KING ABDULAZIZ UNIVERSITY

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

CPCS 223-Analysis & Design of Algorithms

Spring 2020

CPCS223-Spring2020-Project Report

Empirical Analysis of Unique Elements Algorithm

Name: Salwa Safwan Abbara

ID: 1780293

Section: HAR

Instructor: Dr.Muhammad Al-Hashimi

Contents

Tables

Figure

1. Introduction

A most common way of analyzing the efficiency of the algorithms is the mathematical analysis. Though these techniques can be applied successfully to many simple algorithms but not for all. Some simple algorithms have a difficult mathematical computation for analyzing; especially, for investigating the average case efficiency. Moreover, it only focuses on the basic operation. If we want to measure the physical implementation of the algorithm, we might need another way such as empirical analysis. In the following sections, the empirical analysis of two algorithms to determine if an array's elements are unique will be provided.

2. Empirical Analysis of Algorithm

2.1. The Experiment Purpose

The purpose of this experiment is to compare the efficiency of two approaches

to determine if an array's elements are unique: brute force approach and one based on a presort. Deciding which algorithm is faster and then comparing results with the theoretical assertion about the algorithm's efficiency.

2.2. The Efficiency Metric

There are two ways to find the algorithm's efficiency:

1- Counter: count the number of times the algorithm's basic operation

is executed.

2- Time: to measure time the program takes to execute.

In the first way, we need to check the basic operation execution in several places in the program by inserting a counter into a program implementing the algorithm. This way is a straightforward operation and more accurate because it is machine independent, Unlike the time.

In the second way, we time the program implementing the algorithm. This can be done by using some functions such that performance.nom() in JavaScript.

I chose the operation count as an efficiency metric for this study for some reasons, including the operation counter is straightforward operation and more accurate as it is machine independent. On the other hand, the time is typically not very accurate as it depends on many extraneous factors such as the difficulty of clocking the actual running time of the program, and the quality of a program implementing the algorithm and the compiler. besides, given the high speed of modern computers, the running time may fail to register fully and be reported as zero. Even if you run the same algorithms on the same device, other factors such as the network may affect it, and the result for the same inputs may be different from one run to another. the physical running time uses it to specific information about an algorithm's performance in a particular computing environment.

2.3 Characteristics of The Input Sample

The input in unique element algorithms represents the list of size (n).

2.3.1 size

I chose input size increase by a constant amount of 1000 and I took sizes starting at 2000 to 15000. I add some numbers out of this range to ensure that the inputs sample is typical. I decide to add 700,1000 to check the performance of odd size and small input size.

2.3.2 Range

About the range I chose different ranges for different cases (worst, average) by using fill-in method which generates different instance randomly.

actually, I choose different ranges for each case to ensure that the cases will apply. for example, In Brute force method I choice range from 0 to 4294962926(0xffffeeee in hexa)to get the worst case when no value will be duplicated because the biggest range will ensure that we will never have the same number more than once and range from 0 to 268435182 to get the average case when I have probability to get all possible inputs. In the presort unique element algorithm I choice range from 0 to 8 for the worst case the reason for that range that to have all elements almost the same so for any pivot chosen will be the worst and from 0 to 500 for the average case to get all possible inputs which include the cases near the worst.

2.4. Algorithms

We have 2 approaches for a unique element algorithm. Based on the brute force approach and one based on a presort.

2.4.1 Based on brute force approach

From the textbook - Introduction to The Design and Analysis of Algorithms –, Chapter 2,

Section 2.3, ALHORITHM UniqueElements.

2.4.2 Based on presort

Algorithm UniqueElements

//Determines whether all the elements in a given array are distinct

//Input: An array A[0..n − 1]

//Output: Returns “true” if all the elements in A are distinct

// and “false” otherwise

1: Sort A //by quick sort

2: for i ⟵ 0 to n-2 do

3: if A[i]=A[i+1] then

4: return false

5: return true

2.4.3 Quick sort

From Dr. Muhammed Al-hashimi website lecture 0

2.5. Generate a Sample of Inputs

As I mentioned above, I generate the sample of inputs by use build function Math.random to generate random numbers in the specific range I discussed in section 2.3.2.

2.6. Data Observed from Running the Algorithm

2.6.1. Tools

1- Firefox browser to run the code

2- JavaScript language using to implement the code

3- Excel sheet to record data, compute the average and draw graphs.

2.6.2. Observed Data for Counter metric

The following table contains records of 10 inputs for brute force algorithm followed by graph representing these data by using range from 0 to 4294962926 to fill the list.

