CS 1332 F25 Recitation – Week 0

Course Info, IntelliJ, JUnits, Big O

ANNOUNCEMENTS

- **CLOCK IN!**
- It's the **first recitation!**
- Recitation Worksheet 0 is a quick review of course policies (with a Scavenger Hunt! 😀), JUnits/the debugger, and Big-O notation.
    - Some **Scavenger Hunt** hints will be marked with a 🧭 in this guide!
    - Some hints for other **Recitation Worksheet questions** will be marked with a 📄 in this guide!
- All worksheets, live coding files, and potential recitation practice exams will be at **Files** → **Resources** → **Recitation Materials** on Canvas
- Get to know your students, introduce yourself and be sure to write your:
    - Name
    - Email
    - Office hours (and tell them they can attend any office hours starting **tomorrow**) 🧭
- If you want, feel free to talk a bit about yourself so students can get to know you better (major, where you're from, hobbies, favorite cereal, etc.)
- ArrayList HW has been released and is due **Wednesday, August 27th at 11:59:00 pm.** 🧭

ICEBREAKERS

- Ask students to split in groups of 2
- In their group, ask them to introduce themselves to each other and tell their
    - Name
    - Year in GT
    - Major
    - Threads (if CS)
    - Which other classes are they taking this semester
    - What's an interesting fact about you?
    - What did you do this summer?
    - Plans for the school year?
    - Whatever you want!

- Stop the discussion after 5 minutes

COURSE POLICIES

You can walk students through the Canvas site that shows this information (but don't give away too many Scavenger Hunt answers 😉)!

Here are some important policies they should keep in mind regarding assignments:

- All homework will have a single due date
- Homework due dates may differ depending on the length of the assignment
- Homework is always due at **11:59:00 pm ET** for full credit consideration
- There is a grace period until **12:00:00 pm ET** the following day, but their submission will be marked as late by Gradescope, and they will be penalized 25 percentage points for the assignment
    - **Students should not ask for an exception** to the late policy unless they have a valid excuse accompanied by documentation that has been submitted to the Dean of Students office, which should be taken directly to your professor (Dr. HB or Prof. Faulkner) and not the TAs
- Homework turn-in is via Gradescope; turning in homework properly is solely their responsibility
    - If you aren't in Gradescope, you can access it through the link on the sidebar in the Canvas course → show them where this can be found on Canvas
- Submissions should include all necessary files as specified in HW pdf (nothing more and nothing less)
- They should make sure their homework compiles in Gradescope. It should alert them if it didn't
    - They should also **run Checkstyle** on their homework. They may lose up to 10 points if they don't 🧭
- Gradescope is not forgiving about due dates and times
- Gradescope always overrides previous submissions; if they are late, their earlier submission will be overridden
    - We will not grade an earlier submission (or any part of an earlier submission), if you are late or under any other circumstance; we only grade the latest submission
- All exams are **in-person and mandatory.**
    - Online students will take exams in-person during their scheduled class time (Thursdays at 5 PM)

IntelliJ

- **We recommend students to use IntelliJ for CS 1332.** 🧭
- For IntelliJ Setup refer them to Garrett's video: **bit.ly/1332intellijsetup**
- TAs can help them with all of this during office hours
- A step-by-step guide is also pinned on **Piazza** 🧭
- Some random but important tips to highlight:
    - Download IntelliJ Community Edition
    - The video says Java 8 but make sure to use the SDK for **Java 17** 🧭
    - Students can use the most up-to-date version of Checkstyle
    - Please ensure students know how to **create an IntelliJ project** and place their HW files in the **src** directory!

Please allocate 10-15 minutes for IntelliJ installation during recitation.

Visual Studio Code

- For VSCode setup you can refer students to the following guide: **bit.ly/1332VSCode**

JUnits

- Framework for writing unit tests in Java
- Unit testing lets you test pieces of your code (units) for correctness under specific conditions (e.g. do these inputs create the expected output for this function?)
- *Run each written test, even if some tests in the middle crash or fail*
- This class will use JUnits a lot:
    - The student tests we provide use JUnits
    - These tests are not comprehensive though, so it's important to write your own!

Types of JUnit Methods

@Before methods

- These methods run before **each** test is run
- Mostly used to do setup, like initializing variables and objects

@Test methods

- These methods run as actual unit tests
- Each test is independent of one another
- Each method should test your data structure under a specific scenario, such as:
    - "What if I call `addAtIndex()` for **ArrayList** when I need to resize?"

    - “What if I call `get()` for **ArrayList** at the beginning or end of the list?”
- Always try to think of edge cases and test against them!

