This Week’s Topics

- **Reading:**
  - Sedgewick & Wayne: through Section 2.2
- **Topics**
  - Arrays & Matrices
  - Basic Collections:
    - Array Lists: the expandable array
    - Sets: the list of distinct elements
- **Acknowledgements**
  - Slides from Sedgewick & Wayne

Tips for Excelling in CompSci 100e

- Read the Book
- Ask questions
- Keep working until it is correct
- Seek help when stuck
- Visit the professor, TA, and UTAs
- Start early!
- Get the easy points

N-Body Simulation

- **Applications to astrophysics.**
  - Orbits of solar system bodies.
  - Stellar dynamics at the galactic center.
  - Stellar dynamics in a globular cluster.
  - Stellar dynamics during the collision of two galaxies.
  - Formation of structure in the universe.
  - Dynamics of galaxies during cluster formation.

N-Body Simulation

1. Setup initial distribution of particles.
   - Need accurate data and model of mass distribution.
2. Compute forces between particles.
   - Direct sum: \( N^2 \).
   - Appel / Barnes-Hut: \( N \log N \).
3. Evolve particles using ODE solver.
   - Leapfrog method balances efficiency and accuracy.
4. Display and analyze results.
   - \( \frac{dX_i}{dt} = V_i \)
   - \( \frac{dV_i}{dt} = F_i \)
Functions (Static Methods)
- **Java function.**
  - Takes zero or more input arguments.
  - Returns one output value.

- **Applications.**
  - Scientists use mathematical functions to calculate formulas.
  - Programmers use functions to build modular programs.
  - You use functions for both.

- **Examples.**
  - Built-in functions: Math.random(), Math.abs(), Integer.parseInt().
  - Our I/O libraries StdDraw.show(), StdAudio.play().
  - User-defined functions: main().

Libraries
- **Library.** A module whose methods are primarily intended for use by many other programs.

- **Client.** Program that calls a library.

- **API.** Contract between client and implementation.

- **Implementation.** Program that implements the methods in an API.

Anatomy of a Java Function
- **Java functions.** Easy to write your own.

```java
public static double sqrt (double c)
{
    if (c < 0) return Double.NaN;
    double err = 1e-15;
    double t = c;
    while (Math.abs(t - c/t) > err * t)
    t = (c/t + t) / 2.0;
    return t;
}
```

Problem 1: Gambler’s Ruin
- Gambler starts with $stake and places $1 fair bets until going broke or reaching $goal.
  - What are the chances of winning?
  - How many bets will it take?

- **One approach.** Monte Carlo simulation.
  - Flip digital coins and see what happens.
    - Pseudorandom number generation
      - java.util.Random
  - Repeat and compute statistics.
Problem 2: Self-Avoiding Walk

- **Model.**
  - N-by-N lattice.
  - Start in the middle.
  - Randomly move to a neighboring intersection avoiding all previous intersections.

- **Applications.** Polymers, statistical mechanics, etc.

- **Q.** What fraction of time will you escape in an 5-by-5 lattice?
- **Q.** In an N-by-N lattice?
- **Q.** In an N-by-N-by-N lattice?

Two Dimensional Arrays in Java

- **Array access.** Use \(a[i][j]\) to access element in row \(i\) and column \(j\).
- **Zero-based indexing.** Row and column indices start at 0.

```java
int r = 10;
int c = 3;
double[][] a = new double[r][c];
for (int i = 0; i < r; i++) {
    for (int j = 0; j < c; j++) {
        a[i][j] = 0.0;
    }
}
```

Setting 2D Array Values at Compile Time

- **Initialize 2D array by listing values.**

```java
double[][] a = {
    {0.02, 0.92, 0.02, 0.02, 0.02},
    {0.02, 0.32, 0.32, 0.32, 0.32},
    {0.02, 0.02, 0.92, 0.02},
    {0.92, 0.02, 0.02, 0.02},
    {0.47, 0.02, 0.47, 0.02, 0.02}
};
```
Problem 3: Data processing

- Scan a large (~ $10^7$ bytes) file
- Print the words together with counts of how often they occur
- Need more specification?

- How do you do it?

- What is we only wanted the top $k$ (say 20) words?

Possible solutions

1. Use heavy duty data structures (Knuth)
   - Hash tries implementation
   - Randomized placement
   - Lots o’ pointers
   - Several pages
2. UNIX shell script (Doug McIroy)
   ```bash
   tr -cs ":[alpha:]" "\[\n*]" < FILE | \
   sort | \
   uniq -c | \
   sort -n -r -k 1,1
   ```
   See `SimpleWordCount.java`
   - Which is better?
     - K.I.S.?

What can you put into an ArrayList?

- Any Object
  ```java
  Scanner in = ...;
  ArrayList<String> list = new ArrayList<String>();
  while (in.hasNext())
    list.add(in.next());
  ```
- Use a wrapper class (see java.lang.*)
  - int, double, char, boolean,...
    - Integer, Double, Character, Boolean,
- Can have your cake and eat it too
  ```java
  ArrayList<Integer> list = new ArrayList<Integer>();
  for (int k = 0; k < 10; k++)
    list.add(k*k);
  ```
- All made practical by Version 5 of Java

Exploring ArrayLists

- Look at the Java 6 API
- Note interfaces implemented
  - Serializable, Cloneable, Iterable
  - Collection, List, RandomAccess
- Note other descriptive text
  - Regarding performance
  - Constructors and other methods
- Commonly-used methods
  - boolean add(E e)  // append
  - void add(int index, E element)  // insert
  - void clear()
  - boolean contains(Object elem)
  - E get(int index)
  - int indexOf(Object elem)
  - E remove(int index)
  - E set(int index, E elem)  // replace
  - int size()
Amortization: Expanding ArrayLists

- Expand capacity of list when `add()` called
- Calling `add` N times, doubling capacity as needed

<table>
<thead>
<tr>
<th>Item #</th>
<th>Resizing cost</th>
<th>Cumulative cost</th>
<th>Resizing Cost per item</th>
<th>Capacity After add</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3-4</td>
<td>4</td>
<td>6</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>5-8</td>
<td>8</td>
<td>14</td>
<td>1.75</td>
<td>8</td>
</tr>
</tbody>
</table>

2^{m+1} - 2^m+1

What if we grow size by one each time?

Sets

- Set is an unordered list of items
  - Items are unique! Only one copy of each item in set!
- We will use two different implementations of sets
  - TreeSet
    - A TreeSet is backed up by a tree structure (future topic)
    - Keeps items sorted (+)
    - Slower than HashSets ?? (-)
  - HashSet
    - A HashSet is backed up by a hashing scheme (future topic)
    - Items not sorted – should seem to be in random order (-)
    - Faster than TreeSets ?? (+)