Name: Naga Venkata V Vinnakota
Netid:nvv13
Course: ECE 16:332:573 DSA

Assignment-1

Q1. We discussed two versions of the 3-sum problem: A "naive" implementation (O(N^3)) and a "sophisticated" implementation (O(N^2 lg N)). Implement these algorithms. Your implementation should be able to read data in from regular data/text file with each entry on a separate line. Using Data provided under resources (hw1-data.zip) determine the run time cost of your implementations as function of input data size. Plot and analyze (discuss) your data.

Sol:

Problem statement:

A set of integer arrays from range 8-8192 are provided. We need to count the number of triplets that sum up to a given value. There are two implementations namely the Brute-force implementation, Sophisticated implementation generally used to find the three sums.

Brute-Force:

In Brute force, the triplet is found by accessing the array with three for loops and check if the sum is the required value in each iteration.
The complexity of BRUTE FORCE is O(N^3)

Sophisticated Implementation:

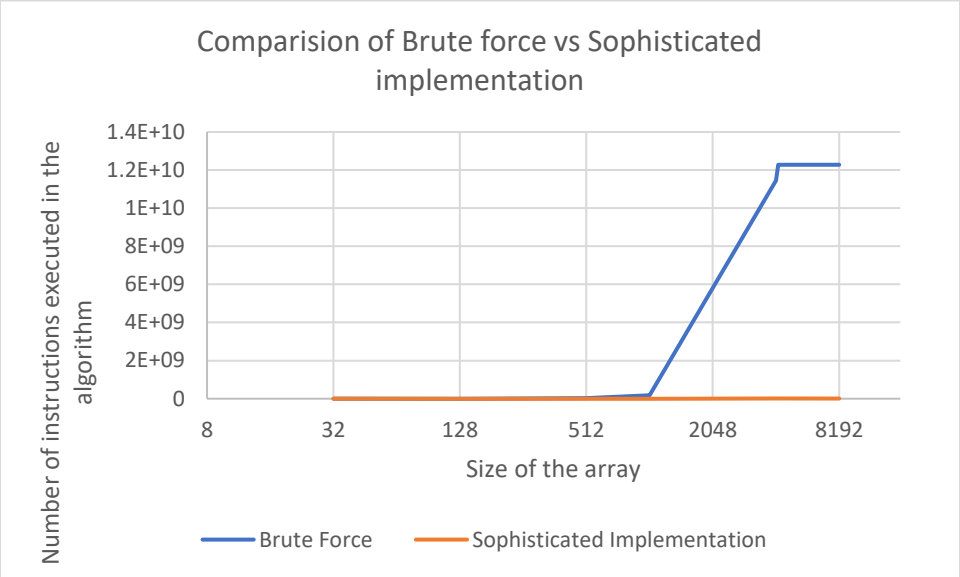
In Sophisticated implementation, First we need sort the array, then find the sum of pairs of numbers using two for loops and we need to do a binary search of the Triplet sum and the sum of each pair on the array. If the difference is found in the array, the count of the triplet increments by '1'.
The complexity of this implementation is O((N^2)*log(N))

The below table gives the number of statements executed in each of the algorithm

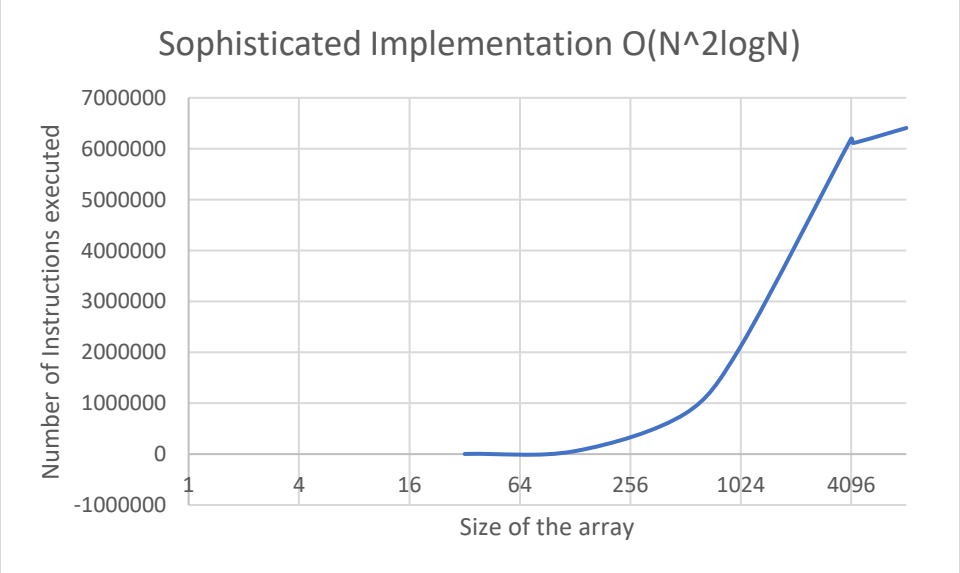
size of the array	Brute Force O(N^3)	Sophisticated Implementation O(N^2logN)
8	56	109
32	4960	2972
128	341376	59340
512	22238720	799073
1024	178433024	2116933
4096	11453249536	6194456
4192	12277577808	6110451
8192	12277577808	6407215

From the above table, It is clear that Brute force executes a very large number of instructions and the count increases exponentially with the size of the array, However, for Sophisticated Implementation even though the number of statements executed is increasing with the size of the array, it is far less than the exponential increase.