How to Structure a JUnit Test

- Populate the data structure
    - e.g. if you’re testing `replace()` for **ArrayWrapper**, adding a few elements to **ArrayWrapper**
    - e.g. if you’re testing `addAtIndex()` for **ArrayList** on resize, adding **INITIAL_CAPACITY** elements
    - This might happen in the **@before** method if some configuration is common to all tests
- Call the appropriate method(s) you want to test in your data structure
- Construct the expected result of the method
    - e.g. an array with the replaced value if testing `replace()` for **ArrayWrapper**
    - e.g. a properly resized array if testing `addAtIndex()` on resize for ArrayList
- Compare the expected result of the method with the actual result
    - This is done with JUnit assert methods, like `assertEquals()`, `assertArrayEquals()`, `assertNull()`, etc.
    - These methods are discussed in greater detail in the “Writing JUnits” guide on Canvas
    - If the expected result doesn’t match the actual result, JUnit will report an error
    - Make sure to provide first the expected parameter and then the obtained result, otherwise you will get confusing error messages

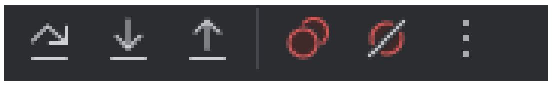
Notes on JUnits

- JUnit can’t test efficiency. You will have to determine that yourself by reasoning about your code
    - Feel free to ask TAs to check in office hours!
- Tests should run independently of each other
    - Except for `add()`, which should be tested extensively before using it to test other methods
- If you write tests, feel free to share them on the pinned post on Piazza! Make sure to host them on GT GitHub
- You can also add timeouts to your tests to handle infinite loops, which is what our tests do. Make sure to remove them when debugging through to avoid weird behavior

JUNITS + DEBUGGER ACTIVITY

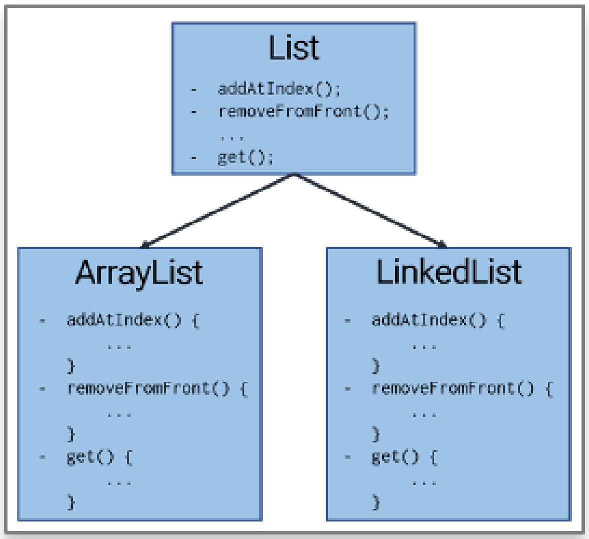
Please walk through the JUnits + debugger activity with students from Recitation Worksheet 0.

- Important pointers:
  - Start with writing a simple JUnit to test `replace()` for **ArrayWrapper**
  - Then, move on to test the *exception* in `replace()`
  - Next, **use the IntelliJ debugger** to debug `testRange()` and `testMax()`
    - Explain/demonstrate the difference between **stepping into, over, and out of a function** (icons below)



  - Emphasize that **JUnits can't determine an efficiency issue!** See the `range()` method's inefficiency for an example.
- If students want a short recap of the activity, refer them to this debugger tutorial: bit.ly/1332debuggerdemo **(NOT comprehensive)**
- There is also detailed documentation on using IntelliJ debugger on jetbrains.com

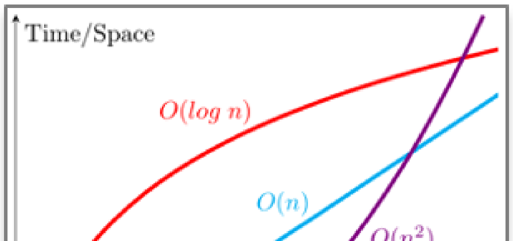
DATA STRUCTURES VS. ADT 📄



- **ADT**: Set of features and properties that define some useful objects (e.g. Lists, Stacks, Queues)
  - Analogous to a Java Interface
- **Data Structure**: Concrete implementation that typically implements an ADT (e.g. ArrayList, Singly LinkedList, Doubly LinkedList)
- We will see how these definitions apply to Lists over this and next recitation