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The worst case
700	244650	244650	244650	244650	244650	244650	244650	244650	244650	244650	244650
1000	499500	499500	499500	499500	499500	499500	499500	499500	499500	499500	499500
2000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000
3000	4498500	4498500	4498500	4498500	4498500	4498500	4498500	4498500	4498500	4498500	4498500
4000	7998000	7998000	7998000	7998000	7998000	7998000	7998000	7998000	7998000	7998000	7998000
5000	12497500	12497500	12497500	12497500	12497500	12497500	12497500	12497500	12497500	12497500	12497500
6000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000
7000	24496500	24496500	24496500	24496500	24496500	24496500	24496500	24496500	24496500	24496500	24496500
8000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000
9000	40495500	40495500	40495500	40495500	40495500	40495500	40495500	40495500	40495500	40495500	40495500
10000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000
11000	60494500	60494500	60494500	60494500	60494500	60494500	60494500	60494500	60494500	60494500	60494500
12000	71994000	71994000	71994000	71994000	71994000	71994000	71994000	71994000	71994000	71994000	71994000
13000	84493500	84493500	84493500	84493500	84493500	84493500	84493500	84493500	84493500	84493500	84493500
14000	97993000	97993000	97993000	97993000	97993000	97993000	97993000	97993000	97993000	97993000	97993000
15000	112492500	112492500	112492500	112492500	112492500	112492500	112492500	112492500	112492500	112492500	112492500

Table 1: The worst case of Brute Force algorithm

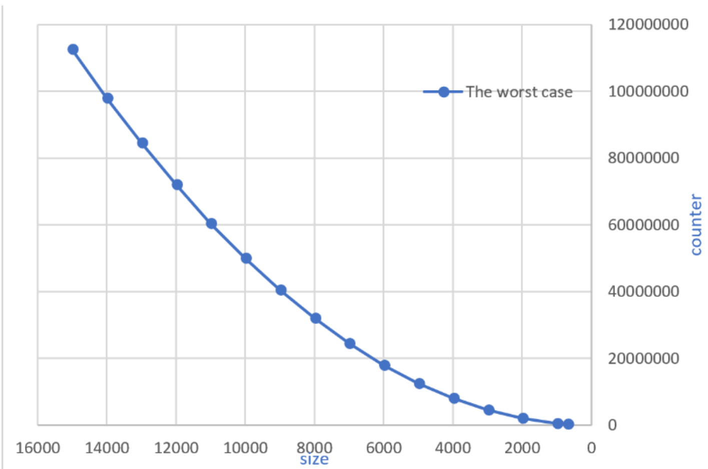

Figure 1: The worst case of Brute Force algorithm

The following table contains records of 10 inputs for brute force algorithm followed by graph representing these data by using range from 0 to 268435182 to fill the list.

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The average
700	244650	244650	244650	244650	244650	244650	244650	244650	244650	244650	244650
1000	499500	499500	499500	499500	391500	499500	499500	499500	499500	499500	488700
2000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000	1999000
3000	4498500	4498500	4498500	3022673	4498500	4498500	4498500	4498500	4498500	4498500	4350917.3
4000	7998000	7998000	6958230	7998000	7998000	7998000	7998000	6934825	7998000	7998000	7787705.5
5000	12497500	8735940	12497500	12497500	6534984	12497500	12497500	8735940	12497500	12497500	11148936
6000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000	17997000
7000	24496500	24496500	24496500	24496500	12742100	24496500	24496500	24496500	12742100	24496500	22145620
8000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000	31996000
9000	40495500	40495500	40495500	40495500	40495500	38016198	40495500	40495500	40495500	8289600	37026980
10000	49995000	49995000	49995000	49995000	47739000	49995000	49995000	8152423	49995000	49995000	45585142
11000	60494500	28703031	60494500	60494500	60494500	60494500	60494500	60494500	8152423	60494500	52081145
12000	71994000	1994000	71994000	71994000	71994000	71994000	71994000	71994000	71994000	58061179	63600718
13000	7621923	64916895	84493500	84493500	1163782	84493500	84493500	84493500	84493500	84493500	66515710
14000	22533816	70712254	14607454	97993000	97993000	50462558	48963034	97993000	97993000	97993000	69724412
15000	41998483	112492500	55141491	112492500	112492500	16686928	103754885	112492500	112492500	13336563	79338085

