CSE 12: PA4

1-28-21

Focus: PA4, Runtime Analysis & Measurement

PA4 Overview

Two Parts

Part 1: Questions

- Big-O Justification
- Analysis of ArrayStringList and LinkedStringList
- 6 Mystery Functions: Determe big-Θ, measure implementations (in part 2) and match

Part 2: Code

- Write a program to measure the mystery methods
- Matches the methods to the source given
- Generate graphs to justify

Tips!

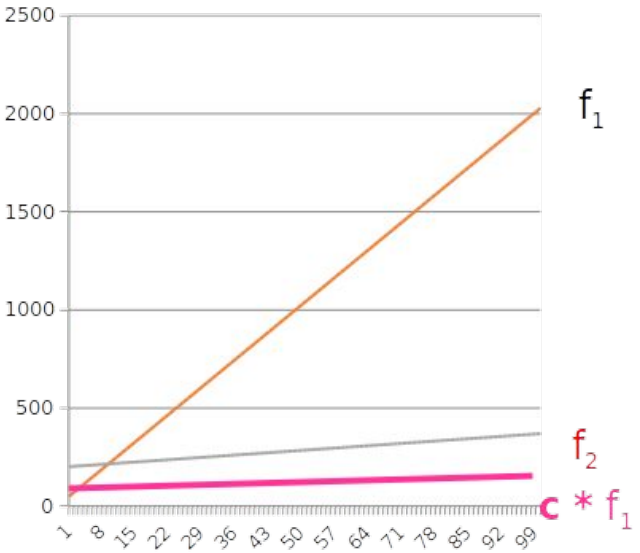
- Follow the format instructions very carefully on Gradescope!
- Only 16/70 points are autograded, don't rely on resubmission for this assignment

Categorizing Runtimes

f(n) = **O**(g(n)), if there are positive constants c and n_0 such that f(n) ≤ **c** * g(n) for all n ≥ n_0.

f(n) = **Ω**(g(n)), if there are positive constants c and n_0 such that f(n) ≥ **c** * g(n) for all n ≥ n_0.

- f_1 is $\Omega(f_2)$ because $f_1 > f_2$ (after about n=10, so we set $n_0 = 10$)
  - f_2 is clearly a ***lower bound*** on f_1 and that's what big-Ω is all about
- But f_2 is $\Omega(f_1)$ as well!
  - We just have to use the "**c**" to adjust so f_1 that it moves below f_2



Summary

Big-O

- **Upper bound** on a function
- f(n) = O(g(n)) means that we can expect f(n) will always be **under** the bound g(n)
  - But we don't count n up to some starting point n_0
  - And we can "cheat" a little bit by moving g(n) up by multiplying by some constant c

Big-Ω

- **Lower bound** on a function
- f(n) = Ω(g(n)) means that we can expect f(n) will always be **over** the bound g(n)
  - But we don't count n up to some starting point n_0
  - And we can "cheat" a little bit by moving g(n) down by multiplying by some constant c

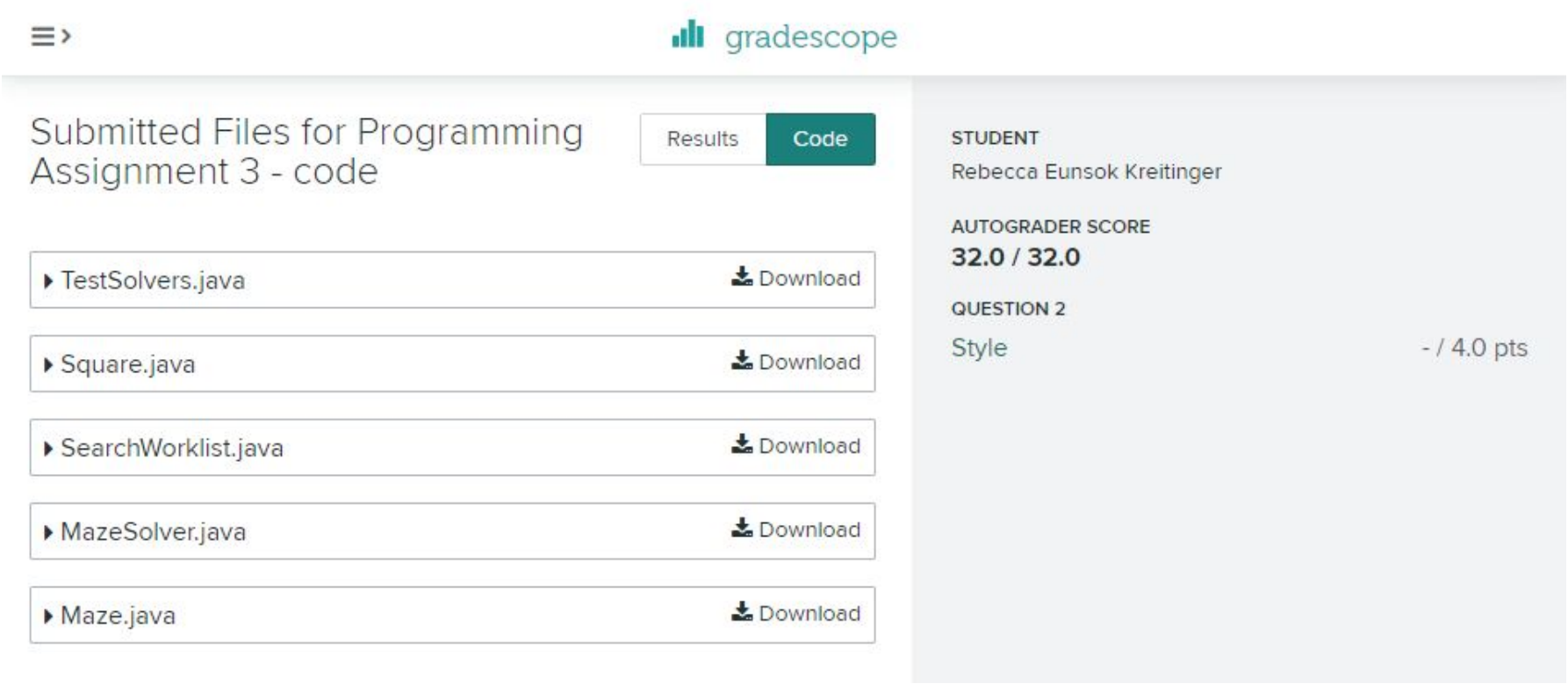
Big-θ

- **Tight bound** on a function.
- If $f(n) = O(g(n))$ *and* $f(n) = \Omega(g(n))$, then $f(n) = \theta(g(n))$.
- Basically it means that f(n) and g(n) are interchangeable
- Examples:
  - $3n+20 = \theta(10n+7)$
  - $5n^2 + 50n + 3 = \theta(5n^2 + 100)$

How to measure runtime in Java

- Remember to do two things to ensure your measurements are as accurate as possible:
  - Turn off Java compiler optimizations
  - Call each method once (dummy call) before calling them to measure their runtimes; the timing of the first call can be noisy and inaccurate
- How to turn off optimization in Eclipse (from the write up; scroll to last page):
  - https://docs.google.com/document/d/1vwckO76TrBT8B5E4xQ2-v2OXncLa6SQWuaQkNZaCPB0/edit
- How to turn off optimization in terminal
  - add in the flag in your **`javac`** and **`java`** commands. Examples:
    `java -Djava.compiler=NONE myClass`
- Example code from discussion on how to calculate runtime of a method using **`System.nanoTime()`** will be posted on the course Github

Verifying submitted file paths



Verifying submitted file paths