Practical 1: Empirical Analysis of Algorithms

Background

There used to be an attitude that still persists today that as long as your code runs and gives the correct output, nothing else matters. In the past if your code ran rather slowly the easy solution was to add more processing power. Ultimately it comes down to which solution (building a better algorithm or adding more computational resources) is the best option given the requirements of your project. If you just need to make something work for a small scale project then focusing too much on optimizing your algorithm is not worthwhile.

However, in a world where algorithms increasingly need to perform effectively with endless amounts of data or on limited hardware, the ability to measure and monitor the performance of alternative algorithmic solutions becomes a very attractive skill. And if you want to work for a top computer company (i.e. Facebook, Google, Tesla) then this is a prerequisite skill.

What am I doing today?

Today's practical focuses on 3 things:

1. Getting setup with Github Classroom
2. Empirical Analysis: timing & graphing performance of a program
3. Empirical Analysis: timing, graphing & comparing the performance of a different program

Instructions

Try all the questions. Ask for help from the demonstrators if you get stuck.

***Remember** each practical is part of your portfolio assignment so make sure to keep your work carefully, commit the code your write to your repository and record the results of your experiments in the repository too.

1.Setting up GitHub Classroom

To get started, accept the GitHub classroom invitation. This will give you access to a folder for starter code, data and helper classes.

To create your own repository for the course use this starter link:
https://classroom.github.com/a/W3uJmO3m

Once set up you should have a folder structure like this:

```
--data
--graphs
In.java
StdIn.java
StdOut.java
Stopwatch.java
```

ThreeSumA.java
ThreeSumB.java

2.Empirical Analysis: ThreeSumA

In the first exercise, the challenge is to assess how the provided algorithm performs, as measured by its running time, as the input size increases (i.e. in this case the number of integers the algorithm is handling). This is a classic algorithm problem that is often asks as part of job interviews.

The challenge is for the algorithm to solve the problem of working out how many unique integer triples in an array or input sum to 0. More formally, given an array **arr[ ]** consisting of **n** integers, find all the unique triples in the array which sum to zero.

Run the ThreeSumA algorithm on various inputs

Check the implementation is correct

1. Modify the algorithm to print out the correct triplets
2. Run the algorithm

Run from your IDE (e.g., Eclipse)

1. Create a new project from the practical folder
2. Build the project
3. Run the project with various input arguments (e.g., data/8ints.txt)
4. Note the output and fill out the table below

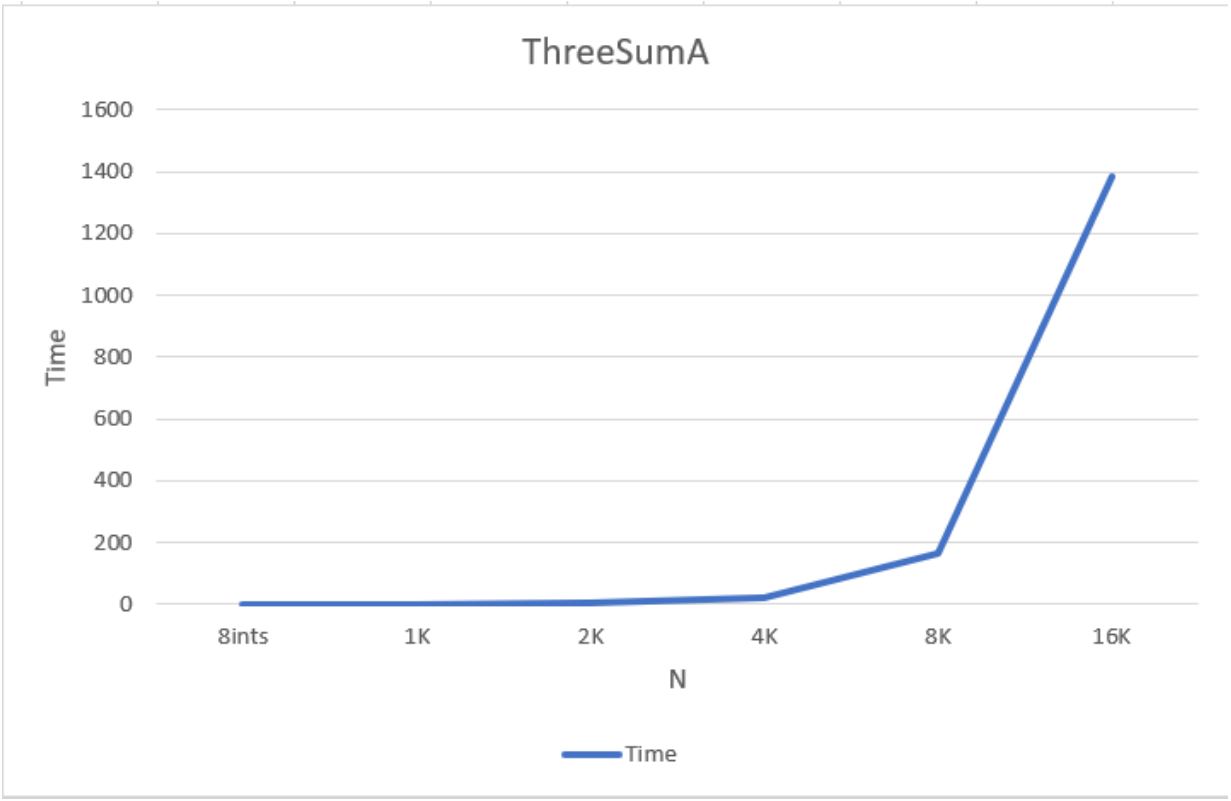
Run from the Command Line or Terminal (Mac)

