SCHOOL OF COMPUTER SCIENCES

SESSION 2019/2020 SEMESTER II

CPT 212 : DESIGN AND ANALYSIS OF ALGORITHMS

# ASSIGNMENT 1: PRINCIPLE OF ALGORITHM ANALYSIS

**SUBMISSION DATE: 12TH APRIL 2020**

**Lecturer's Name: Dr Teh Je sen**

| Name | Matric number | Division of tasks |
|---|---|---|
| Ainil Hawa Binti Abdul Rozak | 141928 | **Quadratic - O** $(n^2)$ |
| Farah Mursyidah Binti Fuahaidi | 144395 | **Linear - O** $(n)$ |

**Table of Contents**

## 1.0 Algorithms and pseudocodes

I. **O ( *n* )**

*Algorithm minAndMax (arr[ ],n)*
*Input: array Arr of n element*
*Output: Minimum and maximum values of generated numbers in array*

```
Int max ← arr [ 0 ]
For  i ← 0 to i <= n-1
        If arr[ i ] > max
                Max ← arr [ i ]
                End if
        End for
Int min ←  arr [ 0 ]
For i← 0 to i <= n-1
If arr[ i ] < min
                Min ← arr [ i ]
                End if
        End for
```

This algorithm shows the process in obtaining minimum and maximum values of values that are stored in the file in an array. This algorithm consists of two separated for loops and each of this loop executes dependently with the value of ‘n’. The value of ‘n’ will be 1,10,100,1000,10 000,90 000 and 100 000.

The first loop is the process of traversing the whole array until index n-1 to search for the randomly generated maximum value in the array. Firstly, The first element of the array will be assigned to the variable max. Then, during the traversing process, if any of the array elements is larger than the value of max, the value of that particular element will be assigned to max.

The third loop is quite similar to the second loop. It involves the process of traversing the whole array until index n-1 to search for the randomly generated minimum value in the array.

Based on this algorithm, it is clear that it is dependent on the input of n and will grow proportionally to the size of input data n.

## II. O ( $n^2$ )

```
Algorithm selectionSort ( arr[ ], n )
        Input: Array arr of n element
        Output: Array arr containing permutation input of the input such that
                Arr[0]≤arr[1]≤...≤arr[n-1]

For i ←0 to i<n-1
        min ←i
        For j = i+1 to j<n
                If  arr[j]<arr[min]
                        min ←j;
                End if

        End for
        temp←arr[i]
        arr[i] ←  arr[min]
        arr[min]←temp;
End for
```

This algorithm sorts an array by repeatedly finding and placing the minimum element from an unsorted component at the beginning. This algorithm will sort the integers that are stored in the file  based on the value of n in increasing order. The value of n will be according to the questions which are n=1,10,100,1000,10 000,90 000, 100 000. This algorithm consists of two loops. The first loop which is for loop, this algorithm will start by checking the first element in the array. The second loop which is for loop, it will loop through all indexes to test against element after i to find the smallest element. Lastly, it will swap the values.

### 2.0 Experiment 1- Comparing algorithms based on runtime

## I. O ( $n$ )

| Value of n | Running time of computer 1 (seconds) | Running time of computer 2 (seconds) |
|---|---|---|
| 1 | Time taken by program is : 0.000000 sec | Time taken by program is : 0.001000 sec |
| 10 | Time taken by program is : 0.001000 sec | Time taken by program is : 0.005000 sec |
| 100 | Time taken by program is : 0.044000 sec | Time taken by program is : 0.035000 sec |

| | | |
|---|---|---|
| 1000 | Time taken by program is : 0.476000 sec | Time taken by program is : 0.236000 sec |
| 10 000 | Time taken by program is : 4.167000 sec | Time taken by program is : 2.728000 sec |
| 90 000 | Time taken by program is : 13.607000 sec | Time taken by program is : 31.869000 sec |
| 100 000 | Time taken by program is : 14.349000 sec | Time taken by program is : 38.598000 sec |

**II. O ( $n^2$ )**

| Value of n | Running time of computer 1 (seconds) | Running time of computer 2 (seconds) |
|---|---|---|
| 1 | Time taken by program is : 0.000000 sec | Time taken by program is : 0.000000 sec |
| 10 | Time taken by program is : 0.001000 sec | Time taken by program is : 0.008000 sec |
| 100 | Time taken by program is : 0.011000 sec | Time taken by program is : 0.036000 sec |
| 1000 | Time taken by program is : 0.111000 sec | Time taken by program is : 0.273000 sec |
| 10 000 | Time taken by program is : 1.316000 sec | Time taken by program is : 2.805000 sec |
| 90 000 | Time taken by program is : 19.453000 sec | Time taken by program is : 58.410000 sec |
| 100 000 | Time taken by program is : 21.796000 sec | Time taken by program is : 67.931000 sec |

