I is for …

- **Interface**
  - LinkedList implements List
- **Inheritance**
  - LinkedList extends AbstractSequentialList

---

### Announcements

- **Assignment P2** out today, due Thur. Feb 13
  - Get it done early, great practice for exam
- **APT-3** due yesterday
  - Last chance to turn in today til 11:59pm
- **Discussion 5** on Feb 10
  - Prepare for exam
- **Exam** next week, Feb 14

---

### PFOWBE

- **Big-Oh and O-Notation**
  - Building a mathematical formalism with intuition
- **Interfaces: List, Set, and Map**
  - When it makes sense to use general type
  - Empirical and Analytical measures of efficiency
- **Maps: API and Problem Solving**
  - Keys and Values
Review ListSplicer.java, removeFirst

- [https://coursework.cs.duke.edu/201spring20/classcode/](https://coursework.cs.duke.edu/201spring20/classcode/)
- Declarations – using List<> interface
  ```java
  List<String> linked = new LinkedList<>();
  List<String> array = new ArrayList<>();
  ```
- Method removeFirst, parameter list
  ```java
  public double removeFirst(List<String> list) {
  ```
- Method removeFirst pass either list
  ```java
  double ltime = splicer.removeFirst(linked);
  double atime = splicer.removeFirst(array);
  ```

list.remove(0)

- list is LinkedList or ArrayList, call List<> methods
  ```java
  @Override
  public double removeFirst(List<String> list) {
    double start = System.nanoTime();
    while (list.size() != 1) {
      list.remove(index: 0);
    }
    double end = System.nanoTime();
    return (end - start) / 1e9;
  }
  ```
- If list is ArrayList – call remove for ArrayList
- If list is LinkedList, call remove for LinkedList

What is “faster”? LinkedList or ArrayList

Access all elements randomly

- What is “faster”? LinkedList or ArrayList
Measuring Efficiency

- Which is faster, LinkedList or ArrayList?
  - What does it depend on?
  - Empirically: depend on computer used?
- ArrayList.remove(0):
  - \( y = 0.0064x^2 - 0.0156x + 0.0238 \)
  - \( R^2 = 0.9984 \)

Random Access Efficiency

- Random Access for Lists
  - \( \text{alist.get(N/2)} \) versus \( \text{llist.get(N/2)} \)
  - Does this depend on \( N \)?
- LinkedList random access of \( n \) elements \( n \) times
  - \( y = 0.0129x^2 - 0.7137x + 1.3337 \)
  - \( R^2 = 0.9889 \)
- ArrayList random access of \( n \) elements \( n \) times
  - \( y = -0.0002x + 5 \times 10^{-5}, R^2 = 0.8169 \)

Big-Oh aka O-Notation

- Intuition: behavior in the limit matters
  - What happens as \( N \) gets large, where we measure performance in terms of \( N \)
  - For polynomials: leading term, no coefficients
    - \( y = 3x \quad y = 6x - 2 \quad y = 15x + 44 \)
    - \( y = x^2 \quad y = x^2 - 6x + 9 \quad y = 3x^2 + 4x \)
  - The first family is \( O(n) \), the second is \( O(n^2) \)

More on O-Notation

- Provides theoretical analysis. Independent of, and can obscure some, empirical details
  - Compare: \( 20N \) hours \( v \) \( N^2 \) microseconds
  - Which is better? Does it depend?
- If an algorithm is \( O(N) \) it’s also \( O(N^2) \) from a technical, mathematical perspective
  - \( O \) is an upper bound, in the limit
  - We try to provide tight, or best bounds/analysis
Big-Oh for Algorithms

- **Binary search:** guess number 1-1024, hi, lo, correct
  - # of guesses? \( O(\log N) \) note \( 2^{10} = 1024 \)
  - If 12 seconds for \( 2^{10} \) then 24 seconds for \( 2^{20} \)

- **Sequential/linear search:** every element of list
  - # elements examined? \( O(N) \)
  - If 12 seconds for \( 2^{10} \) then 24 seconds for \( 2^{11} \)
  - Double input, double time

Big-Oh for More Algorithms

- **Efficient sorting:** merge, quick, Tim
  - # elements examined? \( O(N \log N) \)
  - More time than linear, but not terrible

- **Looking at every pair, or slow sorting, e.g., bubble**
  - # elements examined? \( O(N^2) \)
  - 12 seconds for \( 2^{10} \) then 144 seconds for \( 2^{11} \)
  - Double the input, square the time

