brute-force algorithm

```
public class ThreeSum {

public static int count(int[] a) {
        int n = a.length;
        int count = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i+1; j < n; j++) {
                for (int k = j+1; k < n; k++) {
                    if (a[i] + a[j] + a[k] == 0) {
                        count++;
                    }
                }
            }
        }
        return count;
    }
```

```
    public static void main(String[] args)  {
            String filename = args[0];
            int fileSize = Integer.parseInt(args[1]);
            try {
                Scanner scanner = new Scanner(new File(filename));
                int intList[] = new int[fileSize];
                int i=0;
                while(scanner.hasNextInt()){
                    intList[i++]=scanner.nextInt();
                }
                Stopwatch timer = new Stopwatch();
                int count = count(intList);
                System.out.println("elapsed time = " + timer.elapsedTime());
                System.out.println(count);
            }
            catch (IOException ioe) {
                throw new IllegalArgumentException("Could not open " + filename, ioe);
            }
        }
```

- Empirical Analysis

- Input: 8ints.txt
- Output: 4 and 0

- Input: 1Kints.txt
- Output: 70 and 0.081

- Input: 2Kints.txt
- Output: 528 and 0.38

- Input: 2Kints.txt
- Output: 528 and 0.371

- Input: 4Kints.txt
- Output: 4039 and 2.792

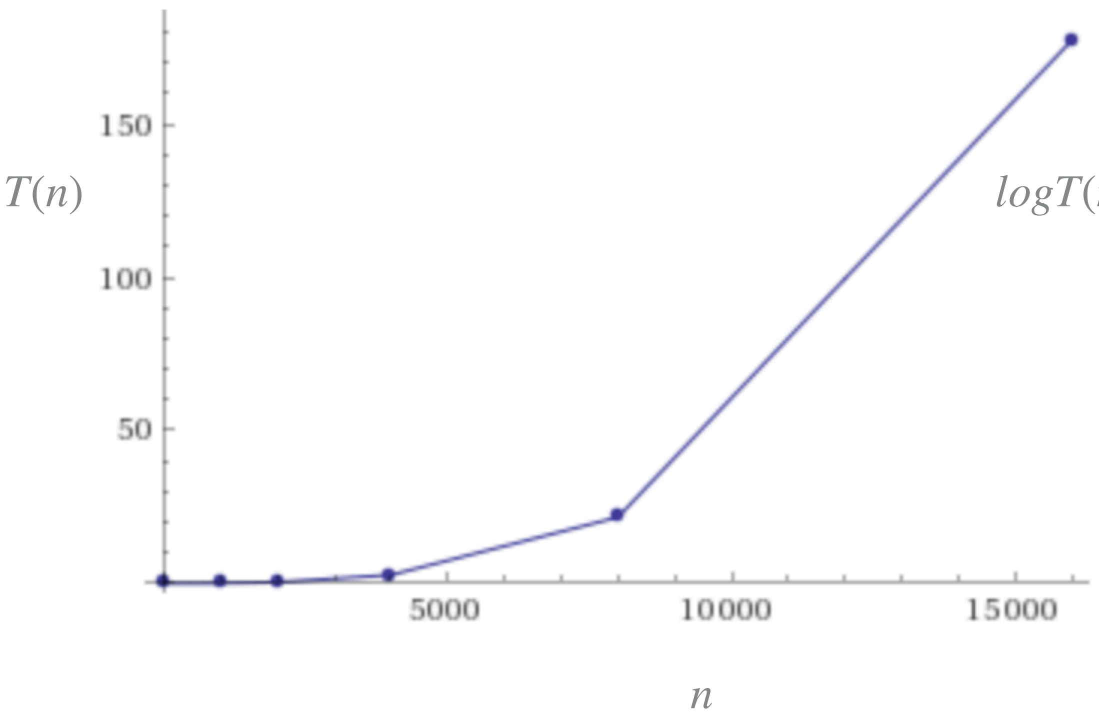
- Input: 8Kints.txt
- Output: 32074 and 21.623

