|  |  |
| --- | --- |
| CAB301 Assignment 1  Empirical Analysis of an Algorithm | N9845097  Ka Long Lee (Eric) |

Table of Contents

[1. Summary 2](#_Toc5495446)

[2. Description of Algorithm 2](#_Toc5495447)

[3. Implementation of the Algorithm 3](#_Toc5495448)

[3.1. Program Implementation 3](#_Toc5495449)

[3.2. Functional Testing 4](#_Toc5495450)

[4. Design of Experiments (2 Pages) 5](#_Toc5495451)

[4.1. Methodology, tools and techniques 5](#_Toc5495452)

[4.2. Data sizes, Test Data set 5](#_Toc5495453)

[5. Experiential results (4 Pages) 6](#_Toc5495454)

[5.1. Basic Operation identified 6](#_Toc5495455)

[5.2. Average execution time 6](#_Toc5495456)

[5.3. Experience to measure the program execution times 6](#_Toc5495457)

[6. Analysis of Experiential results (4 Pages) 6](#_Toc5495458)

[6.1. Experimental results 6](#_Toc5495459)

[6.2. Comparing against the theoretical efficiency prediction 6](#_Toc5495460)

[7. Reference (1 Pages) 6](#_Toc5495461)

[8. Appendices (4Pages) 7](#_Toc5495462)

# Summary

This report is to analysis the time complexity and the average case efficiency of the Brute Force Median algorithm. The Brute Force Median algorithm is implemented in C# 7.3. In this report, it analysed the relation between the number of basic operation and the input size how they influence the execution time of the algorithm. It is obvious that if there is more the numbers of integer in the array, the execution of time is longer it takes.

# Description of Algorithm

The main purpose of the Brute Force Median algorithm is to find out the median value in the given array which contains a set of integer values. Considering the different condition that the array is ordered by. Returning the value in the middle position of the array is not the perfect way to find out the median value of an array. Because the array might be sorted randomly or exist duplicate number in it. That is the reason why we need to implement Brute Force Median algorithm to retrieve the median value in the array accurately in different condition.

The Brute Force Median algorithm takes any array which contains integers and finds the median value of the array. It selects all the element to compare it one by one in a nested for loop. At the end of each iteration, it checks the current selected item and return it to the user if the value of the selected item is median. However, it will keep iterating until the median found in the array or return -1 to the user when the input array is in a invalid format such as empty array.

The algorithm should return the correct median value in the integer value set despite that the order is sorted randomly, in ascending order or in descending order. It also able to return the correct median value whatever the length of the array is odd or even numbers. Duplicate value in the array will not influence the algorithm works. The disadvantage of this algorithm is that the execution time is growing dramatically according to increasing the number of array length. Therefore, the array length is lower, the algorithm is more efficient.

# Implementation of the Algorithm

# Program Implementation

The algorithm was implemented on the C# programming language. The source code of the algorithm displayed in Appendix A is saved in program.cs file. The algorithm takes only one parameter which is an array contains one or more integer numbers and assigned to the variable named as A. First of all, the algorithm creates one local variable named as k. It is for calculating and storing the median position of the input array. There are two steps for calculating the median position of the array. Firstly, dividing the length of the array by two. Secondly, ceiling the previous result to the highest integer value it can be.

If the length of the array is an odd number, ceiling the result can help us pointing to the median position accurately. In contrast, we assume that dividing the length of the array by two is the median position for an array of even length. Because it doesn’t exist a correct way to define the median position for an even length array. Finally, the result will be assigned to the k variable. If the input array was already sorted before it passed to this algorithm, the median value of the array must be located at the k position.

The algorithm then creates a nested for loop which will repeat the operations inside the block according to the numbers exists in the input array. There are total two for loops in the nested loop. Both loops create an indexer variable for selecting elements in the array for different purpose inside each loop. These indexers start from 0 to the length of the array. The main purpose of the outer loop is to determine the array element selected by the indexer of the outer loop whether median or not. On the other hand, the purpose of the inner loop is for calculating how many elements are smaller and equals to the element selected by the indexer of the outer loop.

In the outer loop, it creates an Indexer variable denoted as i. This indexer only allowed to increase once when the outer loop finished. There are two local variables inside this loop. The first variable is named as numsmall which records how many elements in the input array are smaller than the selected element by indexer i which is denoted as A[i]. The second variable is named as numequal which records how many elements are same as A[i].

In the inner loop, Indexer j is created and updated only inside the inner loop. It checks all the elements in the array to compare with A[i] the element. If A[j] which is the selected element by indexer j is smaller than A[i], numsmall variable will increase one. On the other hand, numequal variable will increase one when A[i] equals A[j]. It is significant to note that the basic operation numsmall variable or numequall variable increasing one might not be run when A[j] is bigger than A[i].

At the end of the outer loop, if numsmall variable is smaller than k meanwhile numsmall plus numequal is larger or equals to k, it means that i is at the median position. Therefore, the program will return A[i] the element which selected by indexer i to the user. On the other hand, the invalid output is passed as the parameter. Such as an empty array. The algorithm will return -1 to the user.

