Tina Drew
ECE573
Prof. Jha

035006375
Spring 2018
Homework #1

Question 1

Code Implementation

The code was implemented in python based on code shown in slide 34 if the *fundamentals-ls.18* lecture from class.

Results

The results from the the implementation of the Naive and Sort and Binary Search approach are shown in table 1 and figure 1 below:

	Algorithm time in seconds	
N	**Naive (Brute Force)**	**Sort and Bin Search**
8	0	0
32	0	0
128	0.12	0.02
512	8.13	0.52
1024	63.12	2.22
4096	4121.91	43.19
4192	4459.24	45.65
8192	>10000000	187.58

Table 1: Resuts of “Naive” and “Sort and Binary Search” three-sum implementations.

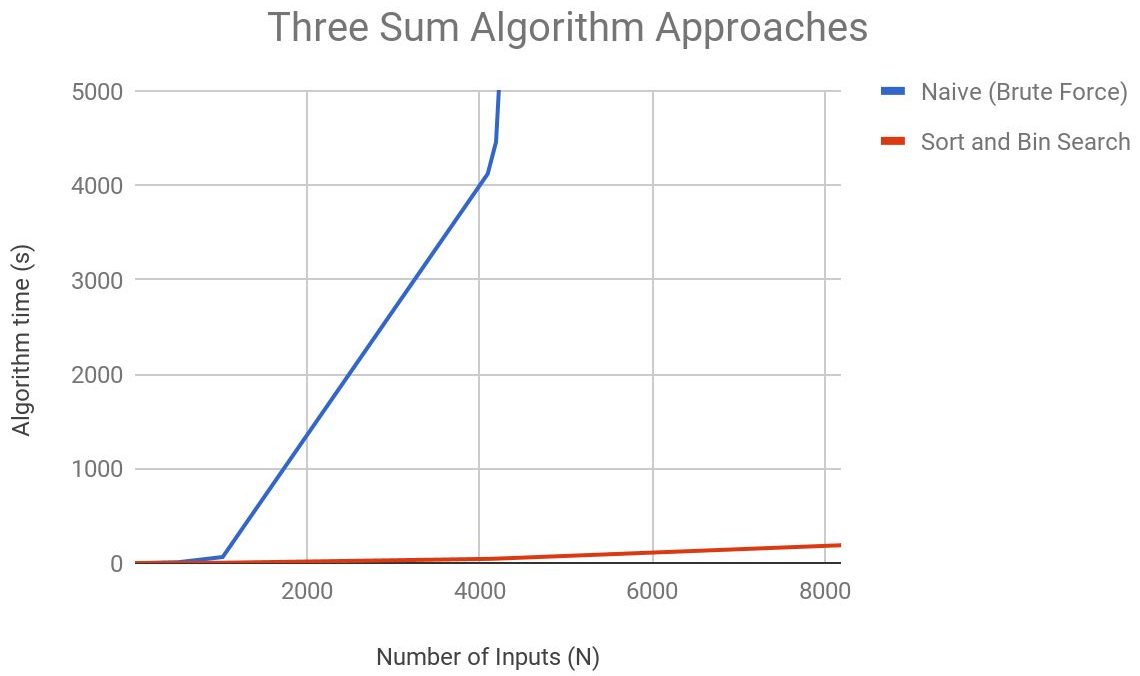


Figure 1: Plot of "Naive" and "Sort and Binary Search" three-sum implementations.

The "Naive" implementation shows exponential growth in time as the data size N increases. The algorithm is extremely expensive in terms of time. In fact I was not able to complete the trail for 8192 points. After running the algorithm for over 20 hours the program was not complete.

The "Sort and Binary Search " version of the three sum algorithm is more efficient than the Naive approach for larger sizes of N. For example the time cost for the the Sort and Binary Search is approximately 45 seconds whereas the Naive approach takes almost 45 seconds to complete execution. This means that Naive approach takes 10 times longer than the Binary Sort and Search approach.

Question 2

Code Implementation

The code was implemented in python. The Quick Union and Weighted Quick Union are run from the same file with the user being given the opportunity to select which version to run.

Results

The results from the the implementation of the Quick Find, Quick Union, and Weighted Quick union are in table 2 and figure 2 below:

	Algorithm time in seconds		
N	**Quick_Find**	**Quick_Union**	**Quick_Union_Weighted**
8	0.06	0	0
32	0.09	0	0
128	0.27	0	0
512	2.63	0	0
1024	1.13	0.42	0.49
4096	16.19	0.03	0.02
8192	49.59	0.89	0.94

Table 2: Results of Union algorithm implementations.