Table 2: The average case for brute force algorithm

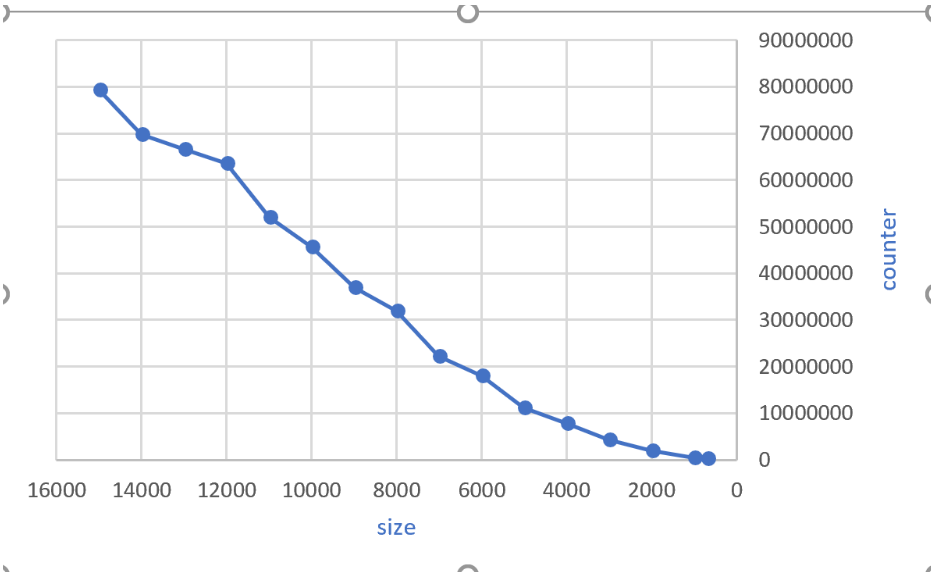

Figure 2: The average case for brute force algorithm

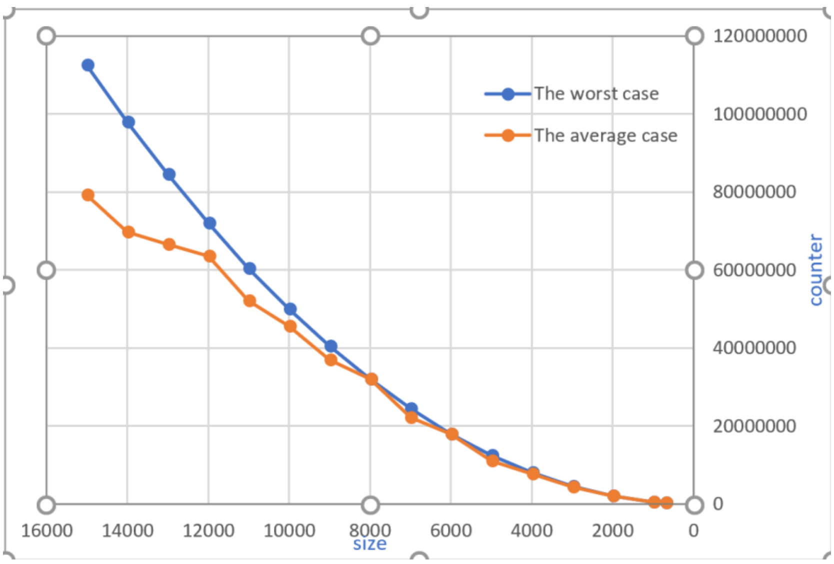

Figure 3: The worst and average case of brute force algorithm

The following tables contains records measure time that the program takes to execute of brute force algorithm. for each input I test it 10 times followed by graph representing these data by using range from 0 to 4294962926 to fill the list for the worst case and from 0 to 268435182 for average case.

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The worst case
700	0.4	0.1	0.5	0.4	0.3	0.5	0.5	0.6	0.3	0.5	0.41
1000	0.8	0.6	0.8	0.8	0.8	0.8	0.6	0.7	0.7	0.7	0.73
2000	2.7	2.8	2.8	3.2	3	2.7	2.9	2.7	3	2.8	2.86
3000	6.3	6.3	6.3	6.2	6.3	6.2	6.3	6.4	6.4	6.4	6.31
4000	11.2	11.2	11.1	11.1	11.1	11	11.1	11.3	11.3	11.3	11.17
5000	17.5	17.2	17.5	17.7	17.8	17.4	17.4	17.3	17.4	17.4	17.46
6000	24.8	25.6	25.1	24.9	24.9	25.1	24.7	24.6	24.8	25.7	25.02
7000	33.8	34.8	34.2	33.8	33.9	34	33.8	34.4	33.6	33.9	34.02
8000	44.8	45.8	45.4	45.4	47.6	44.8	44.6	46	43.7	44.8	45.29
9000	55.7	56.1	55.7	55.5	56	55.7	55.9	55.7	56.2	55.5	55.8
10000	69.5	68.6	69.9	69.7	71.6	68.3	69.1	69.4	68.8	68.9	69.38
11000	83.5	83.5	83.1	84.6	85.8	83.2	83.3	84.1	83	83.3	83.74
12000	99.4	101.2	101.8	100.5	100.4	100	98.7	99.1	99.3	99.2	99.96
13000	117	120.3	118.7	116.2	116.4	116.8	116.5	116	116.5	116.5	117.09
14000	135.4	143.3	137	138.8	138.1	140.1	137	137.8	137.6	138.1	138.32
15000	157.3	158.7	157.9	157.4	157.8	157.9	155.7	155.7	155.7	154.8	156.89