# Functional Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Test instance | Expected Output | Actual Output | Test Result |
| Return median from Sorted array | A = [1,2,3,4,5,6,7] | 4 | 4 | Correct |
| Return median from an array which the length is an even number and elements sorted randomly | A = [8, 12. 7, 9] | 8 | 8 | Correct |
| Return median from an array which the length is an odd number and elements sorted randomly | A = [8, 5, 9, 1, 4] | 5 | 5 | Correct |
| Return median from a reverse order array | A = [9,8,7,6,5,4,3] | 6 | 6 | Correct |
| Return median from an array which contains negative numbers | A = [-5, -3, 5, 4, -2] | -2 | -2 | Correct |
| Return median from an array which contains duplicate numbers | A = [1,2,2,2,4,5,6,6,7] | 4 | 4 | Correct |
| Return Median from an array which only contains one number | A = [4] | 4 | 4 | Correct |
| Return -1 from an empty array | A = [] | -1 | -1 | Correct |

Table 1. Functional Test Case Result

The Implementation of the algorithm in C# Programming language is displayed in Appendix A. The functional correctness of the algorithm was tested by using test function which defined in TestCast.cs file. Appendix B-1 shows the unit test code for the brute force median algorithm. There are eight test function in total to test out the algorithm should return the median value in the array in different condition. These conditions include sorted, reverse, random, duplicate, single length, invalid length, odd length and even length array.

The table above recorded each test array, expected and actual output for each test case. Appendix B-2 also displayed that all the unit test case run correctly in the Visual Studio IDE. These unit test confirm that the implementation of the algorithm returning the median value successfully in every different condition. However, if the input array is an empty array, it will return -1 to the user as the error code.

# Design of Experiments (2 Pages)

# Methodology, tools and techniques

1. The Brute Force Median algorithm were implemented on the C# programming language. C# is one of the famous object-oriented programming languages in 2019 which is able to develop different types of applications such as a web, mobile, server, console applications. In this case, the algorithm is developed as a unit test application. Visual Studio is a software which used to run the experiment program. It provides lots of tools for the experiment, such as unit testing, debug console. Microsoft also provides Visual Studio software running in most of the popular operating system. Such as MacOS, Window, Linux Ubuntu.
2. The following experiment were performed on a 15-inch MacBook Pro 2018 model. The operating system on this computer is MacOS Mojave with 16Gb of RAM memory, Intel Core i9 Processor running at 2.9GHz. It also has 256GB storage space in a solid-state disk (SSD) to achieve the best performance as accurate as possible.
3. The basic operations and the execution times results are recorded into two csv files. It is a Microsoft Excel file than contains table of the record. Using these results, we were able to produce line graphs in Microsoft Excel to analyze the results.
4. The external library CsvHelper was used to write the basic operations and the execution times results into a CSV file.

# Data sizes, Test Data set

1. The Test Data set is generated randomly by the program. The array length of the data starts from 1000 to 20000, in increments of 1000.
2. The numbers in the array are generated randomly between 1 – 20000. The random number generator is used the real time as seed. It can ensure that the program will generate unique random number when the program runs at different time.