1. You'll first need to compile the java files in the directory using 'javac'
2. Once compiled then you can run the algorithm by calling the class name with an argument of an input file of integers from the data folder (e.g., java ThreeSumA 8ints.txt)
3. Run the experiments several times for each input file and record the average of the result in the table below

Record the results of your timing experiments in the table below:

Algorithm	Input	Time	Number of triples?
ThreeSumA	8ints.txt	**0.0**	**4**
	1Kints.txt	**0.421**	**70**
	2Kints.txt	**3.233**	**528**
	4Kints.txt	**22.25**	**4039**
	8Kints.txt	**165.26**	**32074**
	16Kints.txt	**1385.158**	**255181**

Graph the results of your experiments in a graph like below but note that the one below is not necessarily correct.



3.Empirical Analysis: ThreeSumB

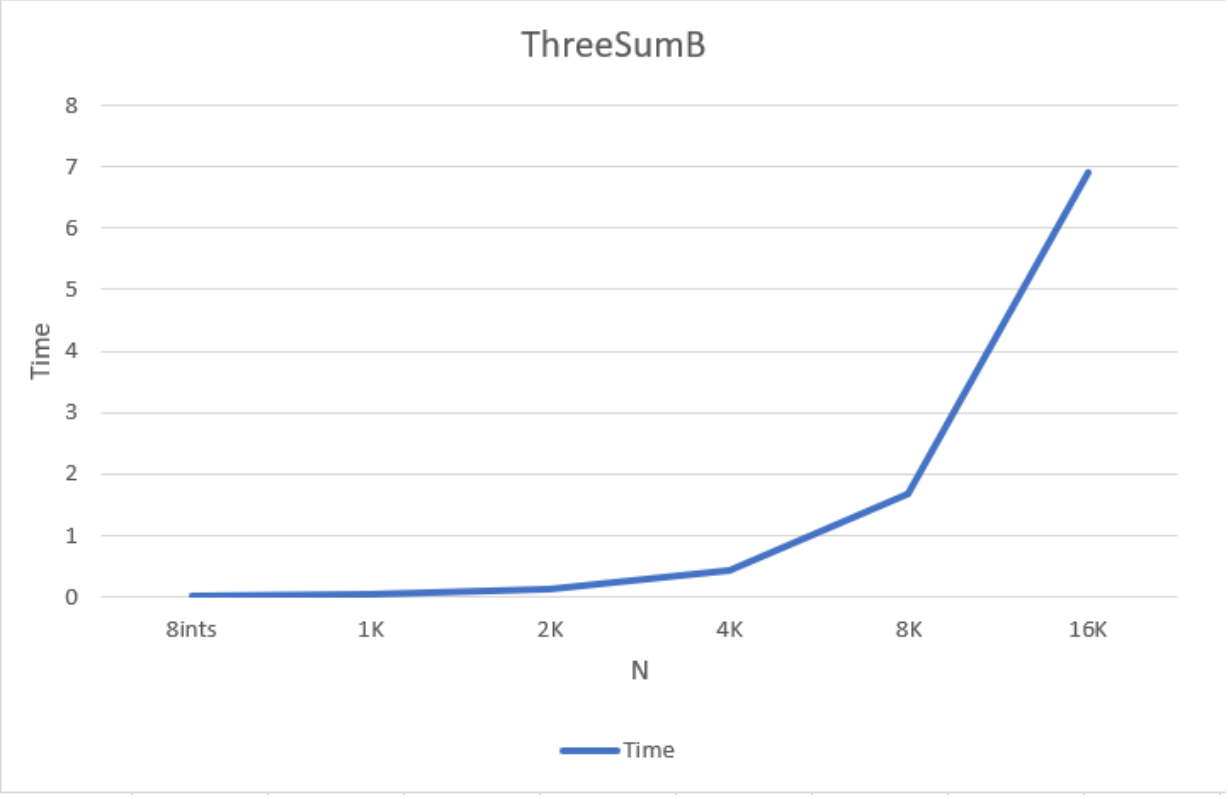
ThreeSumB is a different algorithm that solves the same Three Sum problem. Run the same experiments as before.