Table 3: The worst case of Brute Force algorithm by run time

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The average
700	0.4	0.2	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.2	0.31
1000	0.8	0.6	0.2	0.6	0.4	0.6	0.6	0.6	0.6	0.5	0.55
2000	2.7	2.4	2.4	2.5	2.3	2.2	2.5	2.1	2.5	2.5	2.41
3000	5.1	5.1	4.9	5.1	5.4	4.7	5.6	5.4	5.6	4.9	5.18
4000	9	9.2	9.1	12.1	9.3	9.1	9.1	9.1	9	8.8	9.38
5000	14.2	19.1	11	17.7	14.1	14.6	19.1	14.8	14.4	14.8	15.38
6000	19.7	23.2	19.8	21.8	25.1	23.7	23.8	22.7	21.5	21.5	22.28
7000	27.8	27	27.9	27.7	29.3	29.4	27.1	29.2	31.7	27.3	28.44
8000	35.7	39.3	37	0.5	35.4	35.4	35.9	36.9	1.9	32.2	29.02
9000	48.5	61.8	48.6	47.4	53.9	47.4	14.8	48.2	44.9	45.2	46.07
10000	55.1	55.3	40	56	55.4	55.8	31.5	59.3	55.7	59.3	52.34
11000	76.1	76.2	56.3	72.5	78.8	80.2	19.1	69.4	67	72.7	66.83
12000	79.3	82.9	79.1	82.1	80.5	35	81.5	80	80.1	83.1	76.36
13000	100.4	93.3	94.9	101.2	94.2	93.1	100.8	94.2	94	1.2	86.73
14000	109.7	108.3	107.4	107.8	108.3	109	107.7	112.1	108.2	26.9	100.54
15000	124.9	73.3	131.2	120.9	124	123.2	134.5	132.7	125	137	122.67

Table 4: The average case for brute force algorithm by run time

The following tables contain records of 10 tests of each input of presorted algorithm followed by graph representing these data by using range from 0 to 8 to fill the list for the worst case and range from 0 to 500 for the average case.

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The worst case
700	29811.8	29864.7	29582.1	29652.3	29446.8	29802.1	29615.3	29651.8	29654.5	29710.3	29679.17
1000	58769.3	58709.3	58678.3	59094.1	59644.7	59048.8	59181.2	59377.9	59375.7	59303.6	59118.29
2000	229648	229683.1	228929.8	230696	230534.7	228940.6	228528.5	228675	228840.1	228183.7	229265.95
3000	509502.3	510369.5	509797.3	510425.6	510194.2	512007	511073.3	509853.3	509609.5	510045.7	510287.77
4000	902810	903038.9	903714.8	902620.4	902259.5	903484.3	904683.2	902770.7	904602.3	902909.2	903289.33
5000	1403503.6	1404187.7	1405083.9	1406895.7	1408448.1	1405837.9	1408318	1409369.7	1405389	1406710.7	1406374.4
6000	2020631.8	2020325.2	2019084.6	2022206.8	2022288.8	2020877.4	2020570	2021138.4	2022544.8	2020222.5	2020989
7000	2743763.9	2746070.8	2746909.5	2752282.9	2745899.2	2744142.9	2752051.5	2749678.7	2744333.8	2745258.6	2747039.2
8000	3585665.2	3586834	3580896.7	3584451.7	3582774.1	3583434.9	3585171.1	3582443.9	3581356.6	3587409.6	3584043.8
9000	4534472.5	4534017.3	4530201.8	4537410.3	4536645.7	4531070	4536371.9	4533301.6	4534569.4	4532894.5	4534095.5
10000	5597124.8	5586443.9	5606356.3	5591414.6	5592689.6	5591576.1	5593905.4	5591541.6	5585785.2	5595752	5593259
11000	6762801	6760556.6	6765981.6	6757874.1	6761123.3	6759379.3	6762879.5	6760303	6757399.3	6766485.4	6761478.3
12000	8041400.7	8045895.4	8038208.7	8039184.9	8040408.7	8044901.5	8048717	8037718.7	8042849.2	8038282.9	8041756.8
13000	9435312.7	9434765.7	9435881.2	9434226.4	9440960.7	9437622.8	9439903.6	9434214.4	9434996.5	9436393.8	9436427.8
14000	10936388	10940128	10936570	10938610	10931256	10943010	10938490	10934662	10942442	10937247	10937880
15000	12548567	12548990	12553920	12549653	12545468	12554340	12551278	12558620	12549089	12559790	12551971