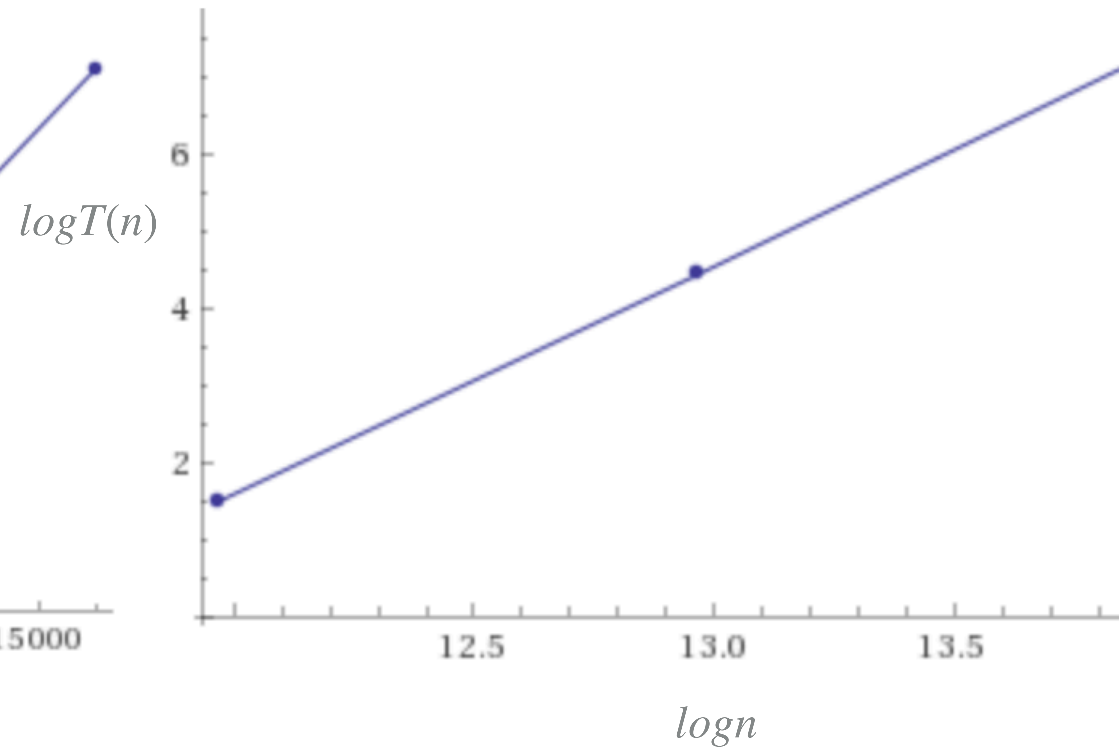
- Input: 16Kints.txt
- Output: 255181 and 177.344

Input size	Time
8	0
1000	0.081
2000	0.38
2000	0.371
4000	2.792
8000	21.623
16000	177.344

- Plots and log-log plots

Straight line of slope 3





- Regression: $T(n) = an^b$ (power-law), where *n* is problem size. $\log T(n) = b \log n + \log a$, where b is slope.
- Experimentally: ~$0.42 \times 10^{-10} n^3$, in our example for `ThreeSum`.

Input size	Time
8	0
1000	0.081
2000	0.38
4000	2.792
8000	21.623
16000	177.344

- Doubling hypothesis

- Doubling input size increases running time by a factor of $\frac{T(n)}{T(n/2)}$
- Run program doubling the size of input. Estimate factor of growth:
  - $\frac{T(n)}{T(n/2)} = \frac{an^b}{a(\frac{n}{2})^b} = 2^b.$
- E.g., in our example, for pair of input sizes 8000 and 16000 the ratio is 8.2, therefore b is approximately 3.
- Assuming we know b, we can figure out a.
  - E.g., in our example, $T(16000) = 177.34 = a \times 16000^3$.
    - Solving for a we get $a = 0.42 \times 10^{-10}$.

- Practice Time

- Suppose you time your code and you make the following observations. Which function is the closest model of $T(n)$?

A. n^2

B. $6 \times 10^{-4} n$

C. $5 \times 10^{-9} n^2$

D. $7 \times 10^{-9} n^2$

Input size	Time
1000	0
2000	0.0
4000	0.1
8000	0.3
16000	1.3
32000	5.1

- Answer

- C. $5 \times 10^{-9} n^2$
- Ratio is approximately 4, therefore $b = 2$.
- $T(32000) = 5.1 = a \times 32000^2$.
- Solving for $a = 4.98 \times 10^{-9}$.s

Input size	Time
1000	0
2000	0.0
4000	0.1
8000	0.3
16000	1.3
32000	5.1

- Effects on performance

- System independent effects: Algorithm + input data
  - Determine b in power law relationships.
- System dependent effects: Hardware (e.g., CPU, memory, cache) + Software (e.g., compiler, garbage collector) + System (E.g., operating system, network, etc).

- Dependent and independent effects determine a in power law relationships.

- Although it is hard to get precise measurements, experiments in Computer Science are cheap to run.

Lecture 5: Analysis of Algorithms

- Introduction
- Experimental Analysis of Running Time
- Mathematical Models of Running Time
- Order of Growth Classification
- Analysis of Memory Consumption

- Total Running Time

- Popularized by Donald Knuth in the 60s in the four volumes of "The Art of Computer Programming".
  - Knuth won the Turing Award (The "Nobel" in CS) in 1974.

- In principle, accurate mathematical models for performance of algorithms are available.

- Total running time = sum of cost x frequency for all operations.
- Need to analyze program to determine set of operations.
- Exact cost depends on machine, compiler.
- Frequency depends on algorithm and input data.

- Cost of basic operations

- Add < integer multiply < integer divide < floating-point add < floating-point multiply < floating-point divide.