Running times in seconds

<table>
<thead>
<tr>
<th>machine: ( 10^9 ) instructions/sec</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>( N )</th>
<th>( O(\log N) )</th>
<th>( O(N) )</th>
<th>( O(N \log N) )</th>
<th>( O(N^2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3E-9</td>
<td>1E-8</td>
<td>3.3E-8</td>
<td>0.0000001</td>
</tr>
<tr>
<td>100</td>
<td>7E-9</td>
<td>1E-7</td>
<td>6.64E-7</td>
<td>0.0001</td>
</tr>
<tr>
<td>1,000</td>
<td>1E-8</td>
<td>1E-6</td>
<td>0.00001</td>
<td>0.001</td>
</tr>
<tr>
<td>10,000</td>
<td>1.3E-8</td>
<td>0.00001</td>
<td>0.0001329</td>
<td>0.102</td>
</tr>
<tr>
<td>100,000</td>
<td>1.7E-8</td>
<td>0.0001</td>
<td>0.001661</td>
<td></td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.0000002</td>
<td>0.001</td>
<td>0.0199</td>
<td></td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.00000003</td>
<td>1.002</td>
<td>65.8</td>
<td></td>
</tr>
</tbody>
</table>
Running times in seconds
machine: $10^9$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3E-9</td>
<td>1E-8</td>
<td>3.3E-8</td>
<td>0.0000001</td>
</tr>
<tr>
<td>100</td>
<td>7E-9</td>
<td>1E-7</td>
<td>6.64E-7</td>
<td>0.0001</td>
</tr>
<tr>
<td>1,000</td>
<td>1E-8</td>
<td>1E-6</td>
<td>0.0001</td>
<td>0.001</td>
</tr>
<tr>
<td>10,000</td>
<td>1.3E-8</td>
<td>0.00001</td>
<td>0.0001329</td>
<td>0.102</td>
</tr>
<tr>
<td>100,000</td>
<td>1.7E-8</td>
<td>0.0001</td>
<td>0.001661</td>
<td>10.008</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.00000002</td>
<td>0.001</td>
<td>0.0199</td>
<td>16.7 min</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.00000003</td>
<td>1.002</td>
<td>65.8</td>
<td>?</td>
</tr>
</tbody>
</table>

Running times in seconds
machine: $10^9$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3E-9</td>
<td>1E-8</td>
<td>3.3E-8</td>
<td>0.0000001</td>
</tr>
<tr>
<td>100</td>
<td>7E-9</td>
<td>1E-7</td>
<td>6.64E-7</td>
<td>0.0001</td>
</tr>
<tr>
<td>1,000</td>
<td>1E-8</td>
<td>1E-6</td>
<td>0.0001</td>
<td>0.001</td>
</tr>
<tr>
<td>10,000</td>
<td>1.3E-8</td>
<td>0.00001</td>
<td>0.0001329</td>
<td>0.102</td>
</tr>
<tr>
<td>100,000</td>
<td>1.7E-8</td>
<td>0.0001</td>
<td>0.001661</td>
<td>10.008</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.00000002</td>
<td>0.001</td>
<td>0.0199</td>
<td>16.7 min</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.00000003</td>
<td>1.002</td>
<td>65.8</td>
<td>31.8 years</td>
</tr>
</tbody>
</table>

What do they all have in common?

• They all took a CompSci course at Duke!

WOTO

What do they all have in common?

• They all took a CompSci course at Duke!

ArrayList Methods

```java
ArrayList<String> words = new ArrayList<>();
words.add("cat");
words.add("fish");
words.add("dog");
String b = words.get(1);
words.set(2, "frog");
int c = words.indexOf("cat");
words.set(1, words.get(c));
```

Words is [ ]
ArrayList Methods

```
ArrayList<String> words = new ArrayList<>();
words.add("cat");
words.add("fish");
words.add("dog");
String b = words.get(1);
words.set(2, "frog");
int c = words.indexOf("cat");
words.set(1, words.get(c));
```

Words is [ cat, fish, dog]

b is fish
ArrayList Methods

ArrayList<String> words = new ArrayList<>();
words.add("cat");
words.add("fish");
words.add("dog");
String b = words.get(1);
words.set(2, "frog");
int c = words.indexOf("cat");
words.set(1, words.get(c));

Words is [ cat, fish, frog]
b is fish
c is 0

Problems and Solutions

• String that occurs most in a list of strings?
  • CountingStringsBenchmark.java, two ideas
    • See also CountingStringsFile for same ideas
    • https://coursework.cs.duke.edu/201spring20/classcode
  • Parallel arrays: word[k] occurs count[k] times
  • Use ArrayLists: 2 “the”, 3 “fat”, 4 “fox”

<table>
<thead>
<tr>
<th>the</th>
<th>fox</th>
<th>cried</th>
<th>fat</th>
<th>tears</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
How does the code work?