Table 5: The worst case of presorted algorithm

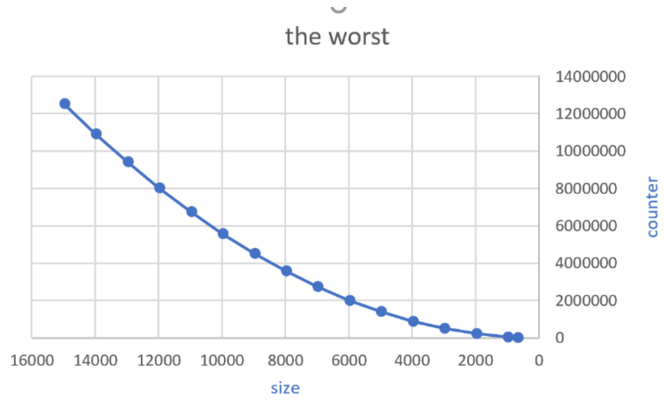

Figure 4: The worst of presorted algorithm

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The average
700	7394.9	7257.2	7537	7486	6967.6	7113.2	7229	7347.8	7284.8	7115.5	7273.3
1000	11263.6	11274.8	11099.4	11338.6	11502.9	11346.5	11226.4	11568.2	11432.3	11375.1	11342.78
2000	28293.2	27635.2	27632.4	27606.4	26759.3	28084.9	27370.2	26861.2	27782.5	26975.3	27500.06
3000	46636.5	48453.5	48334.1	47102	47359.6	47569.4	47397.9	48111.4	46157.2	46444.7	47356.63
4000	70641.2	71280.6	73304.2	71938.2	71801.4	72191.8	71842.3	71664.7	70054.7	71227.8	71594.69
5000	98694.6	99374.6	98380.4	99976	99828	99706.7	100799.4	97764.9	98548.7	101736.4	99480.97
6000	128656.7	133254.2	131306	131419.3	129899	130315.6	132812.1	130405.5	129852.8	131214.6	130913.58
7000	166017.4	166009	169228.9	164730.7	166492.6	168551.3	165115.1	165274.5	166123.6	169457.2	166700.03
8000	204560	208907.6	205986.1	204702.8	204544.8	205743.4	207606.1	206686.1	206365.1	206759.3	206186.13
9000	247792.2	245073	248797.6	251269.3	251487	251662.2	246434.6	252072.6	248920.1	251328.8	249483.74
10000	295733.8	297459.7	296617.5	299933.5	296317.1	296887.6	295323.5	297188.3	300157.6	297084.4	297270.3
11000	344729.1	347479.3	347790.6	345580.8	348820.5	347033.8	343690.5	352095.5	349695	348585.9	347550.1
12000	402105	407419.2	403297.9	404123	405211.8	407211.6	403458.6	401126.3	402880	405278.4	404211.18
13000	457786.5	461265.3	464555.1	464167.4	459926.3	460418	463678.3	463099.8	462644.7	458010.9	461555.23
14000	528308.1	524639.3	524683.8	524598.3	523599	530226.2	523842.6	522853.3	523645.9	527471.2	525386.77
15000	590877.8	588002.2	594279.3	591185.4	590845.4	592492.8	590321	592500.7	591380.2	594488.9	591637.37