The below plot also explains the same.



The number of instructions executed for Sophisticated Implementation is almost negligible when compared to that of the Brute Force.



The above plot shows the Number of instructions executed in Sophisticated Implementation with increasing size of arrays.

Q2. We discussed the Union-Find algorithm in class. Implement the three versions: (i) Quick Find, (ii) Quick Union, and (iii) Quick Union with Weight Balancing. Using Data provided here determine the run time cost of your implementation (as a function of input data size). Plot and analyze your data. Note: The maximum value of a point label is 8192 for all the different input data set. This implies there could in principle be approximately 8192 x 8192 connections. Each line of the input data set contains an integer pair (p, q) which implies that p is connected to q. Recall: UF algorithm should

// read in a sequence of pairs of integers (each in the range 1 to N) where N=8192

// calling find() for each pair: If the members of the pair are not already connected

// call union() and print the pair.

Sol:

Problem statement:

In the Union-Find Algorithm, find is used to find if any two numbers in an array are connected, if not connected the union function connects the two numbers.

Total of three versions were implemented for this Union-Find algorithm.

Quick-Find:

In this version, the find is easy as it costs only '1' instruction to check if a pair is connected or not. But Union takes around N array Access.

Quick-Union:

In this version, find costs 'N' instructions where as Union costs only '1' instruction

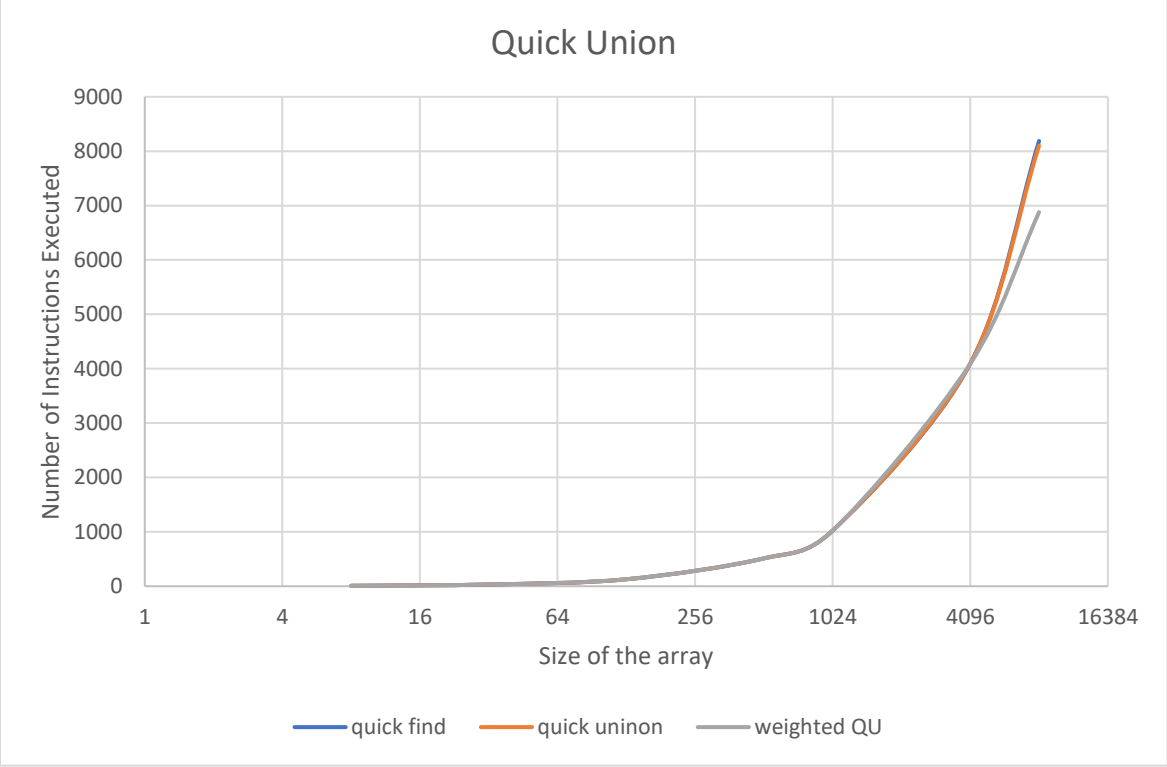
Weighted Quick-Union:

In this version, We track the size as the depth of the tree. The time complexity is the time taken to find the depth of the tree and M+logN

The below table gives the information of the number of instructions executed in each version as a function of size of the array.

size of the array	quick find	quick union	weighted QU
8	8	8	8
32	32	32	32
128	128	128	128
512	512	512	512
1024	1024	1024	1024
4096	4094	4092	4092
8192	8188	6878	6878