Operation	Example	Nanoseconds
Variable declaration	`int a`	c_1
Assignment statement	`a = b`	c_2
Integer comparison	`a < b`	c_3
Array element access	`a[i]`	c_4
Array length	`a.length`	c_5
1D array allocation	`new int[n]`	$c_6 n$
2D array allocation	`new int[n][n]`	$c_7 n^2$
string concatenation	`s+t`	$c_8 n$

- Example: 1-SUM

- How many operations as a function of n?

```
int count = 0;
 for (int i = 0; i < n; i++) {
     if (a[i] == 0) {
         count++;
     }
 }
```

Operation	Frequency
Variable declaration	2
Assignment	2
Less than	$n+1$
Equal to	n
Array access	n
Increment	n to $2n$

- Example: 2-SUM

- How many operations as a function of n?

```
int count = 0;
 for (int i = 0; i < n; i++) {
     for (int j = i+1; j < n; j++) {
         if (a[i] + a[j] == 0) {
             count++;
         }
     }
 }
```

BECOMING TOO TEDIOUS TO CALCULATE

Operation	Frequency
Variable declaration	$n + 2$
Assignment	$n + 2$
Less than	$1/2(n+1)(n+2)$
Equal to	$1/2n(n-1)$
Array access	$n(n-1)$
Increment	$1/2n(n+1)$ to n^2

- Tilde notation

- Estimate running time (or memory) as a function of input size n.
- Ignore lower order terms.
  - When n is large, lower order terms become negligible.

- Example 1: $\frac{1}{6}n^3 + 10n + 100 \quad \sim n^3$

- Example 2: $\frac{1}{6}n^3 + 100n^2 + 47 \quad \sim n^3$

- Example 3: $\frac{1}{6}n^3 + 100n^{\frac{2}{3}} + \frac{1/2}{n} \quad \sim n^3$

- Technically $f(n) \sim g(n)$ means that $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$

- Simplification

- Cost model: Use some basic operation as proxy for running time.
  - E.g., array accesses
- Combine it with tilde notation.

Operation	Frequency	Tilde notation
Variable declaration	$n+2$	$\sim n$
Assignment	$n+2$	$\sim n$
Less than	$1/2(n+1)(n+2)$	$\sim n^2$
Equal to	$1/2n(n-1)$	$\sim n^2$
Array access	$n(n-1)$	$\sim n^2$
Increment	$1/2n(n+1)$ to n^2	$\sim n^2$

- $\sim n^2$ array accesses for the 2-SUM problem

- Back to the 3-SUM problem

- Approximately how many array accesses as a function of input size n?

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = i+1; j < n; j++) {
        for (int k = j+1; k < n; k++) {
            if (a[i] + a[j] + a[k] == 0) {
                count++;
            }
        }
    }
}
```

- n^3 array accesses.

- Useful approximations for the analysis of algorithms

- Harmonic sum: $H_n = 1 + 1/2 + 1/3 + ... + 1/n \quad \sim \ln n$
- Triangular sum: $1 + 2 + 3 + ... + n \quad \sim n^2$
- Geometric sum: $1 + 2 + 4 + 8 + ... + n = 2n - 1 \quad \sim n$, when n power of 2.
- Binomial coefficients: $\binom{n}{k} \quad \sim \frac{n^k}{k!}$ when k is a small constant.

- Use a tool like Wolfram alpha.

- Practice Time

- How many array accesses does the following code make?

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = i+1; j < n; j++) {
        for (int k = 1; k < n; k=k*2) {
            if (a[i] + a[j] >= a[k]) {
            count++;
        }
    }
}
```

A. n^2

B. $n^2 \log n$

C. n^3

D. $n^3 \log n$

- Answer

- $n^2 \log n$

Lecture 5: Analysis of Algorithms

- Introduction
- Experimental Analysis of Running Time
- Mathematical Models of Running Time
- Order of Growth Classification
- Analysis of Memory Consumption

- Order-of-growth

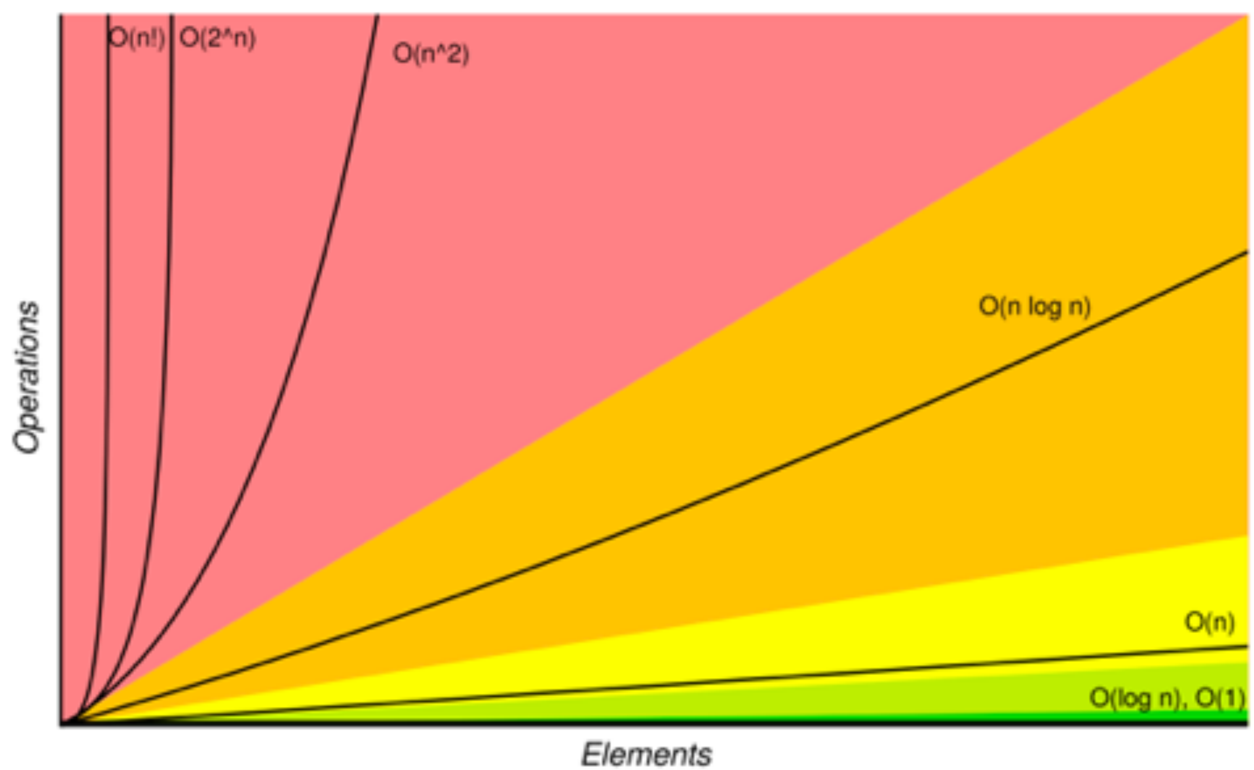
- Definition: If $f(n) \sim cg(n)$ for some constant $c > 0$, then the order of growth of $f(n)$ is $g(n)$.
  - Ignore leading coefficients.
  - Ignore lower-order terms.