## III. Graph

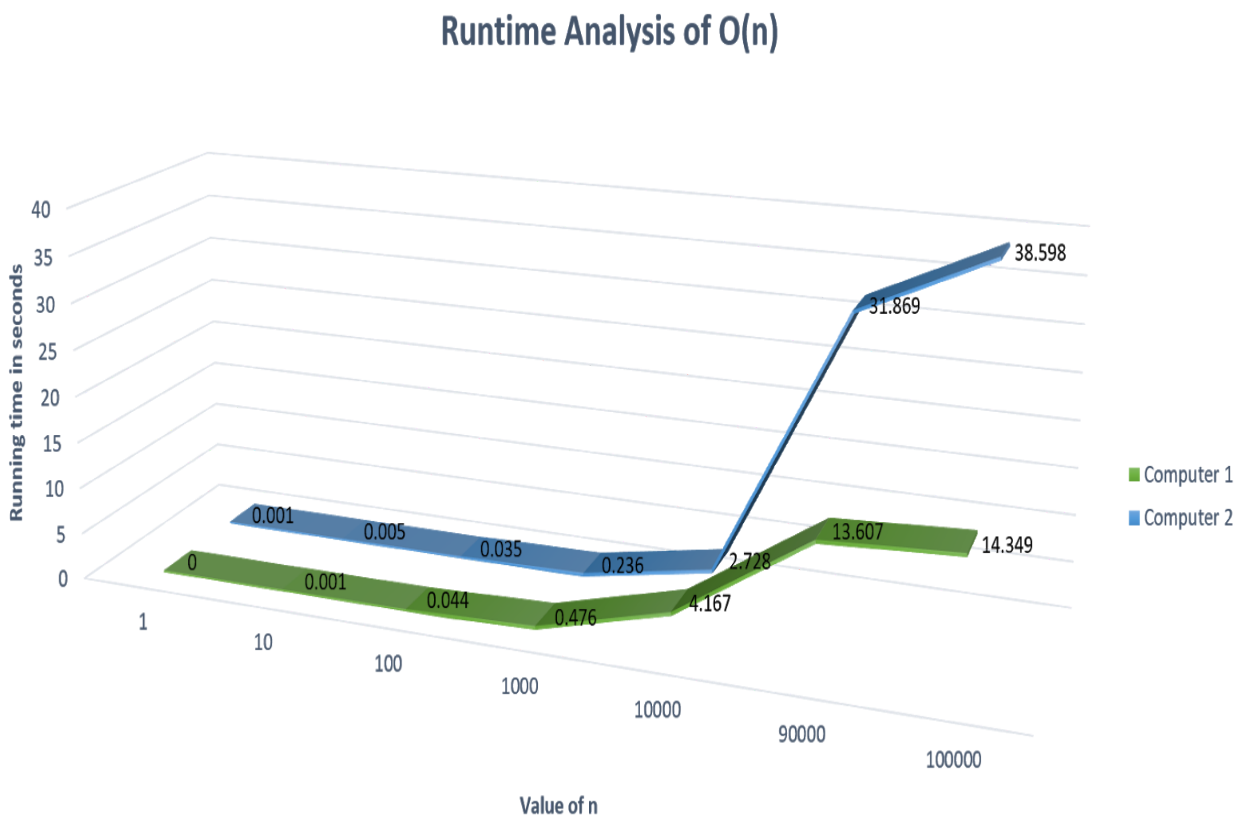



Figure 1.1

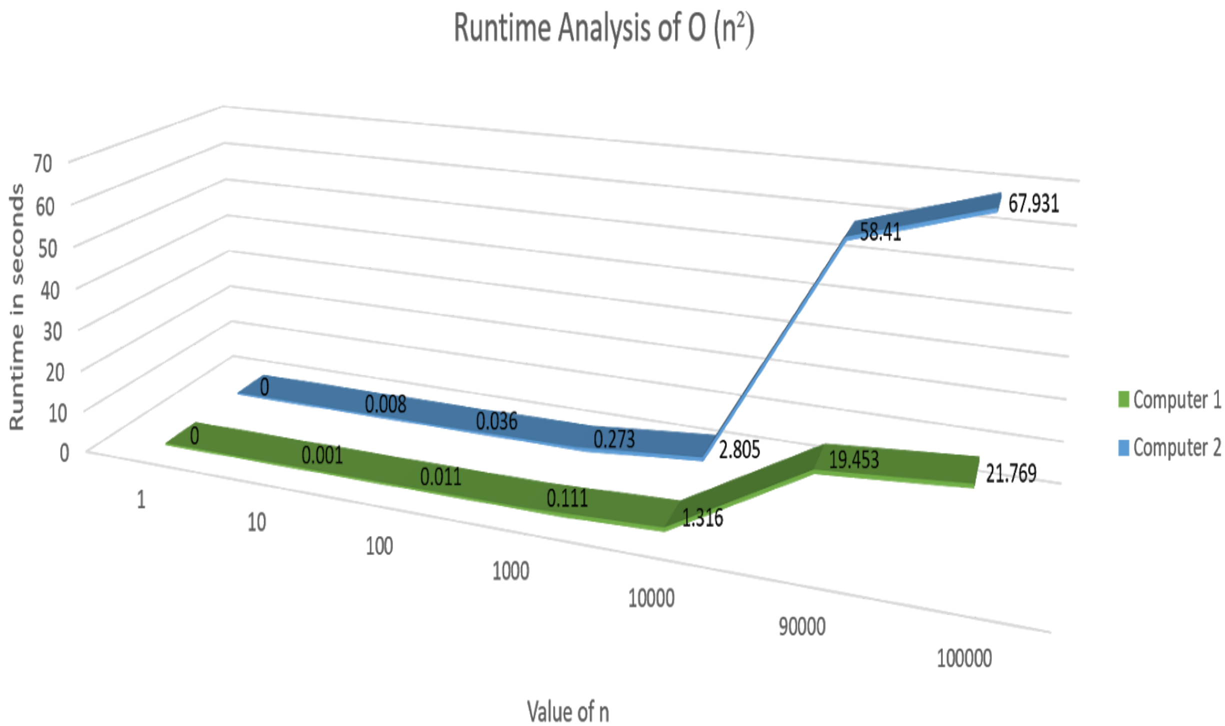

Runtime Analysis of O (n2)
Runtime in seconds
70
60
50
40
30
20
10
0
0
0.008
0.036
0.273
2.805
58.41
67.931
0
0.001
0.011
0.111
1.316
19.453
21.769
1
10
100
1000
10000
90000
100000
Value of n
Computer 1
Computer 2

**Figure 1.2**

## 3.0 Experiment 2 - Comparing algorithms based on primitive operations

### I. O ( $n$ )

| Value of n | Computer 1 | Computer 2 |
|---|---|---|
| 1 | Primitve operation: 18<br>Time taken by program is : 0.01500 sec | Primitve operation: 18<br>Time taken by program is : 0.034000 sec |
| 10 | Primitve operation: 114<br>Time taken by program is : 0.01800 sec | Primitve operation: 114<br>Time taken by program is : 0.04100 sec |
| 100 | Primitve operation: 1020<br>Time taken by program is : 0.02700 sec | Primitve operation: 1020<br>Time taken by program is : 0.05700 sec |
| 1000 | Primitve operation: 10022<br>Time taken by program is : 0.14000 sec | Primitve operation: 10022<br>Time taken by program is : 0.32800 sec |
| 10 000 | Primitve operation: 100030<br>Time taken by program is : 1.31400 sec | Primitve operation: 100030<br>Time taken by program is : 3.18100 sec |
| 90 000 | Primitve operation: 900036<br>Time taken by program is : 11.60800 sec | Primitve operation: 900036<br>Time taken by program is : 25.89800 sec |