Record the results of your timing experiments in the table below:

Algorithm	Input	Time	Number of triples?
ThreeSumB	8ints.txt	**0.007**	**4**
	1Kints.txt	**0.033**	**70**
	2Kints.txt	**0.116**	**528**
	4Kints.txt	**0.425**	**4039**
	8Kints.txt	**1.683**	**32074**
	16Kints.txt	**6.903**	**255181**

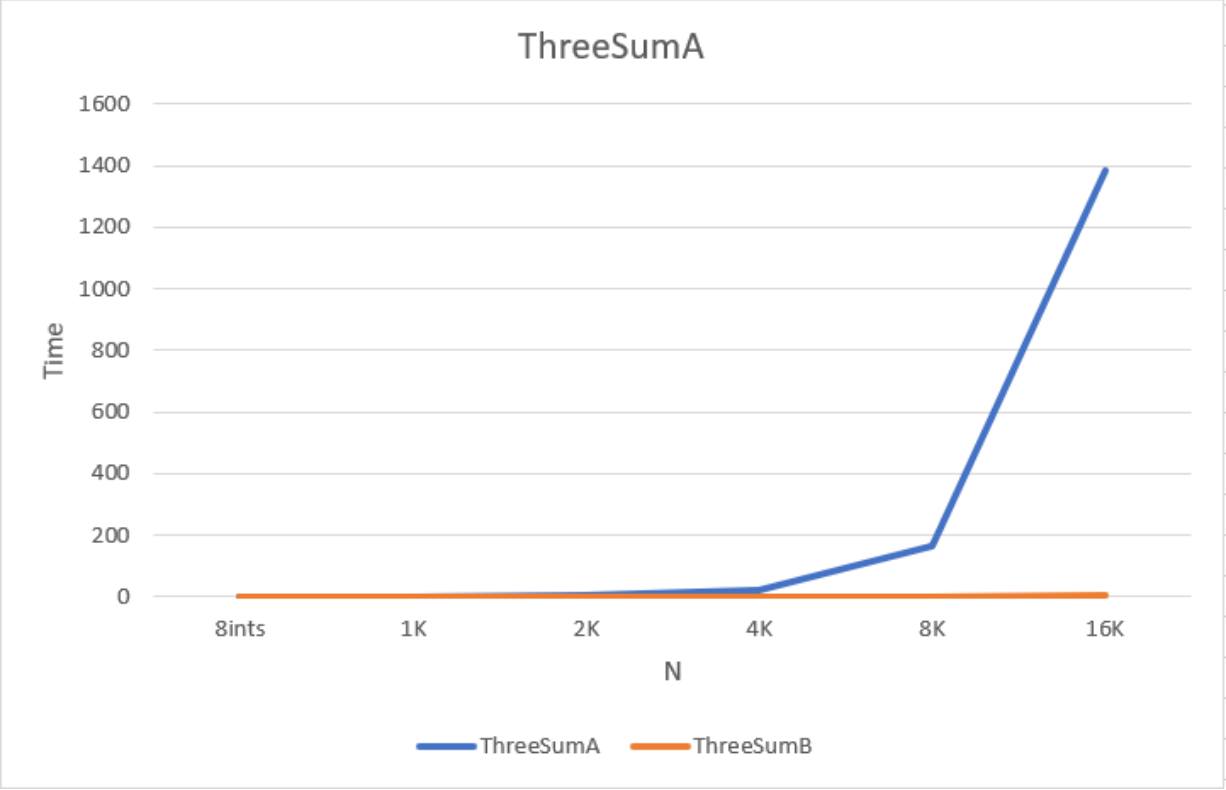
Graph the performance

Again graph the performance of ThreeSumB.



Compare: Graph the two together!

Graph both your experiments in one graph like the one below although **note the one below is not necessarily correct.**



Questions:

1. Which algorithm performs better (i.e. take less time in relation to the size of the input)?
2. Why do you think this is the case?

Q1 Answer: ThreeSumB performs better than ThreeSumA

Q2 Answer: ThreeSumB performs better because the array is sorted using Arrays.sort(). As well as that, ThreeSumA has a time complexity of $O(n^3)$ due to three for loops whereas ThreeSumB has a time complexity $O(n^2 log(n))$ because of two for loops and the use of Binary Search.