BIG-O REVIEW

- Time complexity depicts how the running time of an algorithm/operation on a data structure grows with the size of the input 📄
  - When it comes to data structures, usually the size, e.g. the number of entries in an ArrayList, is the unit of measurement
- Similarly, space complexity is how much extra space an algorithm/operation on a data structure uses as input grows 📄
  - Not very interesting when it comes to data structures, so you'll see more of this when we study algorithms
- Essentially an upper-bound on the number of

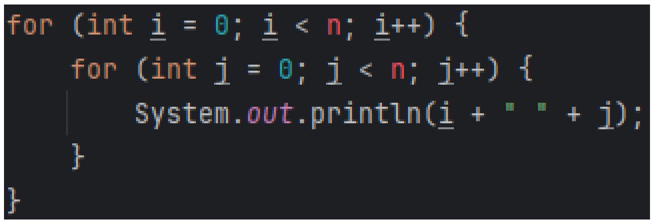


operations as input grows, ignoring constants and lower-order terms
- Clarify that constant time only means that the time taken is constant (e.g. shipping files from city A to city B might be constant, but not necessarily the fastest in practice until sending files via the internet takes longer than the boat)
- Should avoid memorizing tables of complexities; instead, understand where the complexities come from
- Important for this class, but also interviews, jobs, and research

Practice Questions!

Throughout recitations, we will integrate questions from the recitation worksheets! Please walk through the following questions from Worksheet 0 with students:

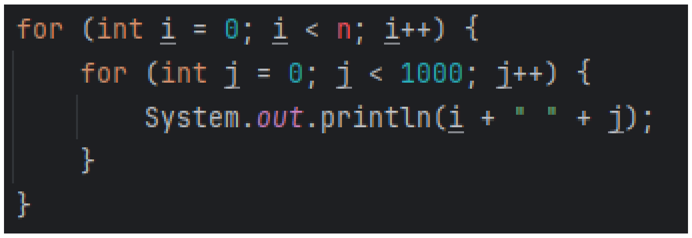
1. What is the time complexity of the following code snippet?



```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        System.out.println(i + " " + j);
    }
}
```

Solution: ***$O(n^2)$***

2. What is the time complexity of the following code snippet?



```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 1000; j++) {
        System.out.println(i + " " + j);
    }
}
```

Solution: ***$O(1000n)$*** = ***$O(n)$***

3. Discuss the difference between the two code snippets above. Why do they have different time complexities?

   Solution: The first snippet's inner loop scales with ***n***, resulting in a quadratic overall runtime complexity. The second snippet's inner loop has a fixed number of iterations, so the total complexity remains linear.

Amortization

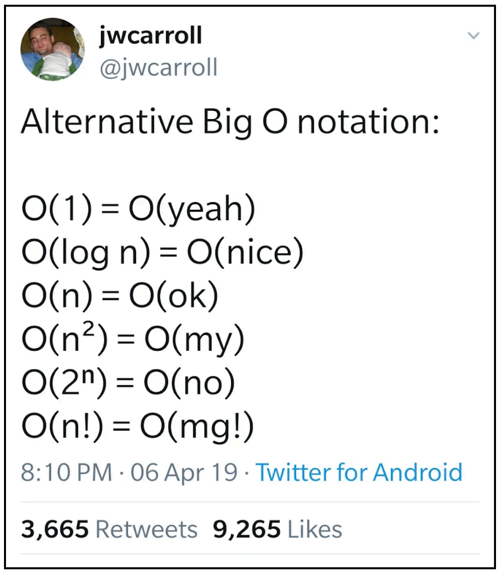
- Amortized time is used when the worst-case runtime is pessimistic (occurs very rarely) 📄
  - It is used when an expensive operation occurs rarely, and each time it occurs it becomes more rare
  - Along the lines of: "Usually this will be ***O(1)***, but every so often it will run in to allow more operations"
- Mathematically, you can argue that the operation happens infrequently enough over time that the time taken by the algorithms can almost be considered ***O(1)***
- Try not to say average or equate it with the average case
  - They are not the same, especially when we get to BSTs and sorting
  - It's more like spreading out the expensive cost over the use of the data structure
- When we ask for **un**amortized analysis, we are asking specifically for this expensive operation

SCAVENGER HUNT

- ***If time permits, go through the answers for the Scavenger Hunt.***
- Answers will be released on Canvas!

CLOCK OUT!

Meme of the week:



jwcarroll
@jwcarroll

Alternative Big O notation:

O(1) = O(yeah)
O(log n) = O(nice)
O(n) = O(ok)
O(n^2) = O(my)
O(2^n) = O(no)
O(n!) = O(mg!)

8:10 PM · 06 Apr 19 · Twitter for Android

3,665 Retweets **9,265** Likes