| 100 000 | Primitve operation: 1000036<br>Time taken by program is : 12.65500 sec | Primitve operation: 1000036<br>Time taken by program is : 29.58500 sec |

### II. O ( $n^2$ )

| Value of n | Computer 1 | Computer 2 |
|---|---|---|
| 1 | Primitive operation: 4<br>Time taken by program is : 0.01400 sec | Primitive operation: 4<br>Time taken by program is : 0.01500 sec |
| 10 | Primitive operation: 419<br>Time taken by program is : 0.01700 sec | Primitive operation: 419<br>Time taken by program is : 0.04500 sec |

| | | |
|---|---|---|
| 100 | Primitive operation: 31537<br>Time taken by program is : 0.02900 sec | Primitive operation: 31537<br>Time taken by program is : 0.07800 sec |
| 1000 | Primitive operation: 3017399<br>Time taken by program is : 0.18700 sec | Primitive operation: 3017399<br>Time taken by program is : 0.35900 sec |
| 10 000 | Primitive operation: 300197640<br>Time taken by program is : 1.852000 sec | Primitive operation: 300197640<br>Time taken by program is : 3.90300 sec |
| 90 000 | Primitive operation: 2827085583<br>Time taken by program is : 31.48900 sec | Primitive operation: 2827085583<br>Time taken by program is : 61.42000 sec |
| 100 000 | Primitive operation: 4232335411<br>Time taken by program is : 35.807000 sec | Primitive operation: 4232335411<br>Time taken by program is : 74.75200 sec |