- We will use this definition in the mathematical analysis of the running time of our programs as the coefficients depend on the system.
- E.g., the order of growth of the running time of the `ThreeSum` program is n^3.

ORDER OF GROWTH CLASSIFICATION

- Common order-of-growth classifications

- Good news: only a small number of function suffice to describe the order-of-growth of typical algorithms.
- 1: constant
- $\log n$: logarithmic
- n : linear
- $n \log n$: linearithmic
- n^2: quadratic
- n^3: cubic
- 2^n: exponential
- $n!$: factorial



bigocheatsheet.com

- Common order-of-growth classifications

Order-of-growth	Name	Typical code	$T(n)/T(n/2)$
1	Constant	a=b+c	1
$\log n$	Logarithmic	while(n>1){n=n/2;...}	~ 1
n	Linear	for(int i =0; i<n;i++{ ...}	2
$n \log n$	Linearithmic	mergesort	~ 2
n^2	Quadratic	for(int i =0;i<n;i++) { for(int j=0; j<n;j++){...}}	4
n^3	Cubic	for(int i =0;i<n;i++) { for(int j=0; j<n;j++){ for(int k=0; k<n; k++){...}}}	8

Lecture 5: Analysis of Algorithms

- Introduction
- Experimental Analysis of Running Time
- Mathematical Models of Running Time
- Order of Growth Classification
- Analysis of Memory Consumption

- Basics

- Bit: 0 or 1.
- Byte: 8 bits.
- Megabyte (MB): 2^{20} bytes.
- Gigabyte: 2^{30} bytes.

- Typical memory usage for primitives and arrays

- `boolean`: 1 byte
- `byte`: 1 byte
- `char`: 2 bytes
- `int`: 4 bytes
- `float`: 4 bytes
- `long`: 8 bytes
- `double`: 8 bytes
- Array overhead: 24 bytes
- `char[]`:2n+24 bytes
- `int[]`:4n+24 bytes
- `double[]`:8n+24 bytes

- Typical memory usage for objects

- Object overhead: 16 bytes
- Reference: 8 bytes
- Padding: padded to be a multiple of 8 bytes
- Example:
  - ```
    public class Date {
            private int day;
            private int month;
            private int year;
    }
    ```
  - 16 bytes overhead + 3x4 bytes for ints + 4 bytes padding = 32 bytes
    ```

- Practice Time

- How much memory does `WeightedQuickUnionUF` use as a function of n?

```
public class WeightedQuickUnionUF{
    private int[] parent;
    private int[] size;
    private int count;