The below plot explains the same i.e. it represents the number of instructions executed for each version for a given array size



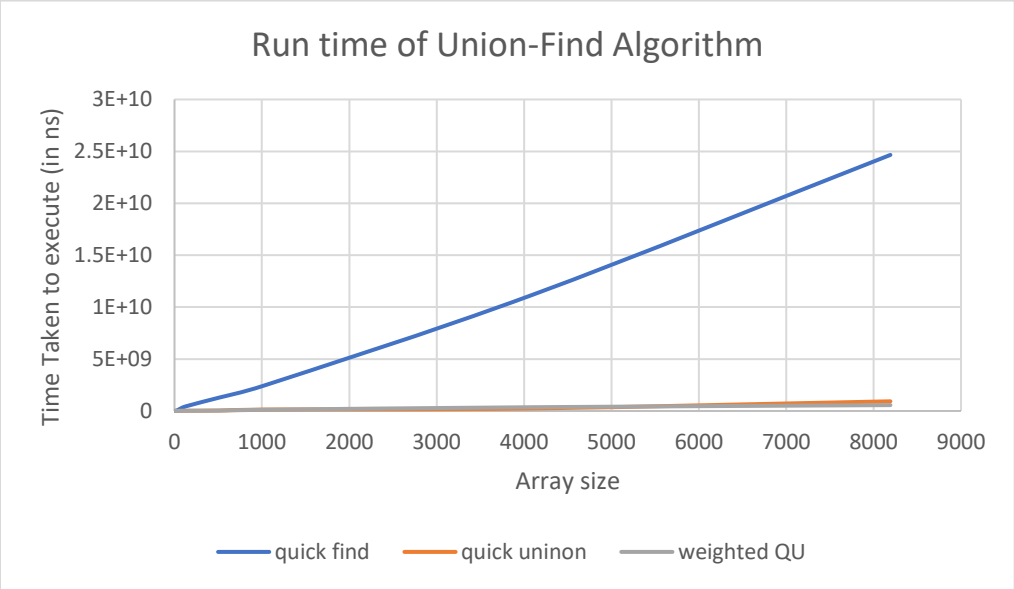
From the Plot it is clear that, the versions are almost same and weighted quick union has smaller value compared to quick-find and quick union

As all the versions have almost same number of instructions executed, We will verify if the Execution time is the same for all the versions.

The below table gives the runtime in nanoseconds for the three versions:

size of the array	quick find	quick uninon	weighted QU
8	15128800	1303600	2286300
32	74071600	3748600	5300000
128	445756300	26539100	21072100
512	1292773500	45373000	38737500
1024	2448455700	138708800	117861400
4096	11185088000	246701500	364232100
8192	24660035200	931812200	567698400

The same can be depicted from the graph;



The increase in runtime of quick find is too high compared to the other versions. Even though Quick-Union and Weighted-Quick Union seems to have almost same runtime, we can observe from the array size 4096 there is a significant increase in Quick-Union compared to weighted quick union. This difference is significant for the array size 8192.

Q3. Recall the definition of "Big Oh" (where F(N) is said to be in O(g(N), when F(N) < c (g(N)), for N > Nc) . Estimate the value of Nc for both Q1 and Q2. More important than the specific value, is the process and reasoning your employ.

Sol:

F(N)<c*g(N) for N>Nc

Analysis of Q1

Brute Force:
Here g(N) is N^3, so we can write the above equation as below,

F(N) <c*N^3 for N>NC
N^3<c*N^3
(or)
(N-2)*(N-2)*(N-2)<c*N^3 [Since each for loop is from range 0 to N-2, 1 to N-1, 2 to N, i.e. number of access = N-2]

For any value of c>1, Nc is 1

Sophisticated Implementation:

Here, g(N) is (N^2)*lgN

(N-1)*(N-1)*lgN < c*(N^2)*lgN [Since each for loop is from range 0 to N-1, 1 to N, i.e. number of access = N-1]

For c>1, N>1, say N=2 and c=2
log(2)<2*4*log(2)

Therefore, the value of Nc is 2 for any value of c>1

Analysis of Q2

Quick Find:

The find takes around one instruction and union costs N array access,
F(N)=N+1 and g(N)=N

N+1=c*N

For c>1.5, and n=2 we have
3<4

Therefore Nc=2

Quick Union:

Finding the root of each number takes around 'N' array access and 1 for Union

F(N)=N+N+1
G(n)=N

N+N+1<c*N
2N+1<c*N
For N=1
3<c
For N=2
5<2c
For N=3
7<3c

Therefore for c>=3, Nc=2

Weighted Quick Union:

Union takes lgN and find takes lgN

F(N)=2*lgN
C(n)=lgN

2*lgN<c*lgN

Therefore for c>=3, Nc will be 2

References:

"https://stackoverflow.com/questions/47872237/how-to-read-an-input-file-of-integers-separated-by-a-space-using-readlines-in-py/47872327" for reading data with spaces

"https://stackoverflow.com/questions/5998245/get-current-time-in-milliseconds-in-python" for getting time in milliseconds