## III. Graph

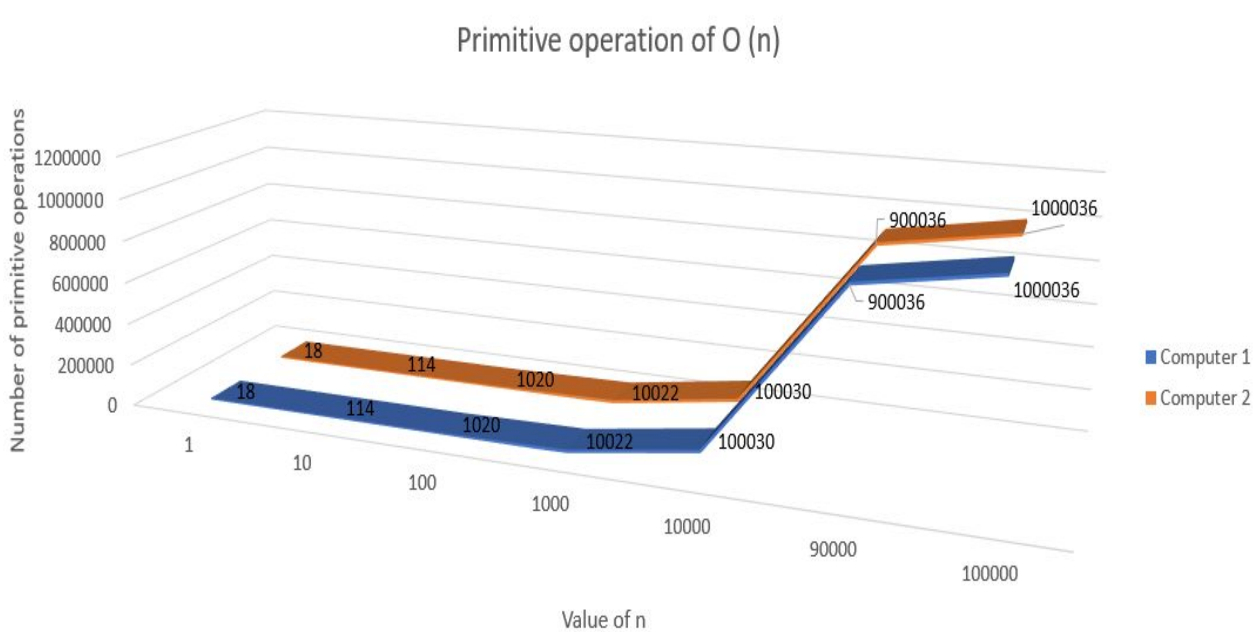



**Figure 1.3**

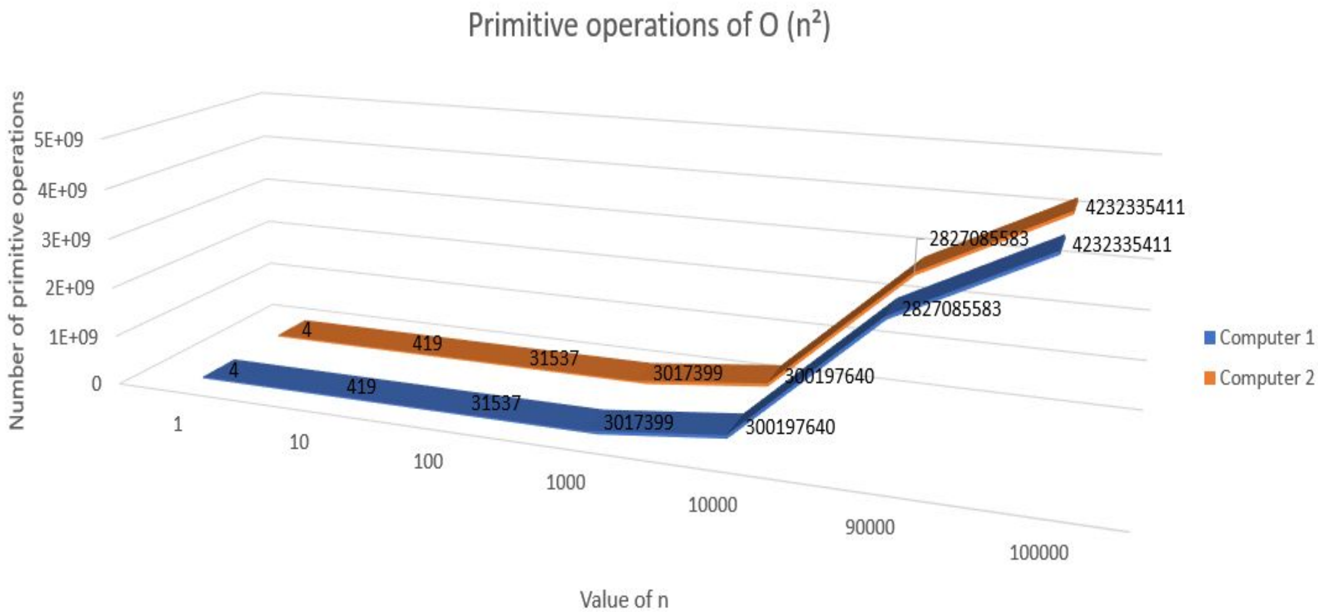

Primitive operations of O (n²)
Number of primitive operations
5E+09
4E+09
3E+09
2E+09
1E+09
0
4
419
31537
3017399
300197640
2827085583
4232335411
4
419
31537
3017399
300197640
2827085583
4232335411
1
10
100
1000
10000
90000
100000
Value of n
Computer 1
Computer 2

