AP CS A and AB: New/Experienced
A Tall Order?

Mark Stehlik
mjs@cs.cmu.edu
http://www.cs.cmu.edu/~mjs
Schedule

- **8:30 – 10:00**
  - Intro, resources, what’s new

- **10:15 - 12:00**
  - Design
  - Inheritance, Interfaces & Abstract Classes

- **12:45 – 3:30**
  - Collections -> Analysis -> Big O
  - Reading sample questions and their rubrics
Intro

- A v. AB?
- Years teaching?
- Java experience?
- The cards (years teaching A/AB, 2 things)
- Materials (AB q/ref, A2 & AB1 samples, role play)
Resources

• AP Central
  ➢ Course descriptions
  ➢ Java subset
  ➢ Sample syllabi

• AP list-serve

• JETT workshops
  ➢ 8/6 & 7 at Florida International
What’s new?

● 2-D arrays to AB

● Well, Java...
  ➢ No reference parameters
  ➢ Collections

● But also...
  ➢ Design
  ➢ Analysis
  ➢ Priority Queues
Design

• Interfaces – proscribe a set of behaviors
  ➢ methods are public by default
  ➢ **NO** implementation can be provided
  ➢ **NO** instance variables can be declared
  ➢ cannot, therefore, be instantiated
    • what does that mean?
  ➢ examples: stack and queue

• Classes realize interfaces by implementing all methods
Design…

- vs. Abstract classes
  - some methods implemented
  - others designated as abstract (which forces the class to be abstract) – these must be implemented by class(es) that extend this abstract class (ultimately)
  - can have instance variables
Design…

- Let’s look at A2

- Let’s look at some other classes
Collections

- Lists, Sets, Maps

- Raises the level of design (and discourse)

- But, then, what is (are) the issue(s)?
Collections…

• **What do they do?**
  ➢ List
    • ordered (positionally) – a sequence
    • duplicates?
  ➢ Set
    • unordered collection
    • duplicates?
  ➢ Map
    • Maps (unique) keys to values
Collections…

- **How do they do it?**
  - Analyze their algorithms
- But what does that mean?
Algorithm Analysis (kudos to ola)

- How do you measure performance?
  - It’s faster! It’s more elegant! It’s safer! It’s cooler!

- Use mathematics to analyze the algorithm (performance as function of input size)

- Implementation is another matter
  - cache, compiler optimizations, OS, memory,…
What do we need?

- Need empirical tests and mathematical tools
  - Compare by running
    - 30 seconds vs. 3 seconds,
    - 5 hours vs. 2 minutes
    - Two weeks to implement code

- We need a vocabulary to discuss tradeoffs
What is big-Oh about?

- Intuition: avoid details when they don’t matter, and they don’t matter when input size (N) is big enough
  - For polynomials, use only leading term, ignore coefficients: linear, quadratic

\[
\begin{align*}
  y &= 3x \\
  y &= 6x - 2 \\
  y &= 15x + 44 \\
  y &= x^2 \\
  y &= x^2 - 6x + 9 \\
  y &= 3x^2 + 4x
\end{align*}
\]
O-notation, family of functions

- first family is $O(n)$, the second is $O(n^2)$
  - Intuition: family of curves, same shape
  - More formally: $O(f(n))$ is an upper-bound, when $n$ is large enough the expression $cf(n)$ is larger
  - Intuition: linear function: double input, double time, quadratic function: double input, quadruple the time
Reasoning about algorithms

- **We have an O(n) algorithm,**
  - For 5,000 elements takes 3.2 seconds
  - For 10,000 elements takes 6.4 seconds
  - For 15,000 elements takes ....?

- **We have an O(n²) algorithm**
  - For 5,000 elements takes 2.4 seconds
  - For 10,000 elements takes 9.6 seconds
  - For 15,000 elements takes ...?
More formal definition

- O-notation is an upper-bound, this means that \( N \) is \( O(N) \), but it is also \( O(N^2) \); we try to provide tight bounds. Formally:

  - A function \( g(N) \) is \( O(f(N)) \) if there exist constants \( c \) and \( n \) such that \( g(N) < cf(N) \) for all \( N > n \).
Other definitions

- $g(n)$ is $O(f(n))$ if $\lim g(n)/f(n) = c$ as $n \to \infty$

- Informally, think of $O$ as “$\leq$”

- Similarly, there’s a notation for lower bounds...
Big-Oh calculations from code

- **Search for element in array:**
  - What is complexity (using O-notation)?
  - If array doubles, what happens to time?

```java
for(int i=0; i < a.length; i++) {
    if (a[i].equals(target)) return true;
}
return false;
```

- **Best case? Average case? Worst case?**
Measures of complexity

- **Worst case**
  - Good upper-bound on behavior
  - Never get worse than this

- **Average case**
  - What does average mean?
  - Averaged over all inputs? Assuming uniformly distributed random data?
Some helpful mathematics

- \(1 + 2 + 3 + 4 + \ldots + N\)
  \[\frac{N(N+1)}{2} = \frac{N^2}{2} + \frac{N}{2}\text{ is } O(N^2)\]

- \(N + N + N + \ldots + N\) (total of \(N\) times)
  \[N^*N = N^2\text{ which is } O(N^2)\]

- \(1 + 2 + 4 + \ldots + 2^N\)
  \[2^{N+1} - 1 = 2 \times 2^N - 1\text{ which is } O(2^N)\]
The usual suspects

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(n^3)$
- ...
- $O(2^n)$
Multiplying and adding big-Oh

• Suppose we do a linear search then we do another one
  ➢ What is the complexity?
  ➢ If we do 100 linear searches?
  ➢ If we do \( n \) searches on an array of size \( n \)?
Multiplying and adding

- Binary search followed by linear search?
  - What are big-Oh complexities? Sum?
  - 50 binary searches? N searches?

- What is the number of elements in the list (1,2,2,3,3,3)?
  - What about (1,2,2, ..., n,n,...,n)?
  - How can we reason about this?
Analysis for AP collections

- ListQueue
- ArrayStack (where’s the top?)
- Lists, Maps, Sets
Priority Queues

- Idea

- Implementation(s)
Language details

- **No reference parameters**
  - All passes are by value (primitives/objects)

- **How do you change something**
  - Through “modifier” methods
  - Through returning a reference to a modified object (x = “changed x”)