    public WeightedQuickUnionUF(int n) {
        parent = new int[n];
        size = new int[n];
        count = 0;
…
}
```

A. ~$4n$ bytes

B. ~$8n$ bytes

C. ~$4n^2$ bytes

D. ~$8n^2$ bytes

- Answer

B. ~$8n$ bytes

- 16 bytes for object overhead
- Each array: 8 bytes for reference + 24 overhead + 4n for integers
- 4 bytes for int
- 4 bytes for padding
- Total $88 + 8n \sim 8n$

Lecture 5: Analysis of Algorithms

- Introduction
- Experimental Analysis of Running Time
- Mathematical Models of Running Time
- Order of Growth Classification
- Analysis of Memory Consumption

Readings:

- Textbook:
  - Chapter 1.4 (pages 172-196, 200-205)
- Website:
  - Analysis of Algorithms: https://algs4.cs.princeton.edu/14analysis/

Practice Problems:

- 1.4.1-1.4.9

Finish Java Catch-All Lecture

Pre and post conditions

- Pre-condition: Specification of what must be true for method to work properly.
- Post-condition: Specification of what must be true at end of method if precondition held before execution.

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

Lecture 4: The Catch-All Java Lecture

- Packages
- JavaDoc
- Exceptions
- Assertions
- Text I/O
- Java GUIs
- Graphics
- Events

I/O streams

- Input stream: a sequence of data into the program.
- Output stream: a sequence of data out of the program.
- Stream sources and destinations include disk files, keyboard, peripherals, memory arrays, other programs, etc.
- Data stored in variables, objects and data structures are temporary and lost when the program terminates. Streams allow us to save them in files, e.g., on disk or CD (!)
- Streams can support different kinds of data: bytes, principles, characters, objects, etc.

https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Files

- Every file is placed in a directory in the file system.
- Absolute file name: the file name with its complete path and drive letter.
  - e.g., on Windows: `C:\temp\somefile.txt`
  - On Mac/Unix: `/home/temp/somefile.txt`
- `File`: contains methods for obtaining file properties, renaming, and deleting files. Not for reading/writing!
- CAUTION: DIRECTORY SEPARATOR IN WINDOWS IS \, WHICH IS SPECIAL CHARACTER IN JAVA. SHOULD BE "\\" INSTEAD.

TEXT I/O

```java
/**
 * Demonstrates File class and its operations.
 * @author https://liveexample.pearsoncmg.com/html/TestFileClass.html
 *
 */

import java.io.File;
import java.util.Date;

public class TestFile {
  public static void main(String[] args) {
    File file = new File("some.text");
    System.out.println("Does it exist? " + file.exists());
    System.out.println("The file has " + file.length() + " bytes");
    System.out.println("Can it be read? " + file.canRead());
    System.out.println("Can it be written? " + file.canWrite());
    System.out.println("Is it a directory? " + file.isDirectory());
    System.out.println("Is it a file? " + file.isFile());
    System.out.println("Is it absolute? " + file.isAbsolute());
    System.out.println("Is it hidden? " + file.isHidden());
    System.out.println("Absolute path is " + file.getAbsolutePath());
    System.out.println("Last modified on " + new Date(file.lastModified()));
  }
}
```

Writing data to a text file

- `PrintWriter output = new PrintWriter(new File("filename"));`
- New file will be created. If already exists, discard.
- Invoking the constructor may throw an I/O Exception…
- `output.print` and `output.println` work with Strings, and primitives.
- Always close a stream!

TEXT I/O

```java
/**
 * Demonstrates how to write to text file.
 * @author https://liveexample.pearsoncmg.com/html/WriteData.html
 *
 */

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;

public class WriteData {
    public static void main(String[] args) {

        PrintWriter output = null;
        try {
            output = new PrintWriter(new File("addresses.txt"));
            // Write formatted output to the file
            output.print("Alexandra Papoutsaki ");
            output.println(222);
            output.print(“Tom Yeh ");
            output.println(128);

        } catch (IOException e) {
            System.err.println(e.getMessage());
        } finally {
            if (output != null)
                output.close();
        }
    }
}
```

Reading data from a text file

- `java.util.Scanner` reads Strings and primitives.
- Breaks input into tokens, demoted by whitespaces.
- To read from keyboard: `Scanner input = new Scanner(System.in);`
- To read from file: `Scanner input = new Scanner(new File("filename"));`
- Need to close stream as before.
- `hasNext()` tells us if there are more tokens in the stream. `next()` returns one token at a time.
  - Variations of next are `nextLine()`, `nextByte()`, `nextShort()`, etc.

TEXT I/O

```java
/**
 * Demonstrates how to read data from a text file.
 * @author https://liveexample.pearsoncmg.com/html/ReadData.html
 *
 */

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class ReadData {
    public static void main(String[] args) {

        Scanner input = null;
        // Create a Scanner for the file
        try {
            input = new Scanner(new File("addresses.txt"));

            // Read data from a file
            while (input.hasNext()) {
                String firstName = input.next();
                String lastName = input.next();
                int room = input.nextInt();
                System.out.println(firstName + " " + lastName + " " + room);
            }
        } catch (IOException e) {
            System.err.println(e.getMessage());
        } finally {
            if (input != null)
                input.close();
        }

    }
}
```

Lecture 4: The Catch-All Java Lecture

- Packages
- JavaDoc
- Exceptions
- Assertions
- Text I/O
- Java GUIs
- Graphics
- Events

GUIs

- AWT: The Abstract Windowing Toolkit is found in the package `java.awt`
  - Heavyweight components.
  - Implemented with native code written for that particular computer.
  - The AWT library was written in six weeks!
- Swing: Java 1.2 extended AWT with the `javax.swing` package.
  - Lightweight components.
  - Written in Java.

JFrame

- `javax.swing.JFrame` inherits from `java.awt.Frame`
- The outermost container in an application.
- To display a window in Java:
  - Create a class that extends `JFrame`.
  - Set the size.
  - Set the location.
  - Set it visible.

JFrame

