Odds and Ends

• (!) Wed night dinner, Monroe 5:30
• Acknowledge assistance as appropriate
Review Questions

• What is an algorithm and how does it relate to a method?
• What are the two questions about execution time?
• What three properties must a value possess to be used in proof of termination?
• What two sections must be present in recursive algorithm?
Zeno's Paradox
Chapter 4

Execution-Time Measurement
Algorithmic Analysis
Execution Time
Benchmarking
Analysis of Algorithms

- Want to compare algorithms apart from processor, language, compiler, etc
- Algorithmic (asymptotic) analysis
  - “gain power by losing precision”
  - Ignore constant amounts and focus on “size” of problem
- Big-Oh Notation
  - If $f(n)$ represents the actual execution time of an algorithm, the algorithm is $O(g(n))$ if, for all values $n$ larger than a fixed $n_0$, there exists a constant $c$, such that $f(n)$ is always bounded by the quantity $c \times g(n)$. 
**Execution Time**

- **Constant Time**
  - Basic operations take “constant time”
  - Also sequences of such operations
  - Also conditionals if:
    - Test is basic and statements in branches are all basic

```java
public class Card {
 ...

    public Color color() {
        if ((suit == diamond) || (suit == heart))
            return Color.red;
        else
            return Color.black;
    }
}
```
Execution Time

- **Simple Loops**
  - Even loops on fixed limits are constant time (but rare)
  - Often, loop based on input values
  - Find time of one iteration multiplied by total iterations

```java
public double minimum(double[] vals) {
    int n = vals.length;
    double minValue = vals[0];

    for (int i=1; i<n; i++){
        if (vals[i] < minValue)
            minValue = vals[i];
    }
    return minValue;
}
```
Execution Time

- $O(n)$ for this algorithm
  - $n-1$ times through loop,
  - each of which is constant

```java
public double minimum(double[] vals) {
    int n = vals.length;
    double minValue = vals[0];

    for (int i=1; i<n; i++){
        if (vals[i] < minValue)
            minValue = vals[i];
    }
    return minValue;
}
```
Execution Time

- Non-standard loop termination conditions
  - Algorithm to determine if argument is prime
  - Only check possible factors up to square-root of n
  - Worst case, we check all the number up to square-root
  - In this case, algorithm is $O(n^{1/2})$

```java
public boolean isPrime(int n) {
    // n must be >= 2
    for (int i = 2; i*i <= n; i++) {
        if (0 == n % i) return false;
    }
    return true;
}
```
Execution Time

• Nested Loops
  – How to determine execution time of nested loop?
  – How do we determine time for simple loop?
Execution Time

- Nested Loops
  - Multiply time of loop by number of iterations
  - Simple independent loop limits: multiply limits

```c
void matprod(float[][] a, float[][] b, float[][] c){
    int n = a.length;
    for(int i=0; i < n; i++){
        for (int j=0; j < n; j++){
            c[i][j] = 0.0;
            for(int k=0; k < n; k++){
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```
Execution Time

- Nested Loops with interacting limits

```java
void bubbleSort(double[] v){
    int n = v.length;
    for(int i=n-1; i>0; i--){  // move large to top
        for(int j=0; j< i; j++){
            if (v[j] > v[j+1]){  // non-increasing
                double temp = v[j];
                v[j] = v[j+1];
                v[j+1] = temp;
            }
        }
    }
}
```
Execution Time

- **Analysis of BubbleSort**
  - How many times does the \( j \) loop get executed?
  - Mathematical induction to the rescue (again)
  - Show sum of numbers 1 to \( n \) is \( n(n+1)/2 \)
  - Thus, algorithm is \( (n^2+n)/2 \), or \( O(n^2) \)

```java
void bubbleSort(double[] v){
    int n = v.length;
    for(int i=n-1; i>0; i--){ // more large to top
        for(int j=0; j< i; j++){
            if (v[j] > v[j+1]){ // non-increasing
                double temp = v[j];
                v[j] = v[j+1];
                v[j+1] = temp;
            }
        }
    }
}
```
Execution Time

• While Loops
  - Determine number of iterations through loop

```java
void insertionSort(double[] v){
    int n = v.length;
    for (int i=1; i<n; i++){
        // move element v[i] into place
        double element = v[i];
        int j = i-1;
        while (j >= 0 && element < v[j]){
            v[j+1] = v[j]; // slide old value up
            j = j-1;
        }
        v[j+1] = element;
    }
}
```
More complicated While Loops

```java
int binarySearch(double[] data, double target) {
    int low = 0;
    int high = data.length;
    while (low < high) {
        int mid = (low + high) / 2;
        if (data[mid] < target) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return low;
}
```
Summing Execution Times

• When adding component complexities, the largest component dominates
  \[ O(n^2 + n) = O(n^2) \]

• Example of the raindrop with surface tension proportional to \( n^2 \) and mass proportional to \( n^3 \)
  \[ \text{[see figures from textbook]} \]
Case history: Analysis of DNA sequences
- Looking for repeated subsequences of m tokens within sequence of n tokens
- Start with $O(m*n^2)$ algorithm
- Reformulate: n by m matrix, sort by m-length row
- $O(m*n \log n)$
- Third try: hash the m-length sequence, store sequence in hash table (duplicates will be collisions)
- $O(m*n)$
- Reduce several days down to part of a day
Benchmarking Execution Times

- Sometimes, want to compare “equivalent” algorithms
  - Bubble sort and Insertion sort both $O(n^2)$
  - (sometimes ignored constants can give clues)
- In such cases, compare run-time performance
- Introducing: GraphMaker
  - “jds.utils” provides graph making utility
  - GraphMaker requires maximum x and y values
  - Data is plotted via Reporter, a helper class
- And: TaskTimer
  - Create classes that extend TaskTimer
  - Write your own initialize and doTask methods
public class TaskTimer{
    private Reporter oput;
    public TaskTimer(Reporter iout){oput=iout;}
    public void initialize(int i){} // must override
    public void doTask(int i){} // must override
    public void run(int start, int stop, int step){
        for(int i=start; i<=stop; i+=step){
            initialize(i);
            System.gc();
            long startTime = System.currentTimeMillis();
            doTask(i);
            long stopTime = System.currentTimeMillis();
            int time = (int)(stopTime-startTime);
            oput.addPoint(i, time);
        }
    }
}

TaskTimer
class BubbleTime extends TaskTimer {
    private double[] data;

    BubbleTime(Reporter out) { super(out); }

    public initialize(int n) {
        data = new double[n];
        for (int i = 0; i < n; i++)
            data[i] = n * Math.random();
    }

    public void doTask(int n) { bubbleSort(data); }

    private void bubbleSort(double[] v) {
    }
}
public class BubbleExperiment extends Frame {
    private GraphMaker gm = new GraphMaker(500,300);

    public BubbleExperiment(){
        setTitle("Bubble & Insertion Sort Comparison");
        setSize(50,30);
        add("Center", gm);
    }

    public void run(){
        BubbleTime bt=new BubbleTime(gm.getReporter("Bubble"));
        bt.run(0,500,100);
        InsertionTime it = new InsertionTime(gm.getRepo.....);
        it.run(0,500,100);
    }

    // continued
public static void main(String[] myArgs){
    BubbleExperiment world = new BubbleExperiment();
    world.show();
    world.run();
}
} // end of BubbleExperiment Class
The Java Big Bang

- Recall that Java program is community or universe of interacting objects. How does it get going?
- The method “main” must be declared public and static
  - Static methods can run apart from instances
  - Often creates the first object (which in turn creates others, and so on) and then kicks things over
  - Sets universe in motion