# Experiential results (4 Pages)

# Basic Operation identified

It is significant that The nested for loop

# Average execution time

# Experience to measure the program execution times

# Analysis of Experiential results (4 Pages)

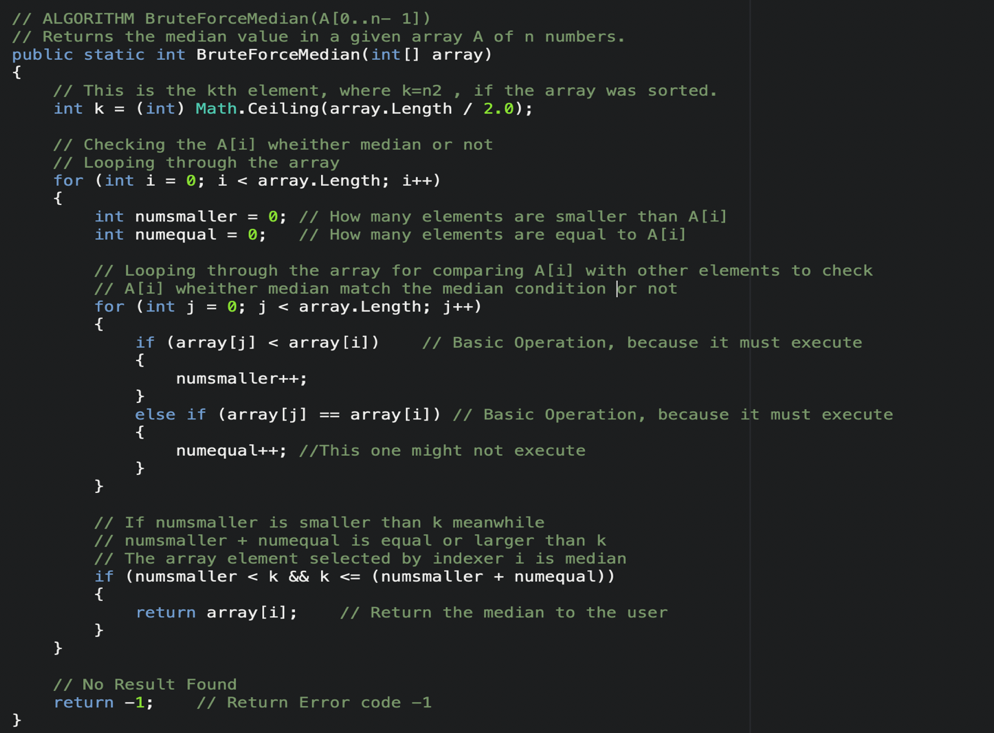
# Experimental results

# Comparing against the theoretical efficiency prediction

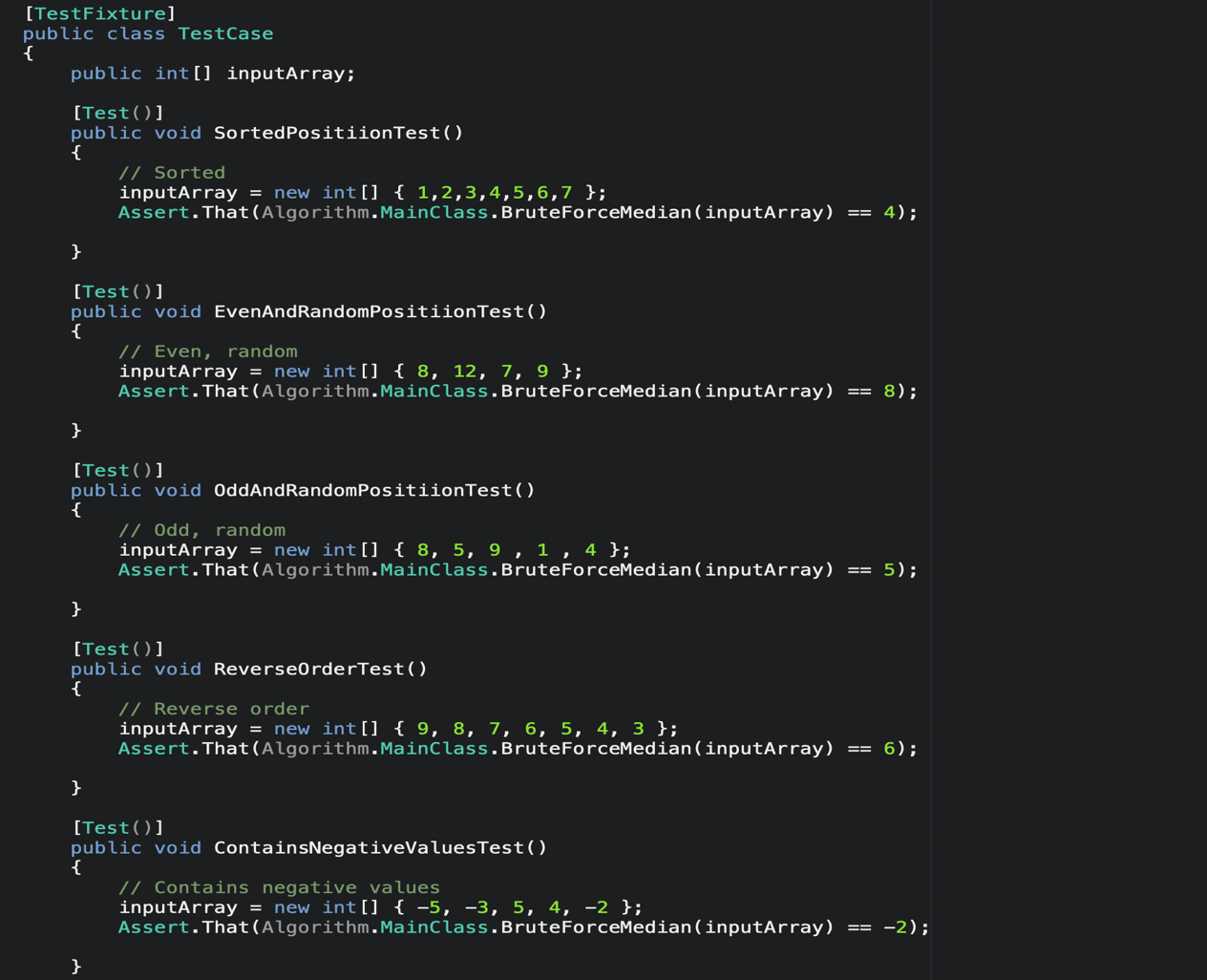
# Reference (1 Pages)

# Appendices (4Pages)

Appendix A – Brute Force Median Algorithm source code in C#



Appendix B-1 – Functional Test code for Brute Force Median Algorithm in C#

x

Appendix B-2 – Test result for Brute Force Median Algorithm

