Size Does Matter Part One

When does size matter? When an algorithm is automated and executes on its input to produce an output. For example, one might have to wait a significant amount of time for a list of ***100 000*** strings to be rearranged into alphabetical order by a computer. The time an algorithm takes to execute compared with another algorithm performing the same task is of considerable interest when the input data set is of significant size. Computer Scientists are interested in estimating, in general, how a particular algorithm's execution time depends on the size of input notwithstanding that different computers run at different speeds. To eliminate the difference in speed between computers, Computer Scientists try to answer the question "what happens to the execution time when the size of the input is doubled?" For example, if the number of strings to be placed in alphabetical order is doubled does it take twice as long, four times as long, etc?

Many students have difficulty solving problems where the solution necessarily involves making estimates. The Physicist Enrico Fermi used to challenge his classes with problems that at first glance seemed impossible. One such problem was that of estimating the number of piano tuners in Chicago given only the population of the city. The answer is about 150 (to see a solution visit www.educational-computing.co.uk). The approach to solving this type of problem became known as the Fermi Approach. This approach relies on knowing some facts, ignoring unnecessary details and making some reasonable assumptions. The same approach can be used to estimate the speed of execution of an algorithm.

It helps if the activity can be made a kinaesthetic one. One such activity is tracing a Bubble Sort on weights that need to be ordered by increasing size of weight. **Figure 1** shows an abbreviated outline of this activity on a set of weights, **A**, **B**, **C**, **D** placed in descending order of weight, **D**, **C**, **B**, **A**. **D** is first compared in weight with **C**, then **D** with **B**, and so on. The process moves **D** creating a new ordering **C**, **B**, **A**, **D**. The weighing cycle begins again and finishes with the ordering **B**, **A**, **C**, **D**. The final cycle produces the desired outcome **A**, **B**, **C**, **D**. If the letter ***n*** is used to represent the number of weights then the number of weighing cycles is ***n – 1***. This is exactly the same as the number of ***fetch-weigh-return*** operations in each cycle, i.e. ***n – 1***. ***Fetch-weigh-return*** operation consists of fetching two adjacent weights, placing these in separate pans of the weighing scales, observing the difference in weight, if any, then returning the weights to their respective positions, swapping their positions if necessary. Let's represent the time in seconds for a ***fetch-weigh-return*** operation by the letter ***t***, then one cycle will take ***(n – 1)*** times ***t*** seconds. Therefore, ***n – 1*** cycles will take ***(n – 1)*** times ***(n – 1)*** times ***t*** seconds. The **Enrico Fermi Approach** says that we can approximate ***(n – 1)*** to ***n*** when ***n*** is large compared to ***1***, say ***n*** = 100 weights.
Therefore,

(n – 1)(n – 1)t is to a good approximation ***n*** times ***n*** times ***t*** when ***n*** is large.

We write ***n*** times ***n*** times ***t*** as n^2t. We now have a formula for how long the weighing algorithm will take to execute on large inputs. Call this total time ***T***.

$$T = n^2t$$

We can now use this formula to estimate by what factor the execution time changes when the input size is doubled, e.g. from ***100*** weights to ***200*** weights.