**Figure 1.4**

**4.0 Discussion**

This paper conducts two experiments which are the runtime of an algorithm based on inputs and also the primitive operations based on inputs. For these two experiments, we observe some factors that affect the result of the outputs. The factors are as below:

a. Effect of different hardware on the analyses.
b. The different growth rates of each algorithm
c. Time complexity, $f(n)$ in terms of the number of inputs $n$.

Both of these experiments are conducted using two different machines/computers. We observe that the running time of an algorithm is highly affected by the speed of the computer itself for example the CPU (not only clock speed), I/O etc. Based on the previous results in tables and graphs, we can see that Computer 1 is faster than Computer 2 in processing and giving output of the algorithm. The running time recorded by Computer 1 is shorter than Computer 2. Below, we include the details of Computers that are used in these experiments:

**Computer 1:**

| | |
|---|---|
| Processor | Intel(R) Core(™) i5-8400 CPU @ 2.80GHz 2.81 |
| Installed RAM | 16 GB |

**Computer 2:**

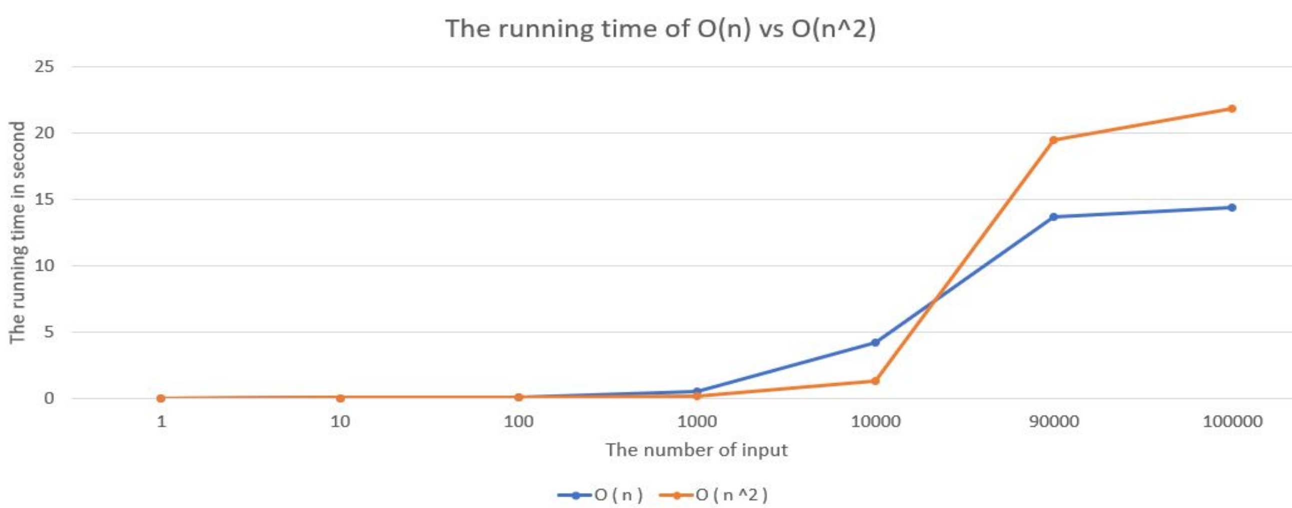
| | |
|---|---|
| Processor | Intel(R) Core(™) i3-6006 CPU @ 2.00GHz 1.99 |
| Installed RAM | 4.00 GB (3.90 GB usable) |

We also found that the runtime of an algorithm depends on the location of the program being stored. From our observation, we found that if the program is stored in the file containing many things in it, the runtime of the algorithm becomes slower. After the program's location has been changed into the file with more space, the runtime becomes faster. Other than that, a program can be slow due to the competition with other users for the resources of the computer.