```
import javax.swing.JFrame;

public class MyFirstGUI extends JFrame {

    public MyFirstGUI() {
        super("First Frame");
        setSize(500, 300);
        setLocation(100, 100);
        setVisible(true);
    }

    public static void main(String[] args) {
        MyFirstGUI mfgui = new MyFirstGUI();
    }

}
```

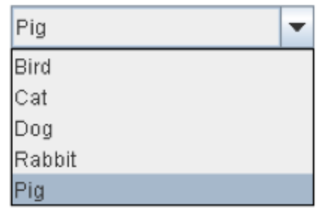
Closing a GUI

- The default operation of the quit button is to set the visibility to false. The program does not terminate!
- `setDefaultCloseOperation` can be used to control this behavior.
- `mfgui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
- More options (hide, do nothing, etc).

Basic components

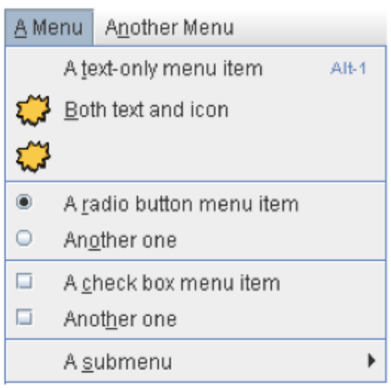
JButton

JCheckBox

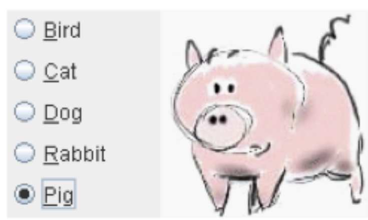


JComboBox

JList



JMenu



JRadioButton

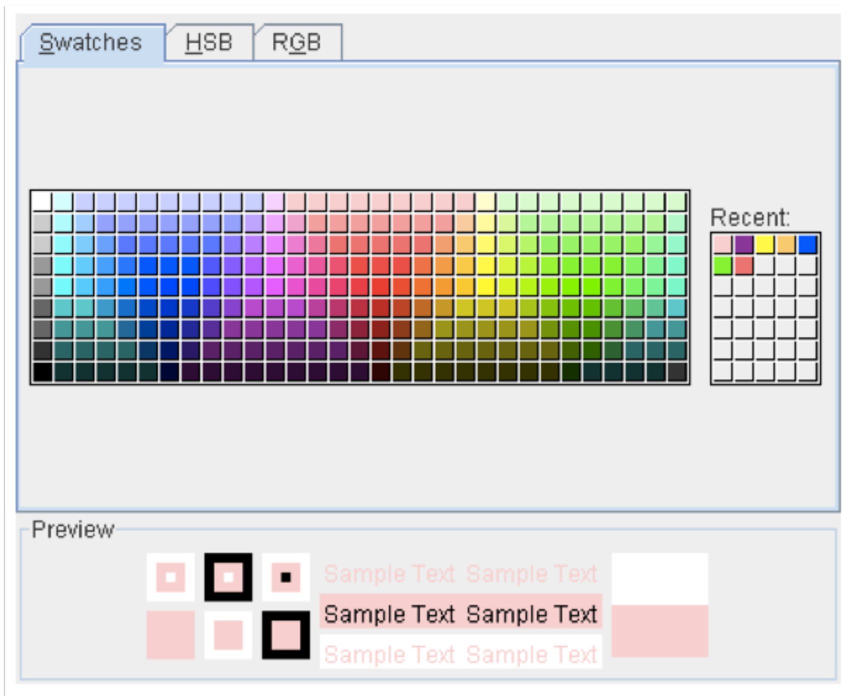
JSlider

JSpinner

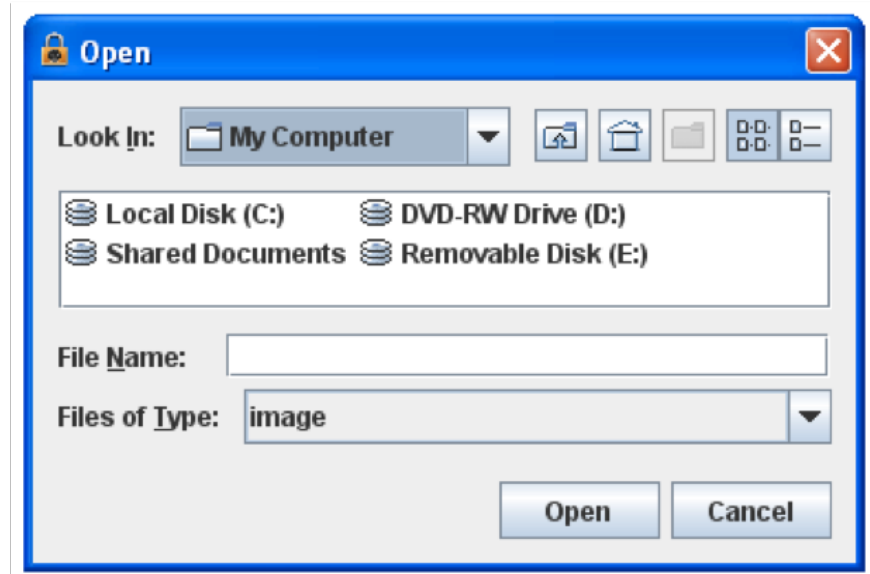
JTextField

JPasswordField

Interactive displays



JColorChooser



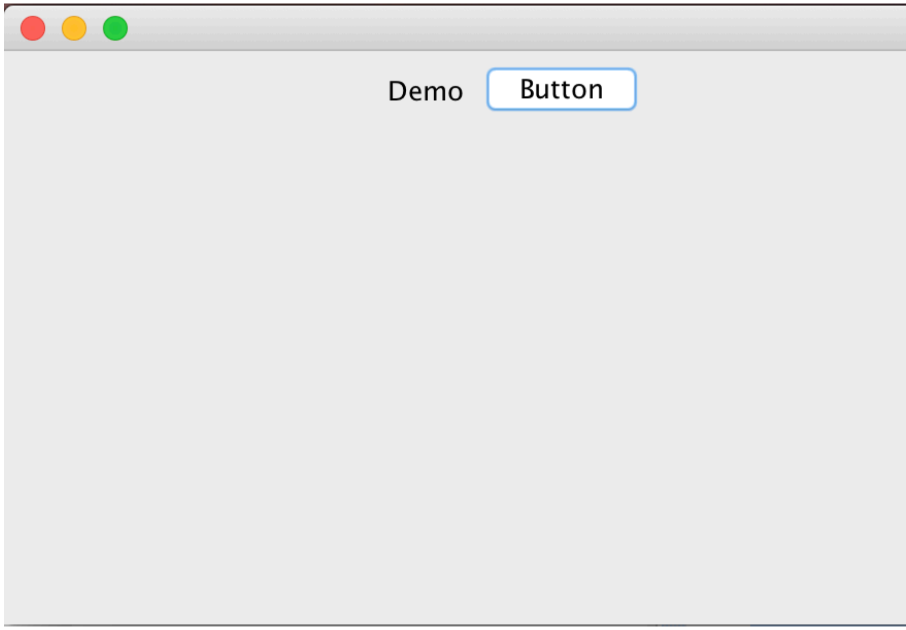
JFileChooser

Adding JComponents to JFrame

