

Empirical analysis of an algorithm

A Report by Douglas Brennan – n7326645



April 2018

CAB301 Algorithms & Complexity

Summary

The following is an in-depth analysis of the complexity of an algorithm that finds the minimum and maximum integers of a given array.

1 - Description of the Algorithm

The main priority of the given function (Figure 1) is to find the maximum and minimum values of a supplied array of integers. To accomplish this the method uses a for loop to iterate through each number, then uses if-statements to check if the iterated number is larger or smaller than the previously stored maximum and minimum numbers. If this occurs the new number is set as either the maximum or minimum value dependant on the value.

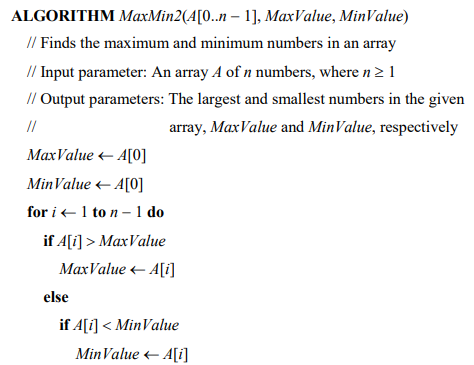


Figure 1

2 - Algorithm Analysis

This segment goes into detail about the complexity of the algorithm and the efficiency in relation to execution time.

2a - Basic Operation Identification

According to (Levitin, 2012), a basic operation is the most important task of an algorithm, it is the operation that contributes the most towards the total running time. Therefore, the basic operation is any task which is nested inside the for loop, they occur for each integer instance in the array. In this case the basic function is recognised as both the if-statements inside the loop.

2b – Algorithm Time Efficiency

The algorithms time efficiency (C(n)) is affected by the size of the input (n) and the value matrix of the array. The algorithm can perform different numbers of basic operations at each iteration, so its best-case, worst-case and average-case efficiency are unique (Paul, 2005).

The best-case scenario is when only one of the two if-statements is used, this occurs when the values are in strictly increasing order, represented as;



The worst-case scenario occurs when the first integer in the array is the largest, this causes the second if-statement to occur for each iteration of the for-loop, represented as;



2c – Average-Case Efficiency & Order of Growth

An algorithms average-case efficiency best characterises the relationship between the number of basic operations performed and the input size (Levitin, 2012). In relation to the algorithm the average-case efficiency is classed as Θ(n), this is because it is bounded above by the worst-case scenario and bounded below by the best-case scenario.

Functions classed as Θ(n) have linear growth for they are asymptotically bound. According to (University of Nebraska-Lincoln), “As n increases, f(n) grows at the same rate as g(n). In other words, g(n) is an asymptotically tight bound on f(n).”

3 – Methodology, Tools & Techniques

This section summarises the computing environment used for data collection.

**Environment:** Visual Studio 2017 was used as the environment to conduct the testing and results were exported to Microsoft Excel for data collection. A console application was chosen within Visual Studio, and the script language was C#.

**System:** The tests were performed using an Acer Aspire laptop with the following specifications;

* Operating System: Windows 10 Home Edition - 64-bit
* Random Access Memory: 16gb
* Processor: Intel® Core™ i7-7700HQ CPU @ 2.80GHZ

**Test Data:** To generate test data various techniques were implemented to calculate large variances of results.

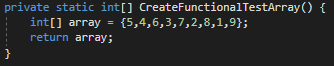
* Functional Test: An array that contains fixed vales which trigger each basic operation multiple times when executed, this is test the algorithms functionality
* Ascending Array Test: An array containing 100 ascending integers from 1-100 was tested to replicate and prove the best-case scenario.
* Descending Array Test: An array containing 100 descending integers from 100-1 was tested to replicate and prove the worst-case scenario.
* Random Array Test: To properly test the average-case scenario, arrays of the same length were tested that contain random variables. A random number generator was implemented to ensure the value matrix of the tested arrays were unique.