The rate of growth for an algorithm is the rate at which the algorithm's cost increases as its input size rises. A growth rate of the first algorithm which finds a minimum and maximum value in an array is $n$ or it can be referred to as linear growth rate(*8.3. Comparing Algorithms — OpenDSA Data Structures and Algorithms Modules Collection*, n.d.). The second algorithm which is selection algorithm containing a factor of $n^2$ is said to have a quadratic growth rate. As you can see from Experiment 1 which are figure 1.1 and figure 1.2, the difference between an algorithm whose running time cost $T(n)=an+b$ and another with running time cost $T(n)= an^2 + bn + c$ becomes immense as $n$ grows. O( $n^2$ ) is larger than O($n$) which means that O( $n^2$ ) is less efficient than C. Some algorithms for problems of a given size of $n$ do not always take the same amount of time. In general, best case performance is not a good measure. It can be very difficult to find the average case so we normally focus on the run time of worst case performance. The algorithm that runs in O($n$) time is better than one that runs in O( $n^2$ ) time. Both algorithms have different runtime even though the same inputs are used.(*Running Time of Algorithms | HackerRank*, n.d.)

In experiment 1 we can conclude that for Linear and Quadratic algorithm: from Computer 1-

| The value of input, n | Runtime of O ( $n$ ) in seconds | Runtime of O ( $n^2$ ) in seconds |
|---|---|---|
| 1 | 0 | 0 |
| 10 | 0.001 | 0.001 |
| 100 | 0.004 | 0.011 |
| 1000 | 0.476 | 0.111 |
| 10000 | 4.167 | 1.316 |
| 90000 | 13.607 | 19.453 |
| 100000 | 14.349 | 21.769 |




Based on the table, at n=1000 and n=10000, the Linear algorithm takes longer than the Quadratic algorithm to execute. But for n=90000, the Quadratic algorithm takes longer time, and for larger values of n, the Linear algorithm performs much better. The reason is because for large values of $n$, any function that has $n^2$ term will grow faster than a function whose leading term is $n$. The leading term is defined as the term with the highest exponent. Generally, algorithms with smaller leading terms can be a better algorithm for large problems. But, algorithms with larger leading terms might be better if we want to solve small problems. It depends on the details of the algorithms, the inputs, and the hardware.

In experiment 2, we analyze the algorithm based on primitive operations. The number of primitive operations would usually increase with the increase of the number of inputs, n. Quadratic algorithms are really practical with a small value of n but it will increase fourfold with a big value of n. Quadratic algorithms perform $n^2$ operations as it has two for loops. The first iteration of the loop probably uses one operation, the second iteration of loop might use 2 operations. Because of that, the number of primitive operations for quadratic algorithms is bigger than linear.

| The value of input, n | The number of primitive operations of O ( $n$ ) | The number of primitive operations of O ( $n^2$ ) |
|---|---|---|

| | | |
|---|---|---|
| 1 | 18 | 4 |
| 10 | 114 | 419 |
| 100 | 1020 | 31537 |
| 1000 | 10022 | 3017399 |
| 10000 | 100030 | 300197640 |
| 90000 | 900036 | 2.827E+09 |
| 100000 | 1000036 | 4.232E+09 |

To characterize the relation between the number of inputs, n and the number of primitive operations, we can use a function *f(n)*. The total number of primitive operations for *O(n)* and O( $n^2$ ):

O(n)

At least: f(n) =10n+8
At most: f(n) = 14n+8

O( $n^2$ )

At least: f(n) = $3n^2 + 13n - 14$
At most: f(n) = $7n^2 + 4n - 1$

Let T(n) be the worst-case time for these two algorithms.

a= time taken by the fastest primitive operation
b= time taken by the slowest primitive operation

Assume that O(n) executes with 14n+8 primitive operations in the worst case:

a(14n+8) <= T(n) <=b (14n+8)

Assume that O( $n^2$ ) executes with $7n^2 + 4n - 1$ primitive operations in the worst case:

a( $7n^2 + 4n - 1$) <= T(n) <=b ( $7n^2 + 4n - 1$)

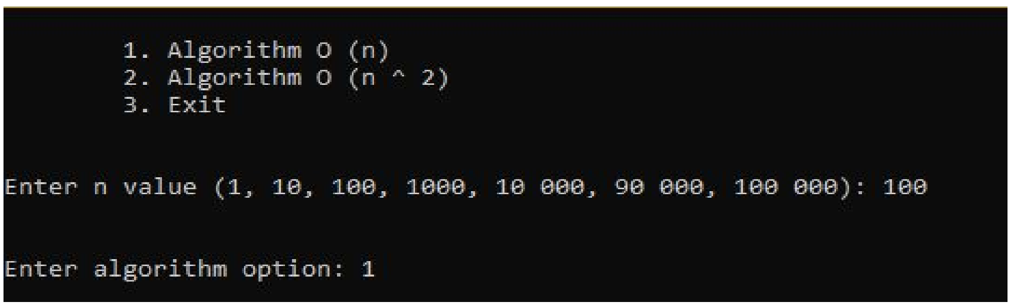
This shows that the running time of T(n) is bounded by two functions.