```
import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class GUIDemo extends JFrame {
    public GUIDemo() {
        // Container cp = getContentPane();
        // cp.setLayout(new FlowLayout());
        // cp.add(new JLabel("Demo"));
        // cp.add(new JButton("Button"));
        JPanel mainPanel = new JPanel(new FlowLayout());
        mainPanel.add(new JLabel("Demo"));
        mainPanel.add(new JButton("Button"));
        setContentPane(mainPanel);
        setSize(500, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        GUIDemo gd = new GUIDemo();
        gd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



Lecture 4: The Catch-All Java Lecture

- Packages
- JavaDoc
- Exceptions
- Assertions
- Text I/O
- Java GUIs
- Graphics
- Events

Java Graphics

- Create arbitrary objects you want to draw:
  - `Rectangle2D.Double`, `Line.Double`, etc.
  - Constructors take x, y coordinates and dimensions, but don't actually draw items.
- All drawing takes place in `paint` method using a "graphics content".
- Triggered implicitly by uncovering window or explicitly by calling the `repaint` method.
  - Adds repaint event to draw queue and eventually draws it.

Graphics context

- All drawing is done in `paint` method of component.
- `public void paint (Graphics g)`
- `g` is a graphics context provided by the system.
- "pen" that does the drawing.
- You call `repaint()` not `paint()`.
- Need to import classes from `java.awt.*`, `java.geom.*`, `javax.swing.*`
- See `MyGraphicsDemo`.