Table 6: The average case for presorted algorithm

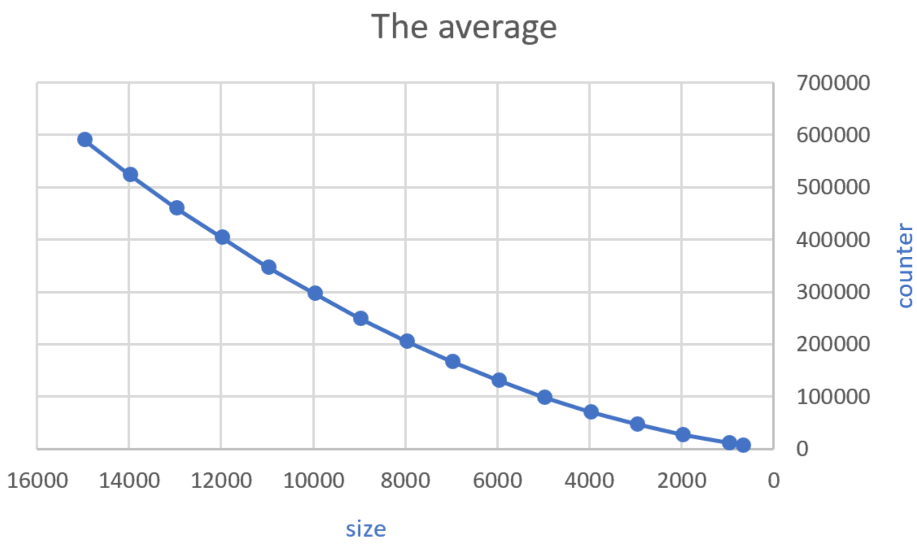

Figure 5: The average of presorted algorithm

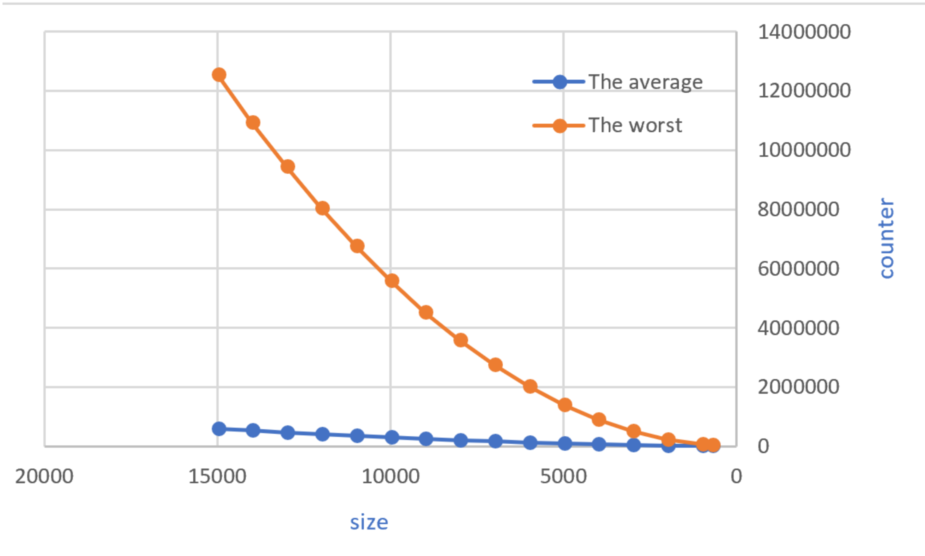

Figure 6: The worst and average case of presorted algorithm

The following tables contain records measure time that the program takes to execute of the presorted algorithm. for each input, I tested it 10 time followed by graph representing these data by using range from 0 to 8 to fill the list for the worst case and range from 0 to 500 for the average case.

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The average
700	0	0	0	0	0	0	0	0	0.2	0.4	0.06
1000	0	0.1	0.2	0.1	0	0.2	0.2	0	0.1	0.1	0.1
2000	0.4	0.4	0.4	0.4	0.4	0.3	0.4	0.4	0.4	0.4	0.39
3000	0.9	0.6	0.7	0.7	0.7	0.8	0.8	0.7	0.7	0.8	0.74
4000	1.1	1.2	1.1	1.3	1.4	1.2	1.3	1.1	1.1	1.3	1.21
5000	1.9	1.9	2	1.9	1.6	2	1.8	1.8	1.8	1.8	1.85
6000	2.8	2.5	3	2.8	2.5	2.5	2.6	2.6	2.5	2.5	2.63
7000	3.6	3.4	3.3	4.3	3.6	3.5	3.8	3.7	3.3	3.5	3.6
8000	4.6	4.6	5.6	4.6	4.6	4.6	4.7	4.6	4.5	4.5	4.69
9000	5.7	5.5	5	6.3	5.7	5.8	5.7	6.4	5.8	5.5	5.74
10000	7.1	10	7.5	6.1	6.6	7.5	7.1	7.3	6.7	7.1	7.3
11000	8.5	10.3	8.5	8.5	8	8.9	8.4	8.3	9.2	8.3	8.69
12000	10	12.2	10.5	13.7	10.3	10	9.4	10.4	10.1	9.6	10.62
13000	12	12.3	12.2	11.7	11.5	12.3	10.9	10.3	11.2	12.1	11.65
14000	13.6	14.1	14	18.9	12.8	13.3	13.1	13.3	13.8	13.3	14.02
15000	14.1	15.8	16.4	20.5	15.8	14.3	15.4	15.9	15.4	16.7	16.03