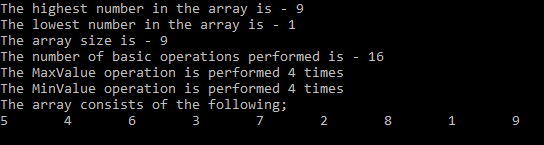
4 – Test Results

This section describes the results of the tests environments described in section 3.

4a – Functional Testing

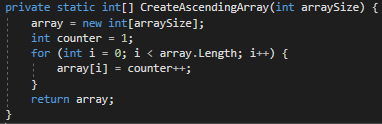
To ensure that the implementation of the algorithm functions correctly, an array containing 9 fixed values were parsed that would cause both basic operations to trigger multiple times. Using random results like in the Consecutive Random Array Test would not provide desirable test results for testing algorithm functionality. The parsed array and corresponding results are as follows;

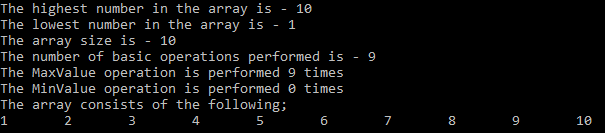




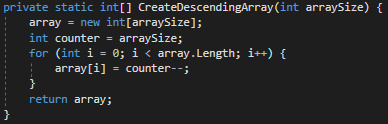
4b – Best, Worst & Average-Case Scenarios

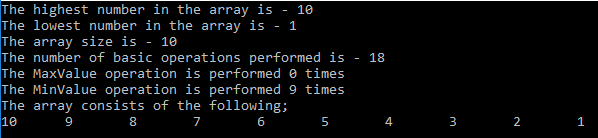
* **Best Case:** As explained in 2b the best-case efficiency occurs when the array matrix values are in ascending order. The below are results of an array of size 10 with values ranging from 1-10 in ascending order. The second if-statement from Figure1 is never performed in this scenario meaning that best-case efficiency (n-1) has been achieved.

****



* **Worst Case:** As explained in 2b the worst-case efficiency occurs when the array matrix values are in descending order. The below are results of an array of size 10 with values ranging from 1-10 in descending order. The second if-statement from Figure1 is always performed in this scenario meaning that worst-case efficiency (2n-2) has been achieved.

****



* **Discovery:** A notable finding of this exercise is that arrays with larger value matrices noticeably increase the frequency in which higher amounts of basic operations are performed. Each below graph shows the results of 100 tests with array sizes incrementing from 1-100. The graph results show that the variable which has the most effect on the number of basic operations (apart from the arrays input length) is the value matrix. The first graph shows arrays with values ranging from 1-10, the second with values ranging 1-100 and the third with values ranging from 1-1000.
* **Discover (Continued):** What can be derived from these graphs is that the element that has the biggest impact on the number of basic operations performed (apart from the arrays length) is the size of the integers in the value matrix.
* **Average-Case Efficiency:** To calculate the average case efficiency tests using the same array size are performed. The following graphs show results of tests with arrays of size 100 which contain random integers. The first graph shows values of 1-10, the second shows values of 1-100 and the third shows values of 1-1000.

Based on the 100 tests of values between 1-10 the average number of basic operations performed equalled 100.84.

Based on the 100 tests of values between 1-100 the average number of basic operations performed equalled 103.06.

Based on the 100 tests of values between 1-1000 the average number of basic operations performed equalled 103.23.

5 – Results & Conclusion

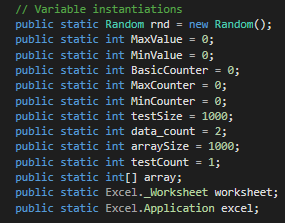
The conducted test results for the algorithm show the mean number of basic operations performed when given random integers is roughly (n+3). Based on the above test results it can be concluded that the order of growth for the given algorithm is linear and has the efficiency class of Θ(n). To further show this, an average-case efficiency was calculated for 1000 iterations of the function with a static array size of 1000;