General graphics applications

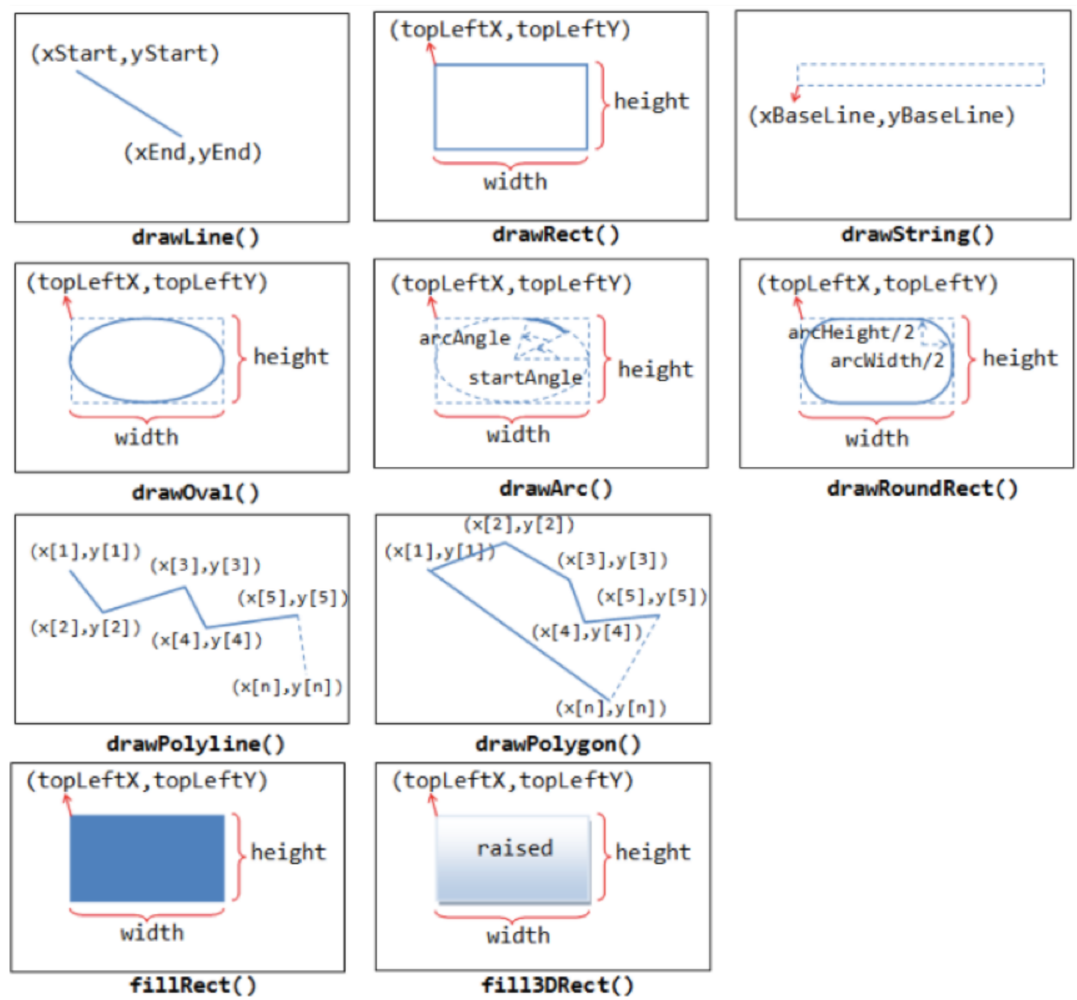
- Create an extension of component (`JPanel` or `JFrame`) and implement `paint` method in subclass.
- At start of `paint()` method cast `g` to `Graphics2D`.
- Call `repaint()` every time you want the component to be redrawn.

Geometric objects

- Objects from classes `Rectangle2D.Double`, `Line2D.Double`, etc. from `java.awt.geom`
- Constructors take parameters x, y, width, height but don't draw object.
- `Rectangle2D.Double`
- `Ellipse2D.Double`
- `Arc2D.Double`
- etc.

Drawing

- `myObj.setFrame(x, y, width, height)`: moves and sets size of component
- `g2.draw(myObj)`: gives outline
- `g2.fill(myObj)`: gives filled version
- `g2.drawString("a string", x, y)`: draws string



java.awt.Color

Lecture 4: The Catch-All Java Lecture

- Packages
- JavaDoc
- Exceptions
- Assertions
- Text I/O
- Java GUIs
- Graphics
- Events

Action listeners

- Define what should be done when a user performs certain operations.
  - e.g., clicks a button, chooses a menu item, presses Enter, etc.
- The application should implement the [ActionListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionListener.html) interface.
- An instance of the application should be registered as a listener on one or more components.
- Implement the `actionPerformed` method.

```
public class MultiButtonApp implements ActionListener {
    ...
        //where initialization occurs:
        button1.addActionListener(this);
        button2.addActionListener(this);

    ...
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == button1){
             //do something
        }
    }
}
```

https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html

Mouse listeners

- Define what should be done when a user enters a component, presses or releases one of the mouse buttons.
- The application should implement the [MouseListener](#) interface
  - Implement methods `mousePressed`, `mouseReleased`, mouseEntered, mouseExited, and `mouseClicked`.
- **Or** extend the [MouseAdapter](#) class
  - Which has default implementations of all of them.

```
public class MouseEventDemo ... implements MouseListener {
        //where initialization occurs:
        //Register for mouse events on blankArea and the panel.
        blankArea.addMouseListener(this);
        addMouseListener(this);
    ...

    public void mousePressed(MouseEvent e) {
       saySomething("Mouse pressed; # of clicks: "
                    + e.getClickCount(), e);
    }
```

https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html

Lecture 4: The Catch-All Java Lecture

- Packages
- JavaDoc
- Exceptions
- Assertions
- Text I/O
- Java GUIs
- Graphics
- Events

Readings:

- Oracle's guides:
  - JavaDoc: https://www.oracle.com/technetwork/articles/java/index-137868.html
  - Exceptions: https://docs.oracle.com/javase/tutorial/essential/exceptions/
  - Assertions: https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html
  - I/O: https://docs.oracle.com/javase/tutorial/essential/io
  - Writing Event Listeners: https://docs.oracle.com/javase/tutorial/uiswing/events/index.html
- Java Graphics: https://github.com/pomonacs622021fa/Handouts/blob/master/graphics.md
- Programming with GUIs: https://github.com/pomonacs622021fa/Handouts/blob/main/JavaGUI.pdf
- Swing/GUI Cheat Sheet: https://github.com/pomonacs622021fa/Handouts/blob/master/swing.md
- Textbook:
  - Chapter 1.2 (Page 107)