- Process each string $s$
  - First time \texttt{words.add(s), counter.add(1)}
  - Otherwise, increment count corresponding to $s$
  - $c[x] += 1$ ?

```java
public static String parallelArrays(List<String> list) {
    ArrayList<String> words = new ArrayList<>();
    ArrayList<Integer> counter = new ArrayList<>();
    for (String w : list) {
        int index = words.indexOf(w);
        if (index == -1) {
            words.add(w);
            counter.add(1);
        } else {
            counter.set(index, counter.get(index) + 1);
        }
    }
}
```

What is complexity of this code?

- Search for each word and … if occurs at $k$
  - +1 to counter.get($k$), else add at end

- Search complexity? $O(M)$ when $M$ different words
  - One search is $O(M)$ – what about all searches?
  - Tracking all words. First time zero, then one, …
  - Avoid analyzing duplicates for the moment
    - Will take longer if we have multiple occurrences of some of $M$ words

Tracking N strings

- Complexity of search? $O(M)$ for $M$ different words
  - Complexity of \texttt{words.indexOf(…)} is $O(M)$
  - what about all calls? $1 + 2 + \ldots + N$ is $N(N+1)/2$

```java
public static String parallelArrays(List<String> list) {
    ArrayList<String> words = new ArrayList<>();
    ArrayList<Integer> counter = new ArrayList<>();
    for (String w : list) {
        int index = words.indexOf(w);
        if (index == -1) {
            words.add(w);
            counter.add(1);
        } else {
            counter.set(index, counter.get(index) + 1);
        }
    }
}
```

Should we be more precise?

- Adding $M$ different words will be $O(M^2)$
  - $1 + 2 + \ldots + M = M(M+1)/2$

- Adding duplicates: we need to be precise about adding $N$ total words.
  - Sometimes word will be found, still $O(M)$ for $M$ different words
  - We have both $M$ and $N$ here, but treat $M == N$ for easier analysis.
CountingStringsFile.java

- Generate an ArrayList of Strings
  - Find the word that occurs the most often
  - See three different methods

Understanding O-notation

- This is an upper bound and in the limit
  - Coefficients don’t matter, order of growth
  - \( N + N + N + N = 4N \) is \( O(N) \) --- why?
  - \( N\times N \) is \( O(N^2) \) – why?
  - \( O(1) \) means independent of \( N \), constant time

- In analyzing code and code fragments
  - Account for each statement
  - How many times is each statement executed?

<table>
<thead>
<tr>
<th>( N )</th>
<th>( 20N )</th>
<th>( 100N )</th>
<th>( 2000N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>200</td>
<td>1000</td>
<td>20,000</td>
</tr>
<tr>
<td>10,000</td>
<td>200,000</td>
<td>1,000,000</td>
<td>20,000,000</td>
</tr>
<tr>
<td>10,000,000</td>
<td>200,000,000</td>
<td>1,000,000,000</td>
<td>20,000,000,000</td>
</tr>
<tr>
<td>10,000,000,000</td>
<td>200,000,000,000</td>
<td>1,000,000,000,000</td>
<td>20,000,000,000,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( N )</th>
<th>( 20N )</th>
<th>( 100N )</th>
<th>( 2000N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>200</td>
<td>1000</td>
<td>20,000</td>
</tr>
<tr>
<td>10,000</td>
<td>200,000</td>
<td>1,000,000</td>
<td>20,000,000</td>
</tr>
<tr>
<td>10,000,000</td>
<td>200,000,000</td>
<td>1,000,000,000</td>
<td>20,000,000,000</td>
</tr>
<tr>
<td>10,000,000,000</td>
<td>200,000,000,000</td>
<td>1,000,000,000,000</td>
<td>20,000,000,000,000</td>
</tr>
<tr>
<td>10,000,000,000,000</td>
<td>(20)*10,000,000,000,000</td>
<td>(100)*10,000,000,000,000</td>
<td>(2000)*10,000,000,000,000</td>
</tr>
</tbody>
</table>
Just Say No.. When you can

\[ O(n^2) \]

Example: Analyze using big-Oh

- What is runtime of `stuff(N)`
  - How to reason about this
- What is return value of `stuff(N)`
  - What if code changes to `sum += k`

```java
public int stuff(int n) {
    int sum = 0;
    for(int k=0; k < n; k += 1) {
        sum += n;
    }
    return sum;
}
```

Counting for O-notation

- Why is O(1) complexity of `sum += n`
  - Is this O(1) for any `x += y`?
  - Loop executes N times, doing O(1) per iteration
    - Total runtime for method? O(n)

Example 2: Analyzing O-Notation

- What is big-Oh of runtime of call `calc(N)`?
  - Num. of statements executed, O(1) for line 146?
  - Use `calc(16)` and generalize
- What is big-Oh of value returned by `calc(N)`?
  - Table? k = 1, 2, 4, 8, 16, 32, 64

```java
public int calc(int n) {
    int sum = 0;
    for(int k=1; k < n; k *= 2) {
        sum += k;
    }
    return sum;
}
```
WOTO