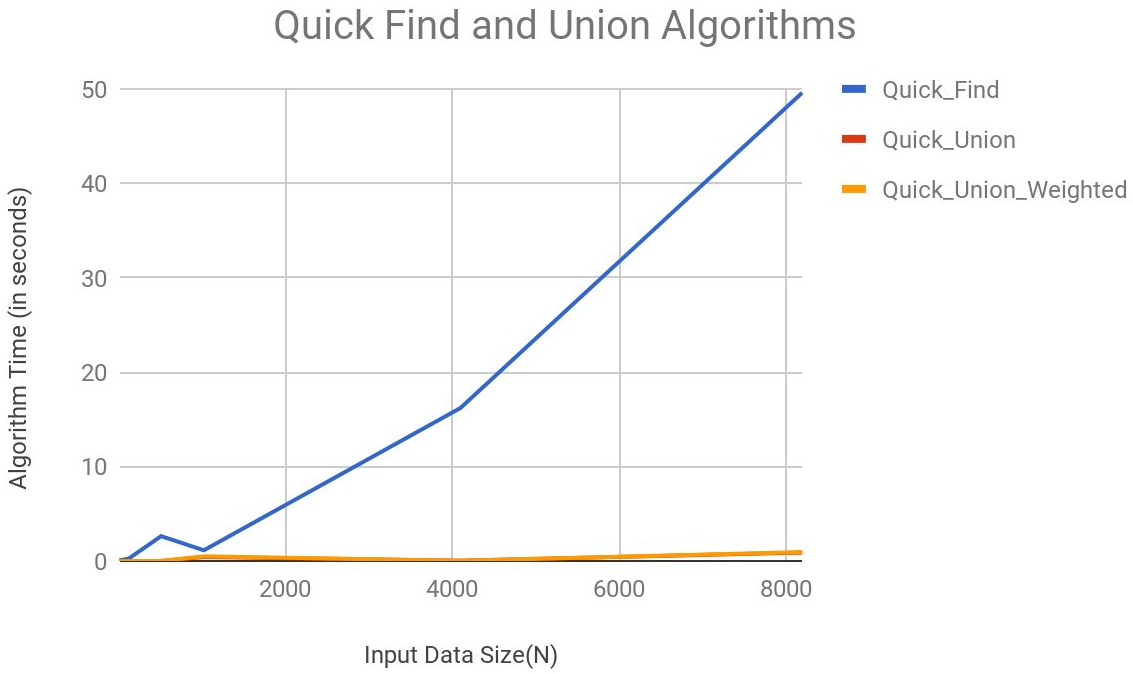


Figure 2: Plot the Union Algorithm results.

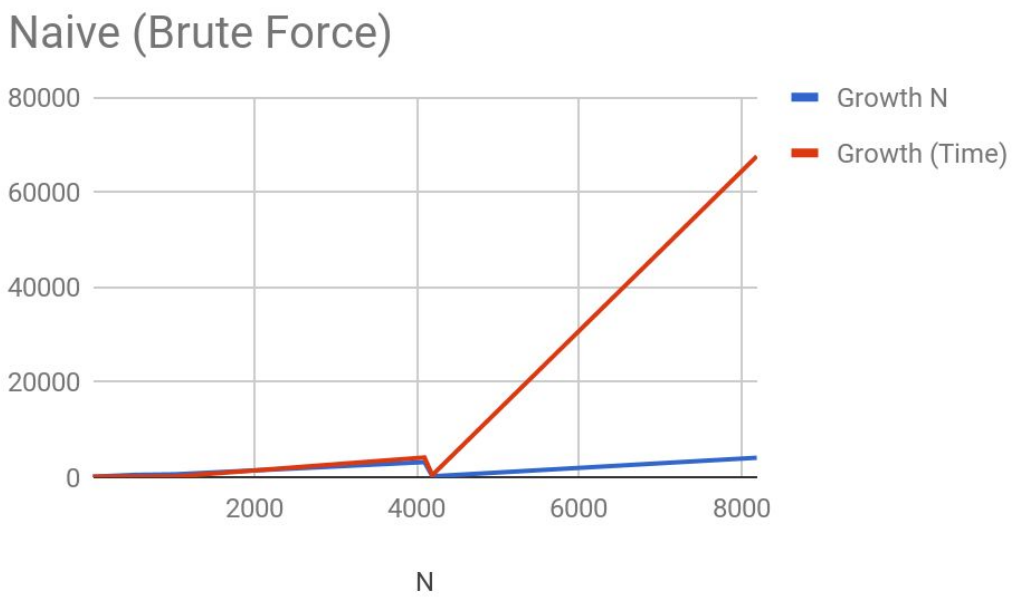
For this question three algorithm types were tested: Quick Find, Quick Union and Weighted Quick Union. Of the three algorithms the Quick Find algorithm had the quickest growth rate. The Quick Union and Weighted Quick Union algorithms show very similar cost in terms of time. What is interesting here is that there is a slight spike in the time cost for the Quick Find algorithms where there are 512 inputs.

