Homework 1: Pratik Mistry – DSA Spring 2020 (pdm79)

***NOTE**: Please refer to Instructions.txt file uploaded on GitHub for running the program against the code files uploaded for the assignment questions*.

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.

Solution:

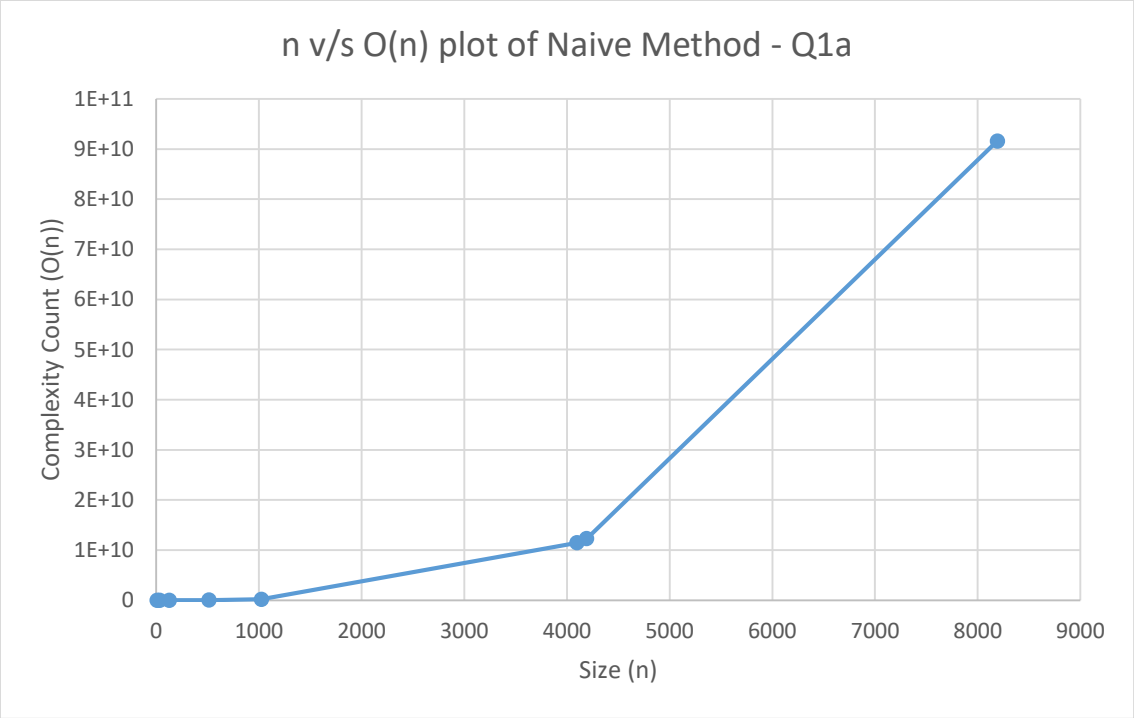
1a. Naïve Method/Approach for 3-sum:

Results: Time complexity for different size and also the Log-Log values of Time complexity and size

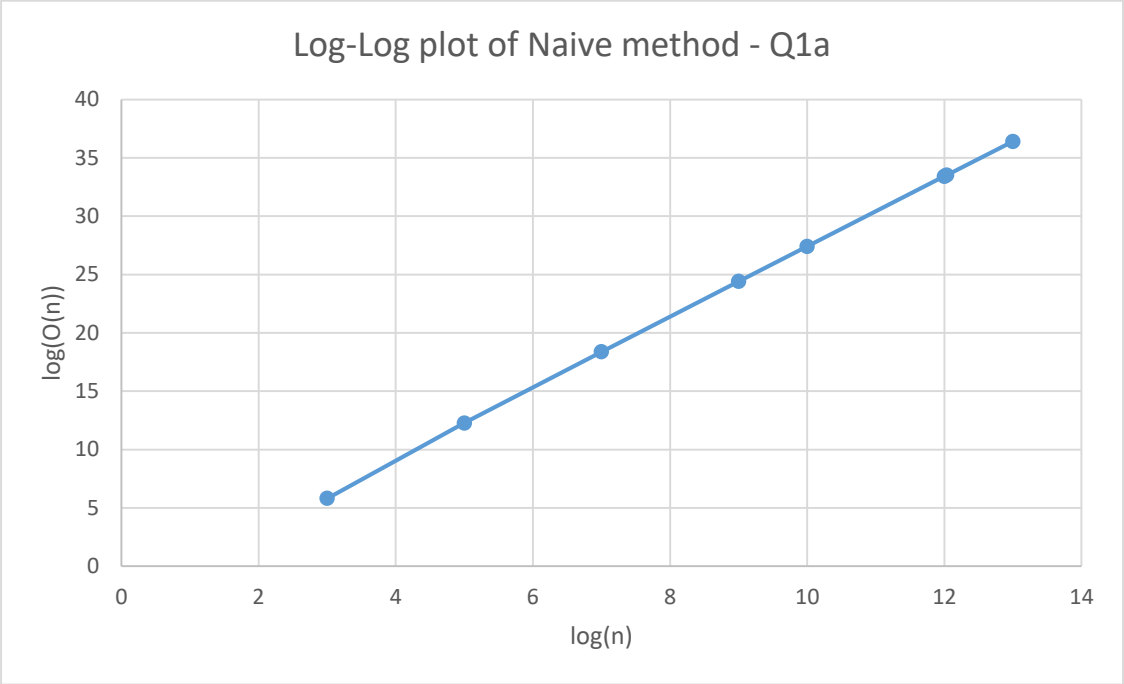
Size (n)	Time Complexity in terms of Complexity Counter(O(n)): n^3	Log(n)	Log(O(n))
8	56	3	5.807355
32	4960	5	12.27612
128	341376	7	18.381
512	22238700	9	24.40657
1024	178433000	10	27.41081
4096	11444900000	12	33.41399
4192	12268800000	12.03342	33.51428
8192	91592400000	13	36.41451