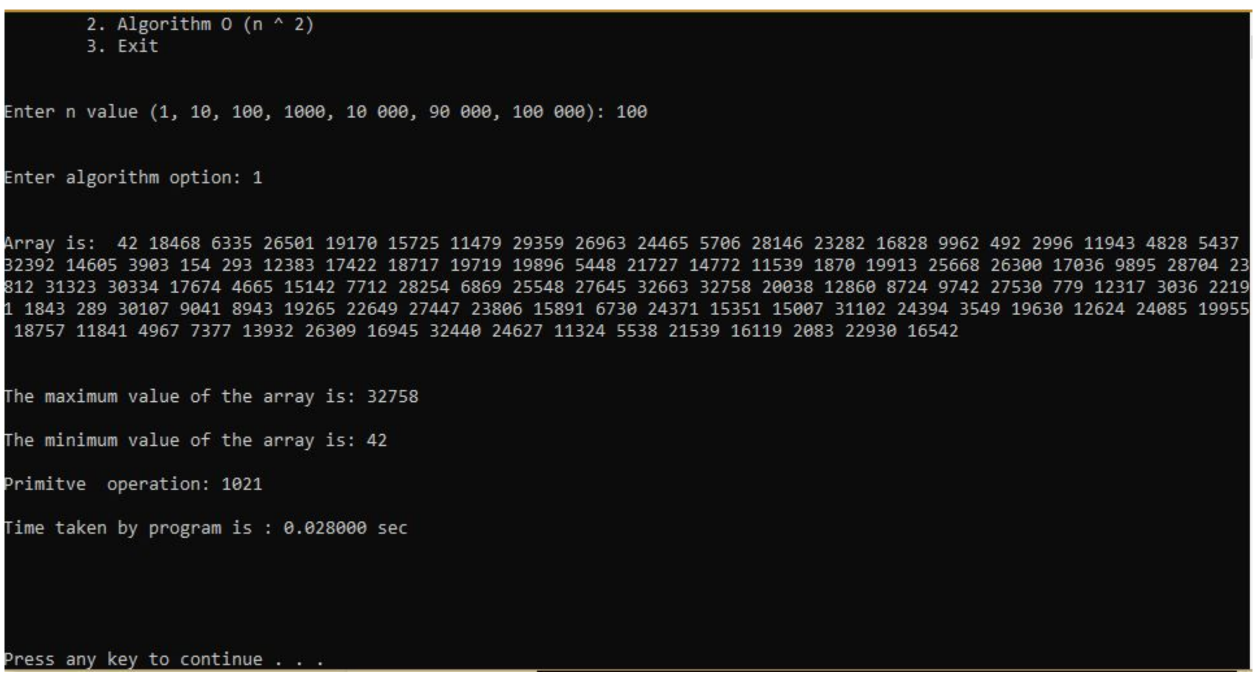
The arrangement of values in the array could affect the total number of primitive operations.The linear algorithm is asymptotically better than the quadratic algorithm although for a small value of *n*, the quadratic algorithm may have a lower running time than the linear algorithm. Typically, running times are characterized in terms of the worst case. Constructing or improving algorithms based on the worst-case time complexity usually leads to better algorithms.

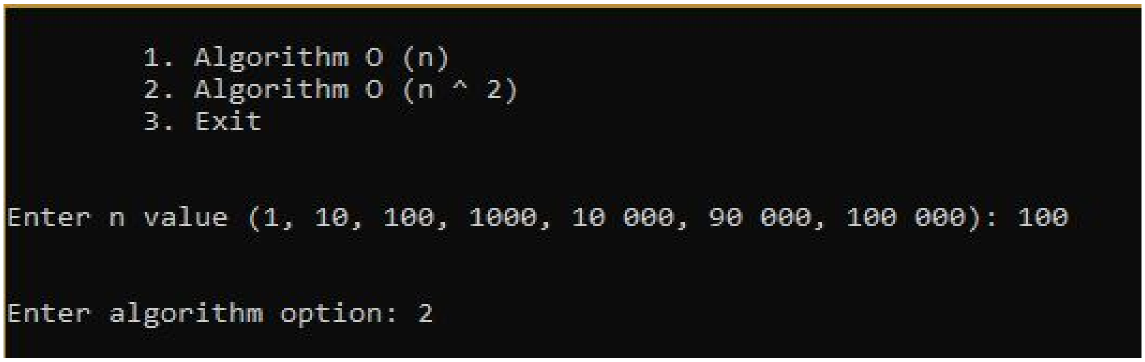
Based on what has been discussed above, we know that both Experiment 1 and Experiment 2 are carried out to analyze the algorithm and to measure the algorithms' performances. Experiment 1 implements the analytical method while Experiment 2 focuses on theoretical analysis. Both of the experiments are measuring the same thing but with different methods. Experiment 1 focuses on determining the dependency of running time on the size of the input.