Question 3

Response

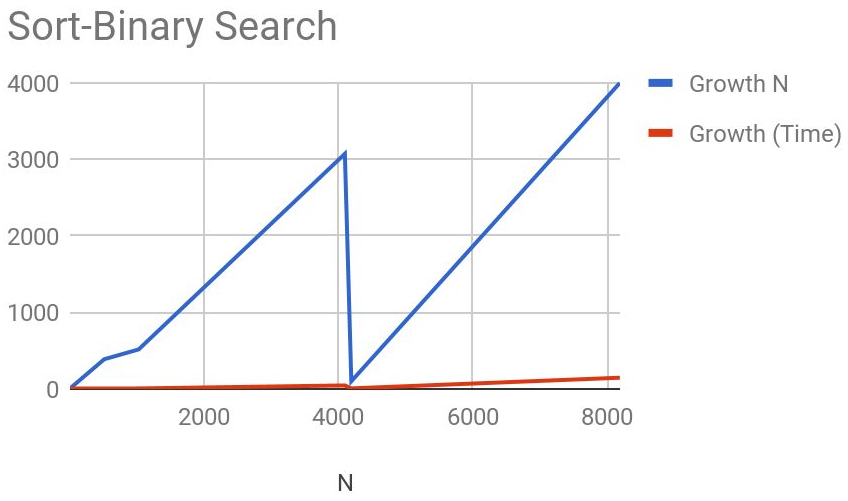
We were asked to find the value of Nc where: F(N) < c (g(N)), for N > Nc). To calcuate this I looked at the values of the data of the algorithm. I defined the growth of time as Tc- Tc-1 where Tc is the current value of time and Tc-1 is the previous value of time. Likewise I define the growth in N as Nc- Nc-1 where Nc is the current value of time and Nc-1 is the previous value of inputs. Based on this attained the following table:

N	Naive (Brute Force)	Growth N	Growth (Time)
8	0		0
32	0	24	0
128	0.12	96	0.12
512	8.13	384	8.01
1024	63.12	512	54.99
4096	4121.91	3072	4058.79
4192	4459.24	96	337.33
8192	72000	4000	67540.76



Nc is approximately 4000

N	Naive (Brute Force)	Growth N	Growth (Time)
8	0	8	0
32	0	24	0
128	0.02	96	0.02
512	0.52	384	0.5
1024	2.22	512	1.7
4096	43.19	3072	40.97
4192	45.65	96	2.46
8192	187.58	4000	141.93



Nc is > 8192

Question 4

Code Implementation

For this question we were asked to find the farthest pair of numbers in input list. I wrote code in python to generate text files with a list of floating point values. Files were created with the following data or input sizes: 8, 32, 128, 512, 1024, 4096, 4192, and 8192. The basic steps of my code are as follows:

1) Retrieve list of floating integers from file
2) Sort the list of integers
3) Use the first and last element of the array as the farthest pair
4) Calculate distance of the pair

Results

The results from the implementation of the Farthest pair algorithm are in table 3 and figure 3 below:

Farthest Pair Algorithm	
Number of inputs (N)	**Algorithm time (ms)**
8	0.00
32	0.00
128	0.00
512	0.00
1024	6.96
4096	15.62
4192	15.63
8192	46.46

Table 3: Results of Farthest Pair Implementation.

