COMP2054-ADE FORMATIVE LAB
Primitive Operation counting and Graphs.

Objective

Give practice at

- Annotating a (fairly trivial) program to do counting of primitive operations.
- Running experiments to look at the scaling behaviour; plot a graph; be able to relate to “Big Oh”.

Description

First, you should download the file Main.java from Moodle.

Your first task should be to read it carefully, and then check that you can compile and run it. Then there are two parts to the lab:

A. “**Counting**”. In the file “Main.java” you are given a fragment piece of code for a method “arrayMax”. It contains lines “c += ??” to increment a counter of primitive operations. You are required to modify these increments to do some approximate counting of the "primitive operations" that it performs.

B. “**Graph**”. Using the resulting, modify the Main routine as appropriate, for example changing the range of values of the input size, and the number of runs at each size.
Take the resulting outputs and use them to produce an initial scatterplot graph (e.g. in Excel) of how the behaviour changes with n. Then add to the graph extra ‘lines’ that illustrate the scaling of different aspects of the runtime: The main aspect of runtime that you should consider is just the worst runtime, and show that it is always bounded above by some linear function. I.e. use that the meaning of “f is O(n)” is that f is bounded above by some multiple of n, and show it using a graph.

C. Advanced:
Also consider the best and average runtimes.
Also, consider different ways of initialising the array, and study how it affects the scaling.
Challenging: suppose that the initial array is initialised at random, and only count how many times the value of “currentMax” is increased (i.e. only increment c for that line, as in the given Main.java). What is the apparent scaling in this case? Surprised? Can you explain? (Only looking for hand-waving answer – not full analysis!).

D. After you have finished with arrayMax, then also implement the program from lec01 that does the repeated “divide by 2”, and do a similar analysis. To get a graph that shows an overall “log n” behaviour.

Comments: This is deliberately rather open-ended – and is designed to just give a chance to practice thinking about:

1. **counting of primitive operations**
2. **how to do empirical studies of scaling – including skills in making graphs**
3. **how to relate these to “theory” of big-Oh**