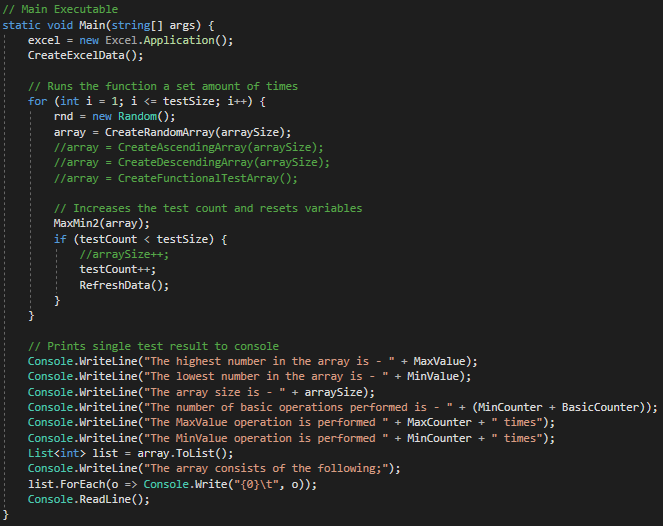
Even when the array size was increased to 1000 the average amount of basic operations performed was 1003.96. This result proves that the order of growth is Θ(n) for it is bounded above by the worst-case scenario (2n-2) and bounded below by the best-case scenario (n-1).

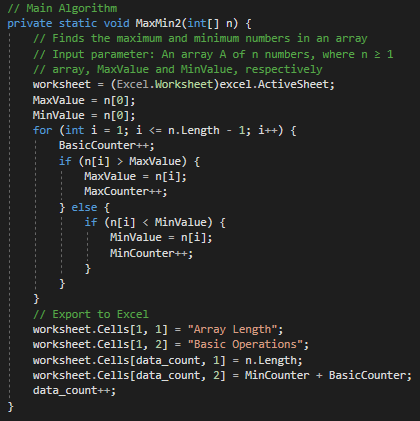
The results show that the average-case efficiency is roughly (n+3) for the given algorithm, and is not much worse than the best-case scenario (n-1). These results could be deemed as controversial for it goes against what Berman and Paul stated as the average-case efficiency (2n – ln n – 1) for this very same algorithm. To the best knowledge of the author, the algorithm for this exercise was implemented correctly, although there is chance for error.

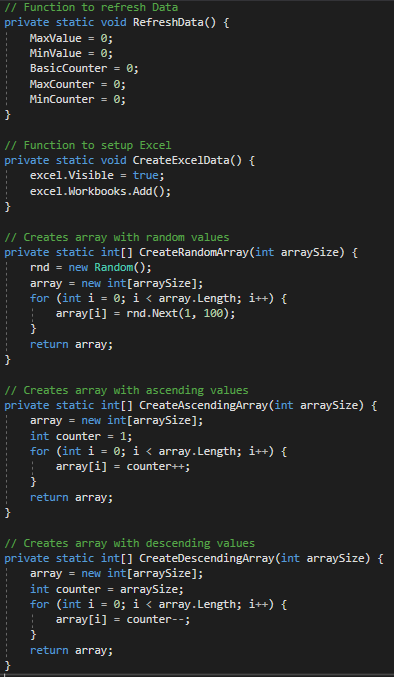
The following shows excerpts of the implemented algorithm in the development environment.

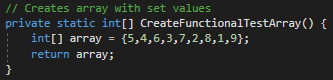
6 – Code Implementation











# 7 - Bibliography

Levitin, A. (2012). *Introduction to The Design & Analysis of Algorithms.* Retrieved from vgloop: http://www.vgloop.com/\_files/1394454921-126688.pdf

Paul, K. A. (2005). *Algorithms: Sequential, Parallel and.* Thomson.

University of Nebraska-Lincoln. (n.d.). *University of Nebraska-Lincoln .* Retrieved from Growth of Functions and: http://cse.unl.edu/~sscott/teach/Classes/cse156S06/Notes/AsymptoticNotation.pdf