Time for ***100*** weights,

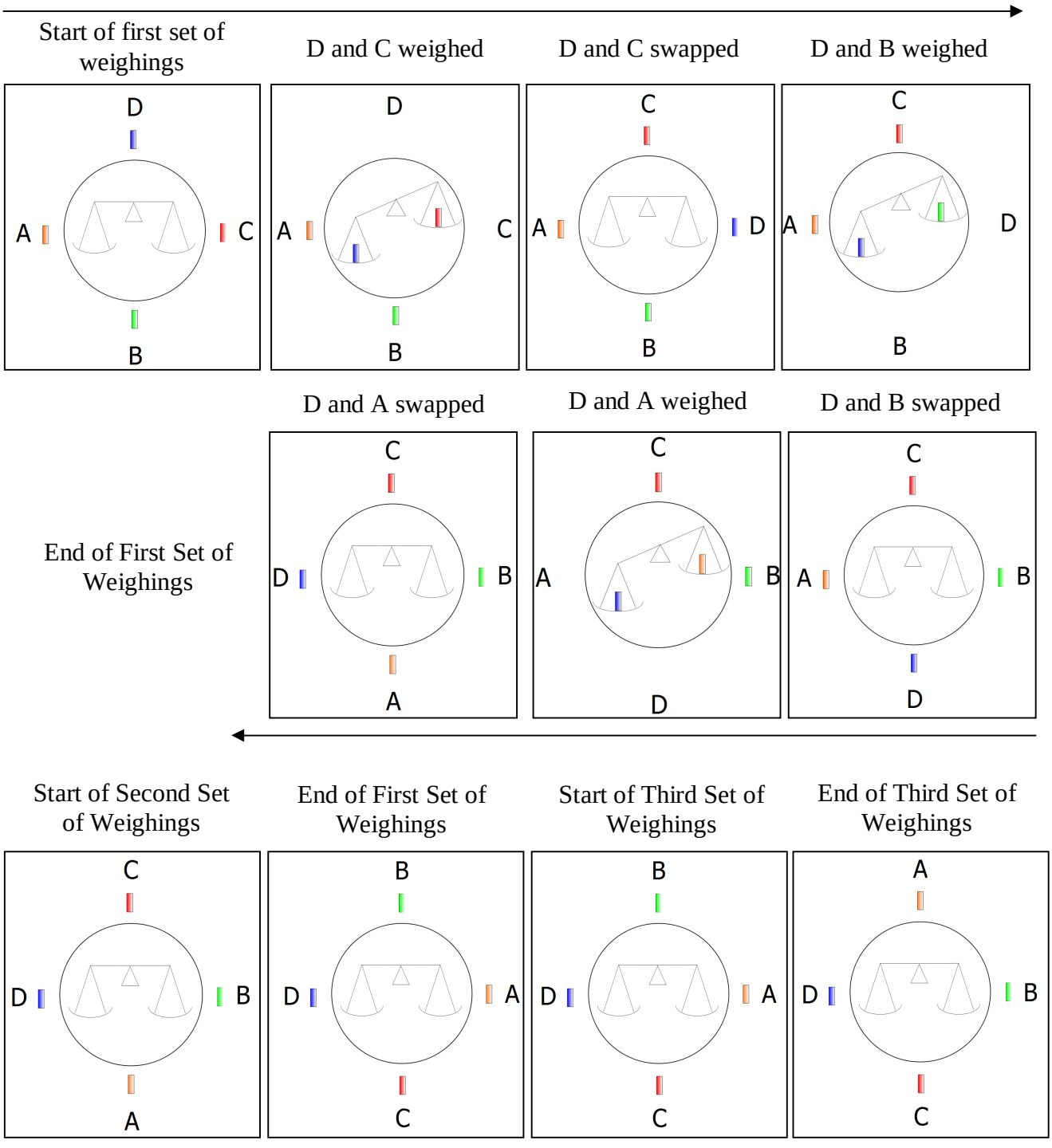
$$T_{100} = 100 \times 100 \times t$$

Time for ***200*** weights,

$$T_{200} = 200 \times 200 \times t$$

This is ***200*** x ***200*** compared with ***100*** x ***100*** of ***t*** or ***40 000*** compared to ***10 000***, a ratio of ***4*** to ***1***. This is bad news, doubling the number of weights results in the re-ordering of the weights process taking ***4*** times as long. Let's say that, that the time ***t*** for the ***fetch-weigh-return*** is ***5*** seconds, then $T_{100} = 10000 \times 5 = 50\,000$ seconds and $T_{200} = 40000 \times 5 = 200\,000$ seconds. This is approximately ***56*** hours compared with ***14*** hours!

Figure 1



In practical terms, this exercise could be carried out with a set of weights constructed from plastic 35mm film canisters or similar canisters[1] obtained from education supply companies such as Griffin Education containing coins to make the different weights. The canisters should be colour coded so that they can be distinguished visually. A weighing balance is not necessary because the weights can be compared by hand. The activity can be extended with a "no exchange of position of weights" indicator that can be a cup placed either up or down and the cycles of weighing continued until the no exchange indicator indicates "no exchange of position of weights" by the cup remaining in the up position. Before each weighing-cycle the cup is set to the up position.

[1] Plastic coloured sauce containers can also be used ï a pack of four can cost as little as £2.99 from discount stores - or alternatively small paper cake cases, each containing a different number of penny coins ï say, one, two, three and four, respecyively - will do.