Table 7: the worst case of presorted algorithm by run time

input size	instance1	instance2	instance3	instance4	instance5	instance6	instance7	instance8	instance9	instance10	The average
700	0.1	0.2	0.1	0	0.1	0	0	0.1	0.1	0.1	0.08
1000	0	0	0	0.1	0	0	0.2	0.1	0	0	0.04
2000	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.12
3000	0.2	0	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.3	0.14
4000	0.2	0.1	0.3	0.2	0.3	0.1	0.2	0.1	0.1	0.2	0.18
5000	0.3	0.3	0.3	0.4	0.4	0.2	0.1	0.3	0.2	0.3	0.28
6000	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.4	0.4	0.2	0.37
7000	0.5	0.4	0.5	0.6	0.3	0.5	0.6	0.2	0.5	0.2	0.43
8000	0.5	0.6	0.5	0.7	0.6	0.6	0.3	0.5	0.5	0.5	0.53
9000	0.7	0.6	0.4	0.6	0.7	0.7	0.8	0.5	0.6	0.8	0.64
10000	0.7	0.4	0.7	0.8	0.7	0.6	1	0.7	0.6	0.8	0.7
11000	0.8	0.9	1	0.8	0.7	0.9	0.8	1	0.8	0.8	0.85
12000	0.8	1	0.9	1.1	1	0.8	1	1	0.9	0.9	0.94
13000	1	0.9	1	1	1	1	1	0.9	0.9	1.2	0.99
14000	1	1.1	1.2	1	1.3	1.3	1.1	1.3	1.1	0.9	1.13
15000	1.1	1.2	1.3	0.9	1.3	1.1	1.2	1	1.2	1.2	1.15

Table 8: the average case of presorted algorithm by run time

2.7 Analyze the data obtained

We noticed from figure1 that the brute force algorithm has an order of growth quadratic€Ө(n^2) in the worst case and the values of counter obtained in table 1 ensure that when we divide the counter of 2n by the counter of n and get the result equal to 4 which implies it is in order of growth quadratic (n^2)as we learn in section 2.1.

Input size	The counter	Counter(2n)/counter(n)
1000 2000	244650 1999000	1999000/499500 =4.002
3000 6000	4498500 17997000	17997000/4498500 =4.001
4000 8000	7998000 31996000	31996000/7998000 =4.001
5000 10000	12497500 49995000	49995000/12497500 =4.001
6000 12000	17997000 71994000	71994000/17997000 =4.001
7000 14000	24496500 97993000	97993000/24496500 =4.001

Table 9: proof the order of growth of brute force algorithm in worst case

From figure2 we can see that the brute force algorithm has an order of growth close to quadratic≈€Ө (n^2)in the average case and the values of counter obtained in the table2 ensure that when we divide the counter of 2n by the counter of n and get the result close to 4.

Input size	The counter	Counter(2n)/counter(n)
1000 2000	488700 1999000	1999000/488700 =4.01
3000 6000	4350917.3 17997000	17997000/4350917.3 =4.1
4000 8000	7787705.5 31996000	31996000/7787705.5 =4.1
5000 10000	11148936 52081145	52081145/11148936 =4.4
6000 12000	17997000 63600718	63600718/17997000 =3.5
7000 14000	22145620 69724412	69724412/22145620 =3.6

Table 10: proof the order of growth of brute force algorithm in average case

From figure4 we can see that the presorted algorithm has order of growth quadratic€ϴ(n^2) in the worst case and the values of counter obtained in the table5 ensure that when we divide the counter of 2n by the counter of n and get the result equal to 4.

Input size	The counter	Counter(2n)/counter(n)
1000 2000	59118.29 229265.95	229265.95/59118.29 =3.9
3000 6000	510287.77 2020989	2020989/510287.77 =4
4000 8000	903289.33 3584043.8	3584043.8/903289.33 =4
5000 10000	1406374.4 5593259	5593259/1406374.4 =4
6000 12000	2020989 8041756.8	8041756.8/2020989 =4
7000 14000	2747039.2 10937880	10937880/2747039.2 =4

Table 11: proof the order of growth of presorted algorithm in worst case

From figure5 we can see that the presorted algorithm has order of growth linearithmic €ϴ(nlogn) in the average case and the values of counter obtained in the table6 ensure that when we divide the counter of 2n by the counter of n and get the result slightly more than two as we learn in section 2.1. Actually, we get result slightly more than nlogn

Input size	The counter	Counter(2n)/counter(n)
1000 2000	11342.78 27500.06	27500.06/11342.78 =2.4
3000 6000	47356.63 130913.58	130913.58/47356.63 =2.7
4000 8000	71594.69 206186.13	206186.13/71594.69 =2.8
5000 10000	99480.97 297270.3	297270.3/99480.97 =2.8
6000 12000	130913.58 404211.18	404211.18/130913.58 =2.8
7000 14000	166700.03 525386.77	525386.77/166700.03 =2.9

Table 12: proof the order of growth of presorted algorithm in average case

We can conclude from table3 that the brute force algorithm has an order of growth quadratic by the run time€ϴ(n^2) in the worst case when we divide the run time of 2n by the run time of n and get the result equal to 4 which is mean it is in order of growth quadratic.