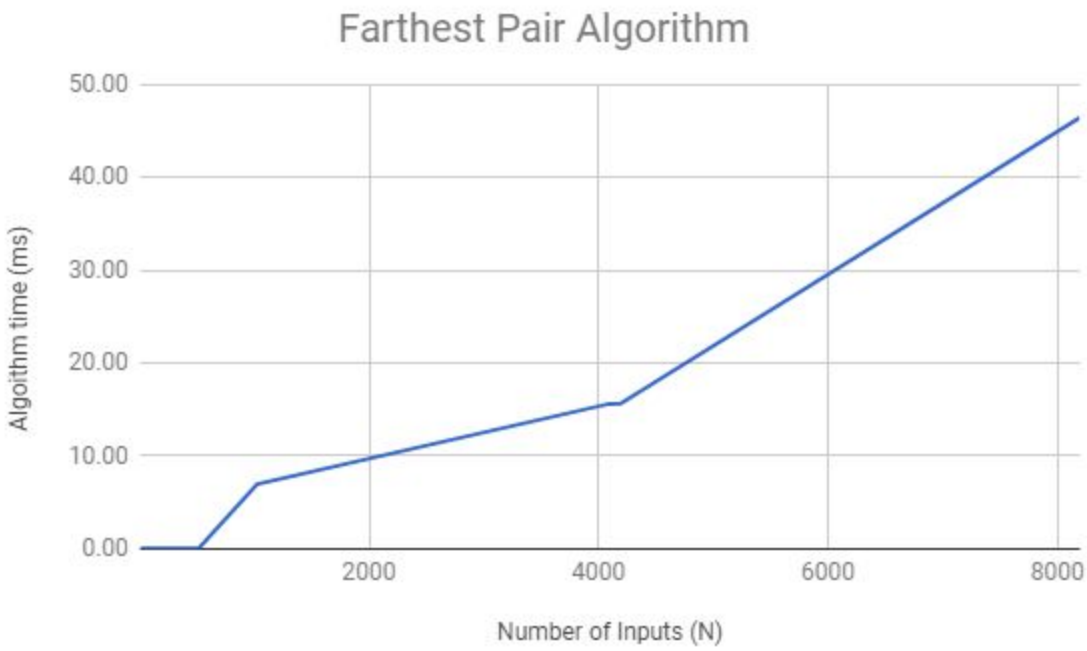


Figure 3: Plot the Farthest Pair Algorithm results.

Question 5

Code Implementation

The code for this code I reused part of code of the Sort and Binary Search algorithm for part 1. I replaced the binary search with a linear search algorithm. The basics of my linear search algorithm are listed below

1) Retrieve 2 values from the array
2) Add the two values retrieved and store as int ITEM
3) Sort the input list
4) Search the sorted input list for value = -ITEM
   (If current value is > - ITEM break)

Results

The results from the implementation of the Sort and Linear Search algorithm are in table 4 and figure 4 below:

Sort-Linear Search Algorithm	
N	**Algorithm Time (in seconds)**
8	0
32	0
128	0.01
512	0.14
1024	0.54
4096	0.88
4192	9.17
8192	36.1

Table 4: Results of Farthest Pair Implementation.

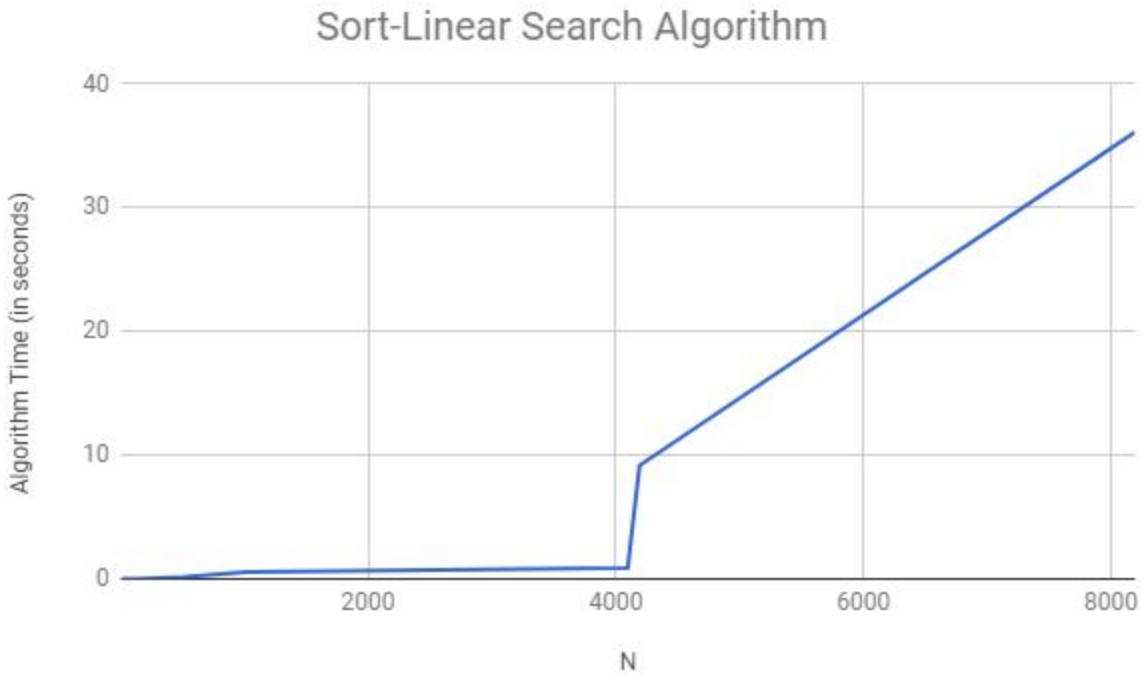


Figure 4: Plot the Sort-Linear Search Algorithm results.

This algorithm seems to be is relatively time efficient when compared to the Naive and Sort-Binary Search Algorithm. This is because all input values for this example were positive integers. Since the linear search stops the loop if value in the array is > -ITEM this algorithms ends on the first call of the loop. However, if array contains more negative values it could approach times close to that of the Naive approach.