CSE 8B discussion: week 8

Recursion, Java standard library, practice resources
Agenda

1. Recursion
2. Java standard library
3. Practice resources and sample problems
HW5 review
Recursion

- **Definition:**
  Solving a problem by successively reducing it to the same problem with smaller inputs

- **Motivation:**
  Some problems are thought of most naturally reasoned about and solved recursively
  - I swear
Classic example of recursion: factorial of an integer

- $5! = 5 \times 4 \times 3 \times 2 \times 1$
- stated recursively: $n! = n(n - 1)!$
Counting pattern occurrence

- Given finite binary string “100101101…”, count how many times “00” occurs in the string

Think about this both iteratively (i.e., with loops), and recursively
Counting pattern occurrence: solution

- Iterative: check each pair of characters, see if it matches pattern
- Recursive: ?
A little harder: merging sorted arrays

- Given sorted arrays $a = [a_1, a_2, a_3, ...]$, $b = [b_1, b_2, b_3, ...]$, produce a single sorted array that contains all elements from both $a$ and $b$.
Agenda

1. Recursion
2. Java standard library
3. Practice resources and sample problems
Java standard library

- **standard library**: set of classes/methods/etc. that a programming language comes with
- Don’t reinvent the wheel (you might mess it up), be more productive
- Lots of pre-written code that is super helpful to know exists

Everything is **HERE**
Core packages, classes, and interfaces

- `java.io` -- input/output to streams (keyboard, files, anything)
  - InputStream, OutputStream, Reader, Writer
  - Scanner
  - File

- `java.lang` -- classes “fundamental to the design of the language” *(no need to import these!)*
  - CharSequence -- things like String, StringBuilder
  - Comparable -- lets your object be comparable to others via compareTo()
  - Iterable -- makes instances of your class “for-each”-enabled!
  - Math -- static methods like abs, trig, logs, rounding, min/max,
  - Runnable, Thread -- lets your program multitask (see threads; week 9!)
  - String -- hard to build up one character at a time (see StringBuilder)
  - System -- System.in / .out, garbage collector, copy arrays(?!)}
Primitive wrapper classes

- Want a List of ints?
  - `List<int> myFavoriteInts = new ArrayList<int>();`

- **COMPILE-TIME ERROR!** LOL, too bad, data structures can only contain reference types
  - recall: what’s a reference type? Anything *capitalized*: Map, Set, Person, Ferrari

- Introducing
  the *primitive wrapper classes*

- Many have lots of convenience methods (e.g., parse a number from a String in any base!)
But I want a List of ints!

- `List<Integer> myFavoriteInts = new ArrayList<Integer>();
  for (int i = 0; i < 10; i++)
    myFavoriteInts.add(i);

- This **works**
- It’s called **autoboxing**

- `int sum = 2 + myFavoriteInts.get(3);`

- This **works**
- It’s called **unboxing**

Who’s ever heard of a snozzberry?
More goodies

- `java.math.BigInteger` and `BigDecimal` -- arbitrarily large integer and decimal numbers
- `java.time` -- classes for working with date/time
- `java.util` -- a mix of helpful utility classes
  - all of the Collections we’ve talked about (implementations of Map, List, Set, etc.)
  - Arrays -- all kinds of utility methods for arrays ([], not ArrayLists)
  - Scanner -- get formatted input from a stream (like System.in, or a file)
  - Random -- random number generation
- `java.util.concurrent` -- if you’re doing multithreaded programming
- `java.util.regex` -- Java’s regular expressions implementation
- `javax.crypto` -- encrypt and decrypt your program’s data

...and more for programming with audio, networking, etc.
Takeaway

Know thy standard library!

whatever language you work with, it helps immensely
Agenda

1. Recursion
2. Java standard library
3. Practice resources and sample problems
Resources to get better at solving problems

- A handful of websites let you write code and they test your solution against many test cases
  - [https://www.hackerrank.com/](https://www.hackerrank.com/)
  - [https://leetcode.com/](https://leetcode.com/)
- Problems ranked easy/medium/hard
- Start with easy ones
- Helps you develop a “muscle memory” in how to write Java code
  - helps avoid syntax errors, etc.
  - also exercises your ability to leverage your language’s **standard library**
  - questions are sometimes like those you see in interviews
  - speaking of which...
- Some companies even use HackerRank to screen candidates
Some real interview questions I’ve been asked

- **Fizz buzz**
- Given a List<List<Integer>> (like a matrix of integers), return a list of the sum of all the elements in each row
  - \([ \{1, 4, 6\}, \{2, 3, 5\}, \{1, 9, 8\}\] \(\Rightarrow\) \([11, 10, 18]\)
- Given a String, write a method that returns true/false whether or not the string is a palindrome (try it iteratively and recursively!)
  - isPalindrome("racecar") \(\Rightarrow\) true, isPalindrome("i love discussion") \(\Rightarrow\) false [not a palindrome ;) ]
- Given a String, return the longest substring that is a palindrome
  - longestPalindromicSubString("intheskyisufotofu") \(\Rightarrow\) "ufotofu"
- Given a String, return the first non-repeating character, case-insensitive
  - "Doggy woggy is tired" \(\Rightarrow\) ‘w’
- Given a list of numbers, shuffle all the 5s to the end, while leaving the other numbers’ relative positioning to each other unchanged
  - \([1, 6, 5, 5, 70, 39, 5, 2]\) \(\Rightarrow\) \([1, 6, 70, 39, 2, 5, 5, 5]\)
Most all problems have multiple solutions

- After you solve one, google it and see others’ solutions
- Often there is a “direct” solution, in addition to one or more more clever solutions that are more efficient
- More efficient == better Big O bound, aka better time complexity
  - time it takes the algorithm to run grows in proportion to the bound, e.g., an O(n^2) algorithm takes time that scales proportional to the square of the input size; 4n elements results in roughly 16 times the runtime, when n is large.
Agenda

1. Recursion
2. Java standard library
3. Practice resources and sample problems