Input size	run time	run time (2n)/ run time (n)
1000 2000	0.73 2.86	2.86/0.73 =3.917
3000 6000	6.31 25.02	25.02/6.31 =4
4000 8000	11.17 45.29	45.29/11.17 =4
5000 10000	17.46 69.38	69.38/17.46 =4
6000 12000	25.02 99.96	99.96/25.02 =4
7000 14000	34.02 138.32	138.32/34.02 =4

Table 13: proof the order of growth of brute force algorithm in worst case by run time

We can conclude from table4 the brute force algorithm has an order of growth close to quadratic≈€ϴ (n^2) by the runtime in the average case when we divide the run time of 2n by the run time of n and get the result almost close to 4.

Input size	run time	run time (2n)/ run time (n)
1000 2000	0.55 2.41	2.41/0.55 =4.3
3000 6000	5.18 22.28	22.28/5.18 =4.1
4000 8000	9.38 29.02	29.02/9.38 =3.3
5000 10000	15.38 52.34	52.34/15.38 =3.7
6000 12000	25.02 99.96	99.96/25.02 =4
7000 14000	22.28 100.54	100.54/22.28 =4.1

Table 14: proof the order of growth of brute force algorithm in average case

We can conclude from table7 that the presorted algorithm has an order of growth quadratic≈€ϴ (n^2) by the runtime in the worst case when we divide the run time of 2n by the run time of n and get the result almost equal to 4.we can see that the counter more accurate than the time in values.

Input size	run time	run time (2n)/ run time (n)
1000 2000	0.1 0.39	0.39/0.1 =4
3000 6000	0.74 2.63	2.63/0.74 =3.6
4000 8000	1.21 4.69	4.69/1.21 =4
5000 10000	1.85 7.3	7.3/1.85 =4
6000 12000	2.63 10.62	10.62/2.63 =4
7000 14000	3.6 14.02	14.02/3.6 =3.9

Table 15: proof the order of growth of presorted algorithm in worst case by run time

We can conclude from table8 that the presorted algorithm has an order of growth linearithmic ≈€ϴ(nlogn)by the run time in the average case when we divide the run time of 2n by the run time of n and get the result slightly more than two. Actually, we get result slightly more than nlogn

Input size	The counter	Counter(2n)/counter(n)
1000 2000	0.04 0.12	0.12/0.04 =3
3000 6000	0.14 0.37	0.37/0.14 =2.6
4000 8000	0.18 0.53	0.53/0.18 =3
5000 10000	0.28 0.7	0.7/0.28 =2.6
6000 12000	0.37 0.99	0.99/0.37 =2.7
7000 14000	0.43 1.13	1.13/0.43 =2.7

Table 16: proof the order of growth of presorted algorithm in average case by run time

In the end, we concluded that the brute force algorithm has an order of growth quadratic in the worst case and close to quadratic in average case by the counter. The presorted algorithm has an order of growth quadratic in the worst case and linearithmic in the average case by the counter. Also, we have the same order of growth by the run time nearly.

Now, we can use the Limits for comparing orders of growth of two specific functions to determine which function has the higher, lower and the same order of growth.

Input size	$\lim_{x \to n} \frac{n \log n}{n^2}$
700	0.0135017
1000	0.0099658
2000	0.0054829
3000	0.0038502
4000	0.0029914
5000	0.0024575
6000	0.0020918
7000	0.0018247
8000	0.0016207
9000	0.0014595
10000	0.0013288
11000	0.0012205
12000	0.0011292
13000	0.0010512
14000	0.0009838
15000	0.0009248

As we see, all the values close to 0 which mean that the presorted algorithm in average case has an order of growth(nlogn) lower than the brute force algorithm(n^2).

Input size	$\lim_{x \to n} \frac{n^2}{n^2}$
700	1
1000	1
2000	1
3000	1
4000	1
5000	1
6000	1
7000	1
8000	1
9000	1
10000	1
11000	1
12000	1
13000	1
14000	1
15000	1

In the above table we can see that all the values equal 1 which means that the presorted algorithm in the worst case(n^2) has the same order of growth to the brute force algorithm(n^2).

2.8 Comparison to Theory

The mathematical analysis of brute force in the worst case=$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1 = \frac{(n-1)n}{2} €\Theta(n^2)$

In the average case €ϴ(n^2).

The mathematical analysis of presorted algorithm in the worst case €ϴ(n^2) and in average €ϴ(nlogn).

The empirical analysis leads us to the same result of the mathematical analysis.

3. Conclusion

Empirical analysis is an alternative way of the mathematical analysis and it is applicable to all algorithms while it is difficult in the mathematical computation to analyze, especially, for investigating the average case efficiency. We use the empirical analysis of two approaches to determine the order of growth of each approach. We find by using the table and scatterplot that the presorted algorithm faster than the brute force algorithm in the average case but the same efficiency in the worst case.