Because of that, we test the algorithm with various sizes of input. The result will then be plotted in a graph so that it can be visualized. However, Experiment 1 has a few limitations where experiments can only be tested with a limited set of input. Besides, the result of Experiment 1 is highly influenced by the hardware and software environments. Because of that, theoretical analysis which is implemented in Experiment 2 has been carried out. As to cope with the constraints of analytical methods, Experiment 2 measures the performance of the algorithm by calculating primitive operations. Instead of determining the specific execution time of each primitive operation, we will count the number of primitive operations executed, and use this number t as a high-level estimate of the running time of the algorithm. Primitive operations counts actually correspond with the actual running time in a specific hardware and software environment.

### 5.0 Screenshots



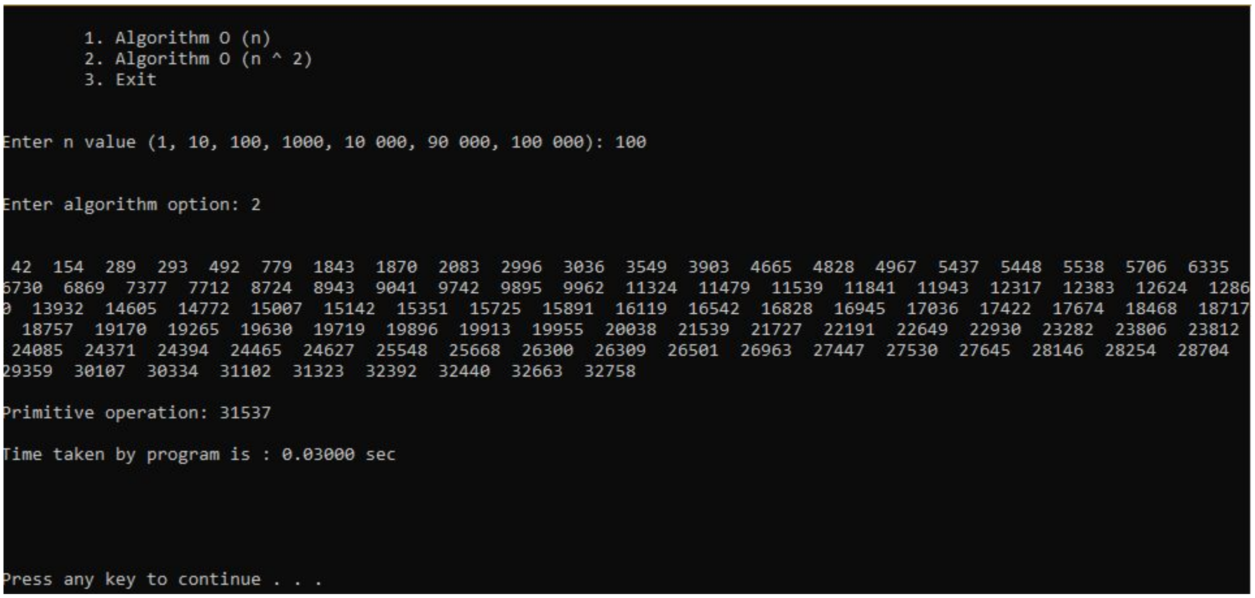





```
        1. Algorithm O (n)
        2. Algorithm O (n ^ 2)
        3. Exit

Enter n value (1, 10, 100, 1000, 10 000, 90 000, 100 000): 100

Enter algorithm option: 2
```



```
        1. Algorithm O (n)
        2. Algorithm O (n ^ 2)
        3. Exit

Enter n value (1, 10, 100, 1000, 10 000, 90 000, 100 000): 100

Enter algorithm option: 2

 42  154  289  293  492  779  1843  1870  2083  2996  3036  3549  3903  4665  4828  4967  5437  5448  5538  5706  6335
6730  6869  7377  7712  8724  8943  9041  9742  9895  9962  11324  11479  11539  11841  11943  12317  12383  12624  1286
0  13932  14605  14772  15007  15142  15351  15725  15891  16119  16542  16828  16945  17036  17422  17674  18468  18717
  18757  19170  19265  19630  19719  19896  19913  19955  20038  21539  21727  22191  22649  22930  23282  23806  23812
 24085  24371  24394  24465  24627  25548  25668  26300  26309  26501  26963  27447  27530  27645  28146  28254  28704
29359  30107  30334  31102  31323  32392  32440  32663  32758

Primitive operation: 31537

Time taken by program is : 0.03000 sec



Press any key to continue . . .
```

**6.0 References**


*8.3. Comparing Algorithms — OpenDSA Data Structures and Algorithms Modules Collection*. (n.d.). Retrieved April 12, 2020, from https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/AnalIntro.html

*Running Time of Algorithms | HackerRank*. (n.d.). HackerRank. Retrieved April 12, 2020, from https://www.hackerrank.com/challenges/runningtime/problem

Goodrich , M. T. G. (n.d.). *Algorithm Design and Applications*.