Graphs:

1. Size(n) v/s Complexity Counter ((O(N)) Plot



2. Log(n) v/s Log(O(n))) Plot



Explanation:

- The first implementation of the algorithm is by Naïve Approach i.e. Brute Force approach with the time complexity of order **n^3**. The first graph shows the growth order of the complexity which is cubic time.
- For better understanding and verification of the results, the second graph i.e. Log-Log plot can be referred with the values present in the above table.
- Theoretically, Log(O(n)) = 3Log(n) for Naïve method where O(n) = n^3. Thus, the equation is a straight line with slope 3 (theoretically)
- But, practically the value of slope is (say for n = 4192 and 8192):

(36.41-33.51)/(13-12) = 2.9 which is very close to theoretical slope i.e. 3

- Thus, the results with the graph clearly indicates that the Time complexity for the Naïve Method/Approach is **n^3.**

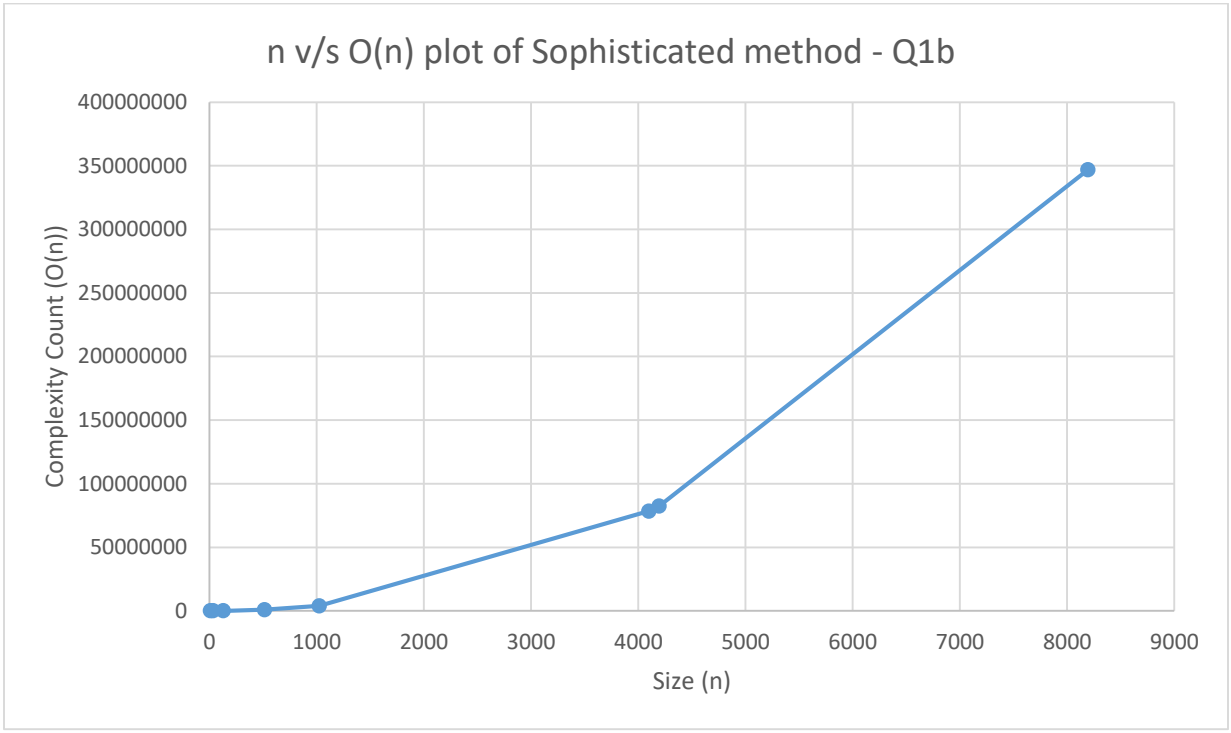
1b. Sophisticated Method/Approach for 3-Sum:

Results: Time complexity for different size and also the Log-Log values of Time complexity and size

Size (n)	Time Complexity in terms of Complexity Counter(O(n)): n^2log(n)	Log(n)	Log(O(n))
8	31	3	4.954196
32	1307	5	10.35204
128	36075	7	15.13871
512	832939	9	19.66785
1024	3850920	10	21.87677
4096	78322300	12	26.22292
4192	82310600	12.03342	26.29457
8192	346791000	13	28.36949

Graphs:

1. Size(n) v/s Complexity Counter ((O(N)) Plot



2. Log(n) v/s Log(O(n))) Plot

Explanation:

- The second implementation of the algorithm is by Sophisticated Method/Approach with the time complexity of order **n^2log(n)**. The first graph shows the growth order of the complexity.
- For better understanding and verification of the results, the second graph i.e. Log-Log plot can be referred with the values present in the above table.
- Theoretically, Log(O(n)) = 2Log(n) + Log(Log(n)) for Sophisticated method where O(n) = n^2Log(n). Thus, the equation is a straight line of form Y = mX + C where m is slope and it is 2 (theoretically)
- But, practically the value of slope is (say for n = 4192 and 8192):
  (28.37-26.3)/(13-12) = 2.07 which is very close to theoretical slope i.e. 2
- Thus, the results with the graph clearly indicates that the Time complexity for the Sophisticated Method/Approach is **n^2Log(n).**

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.

Solution:

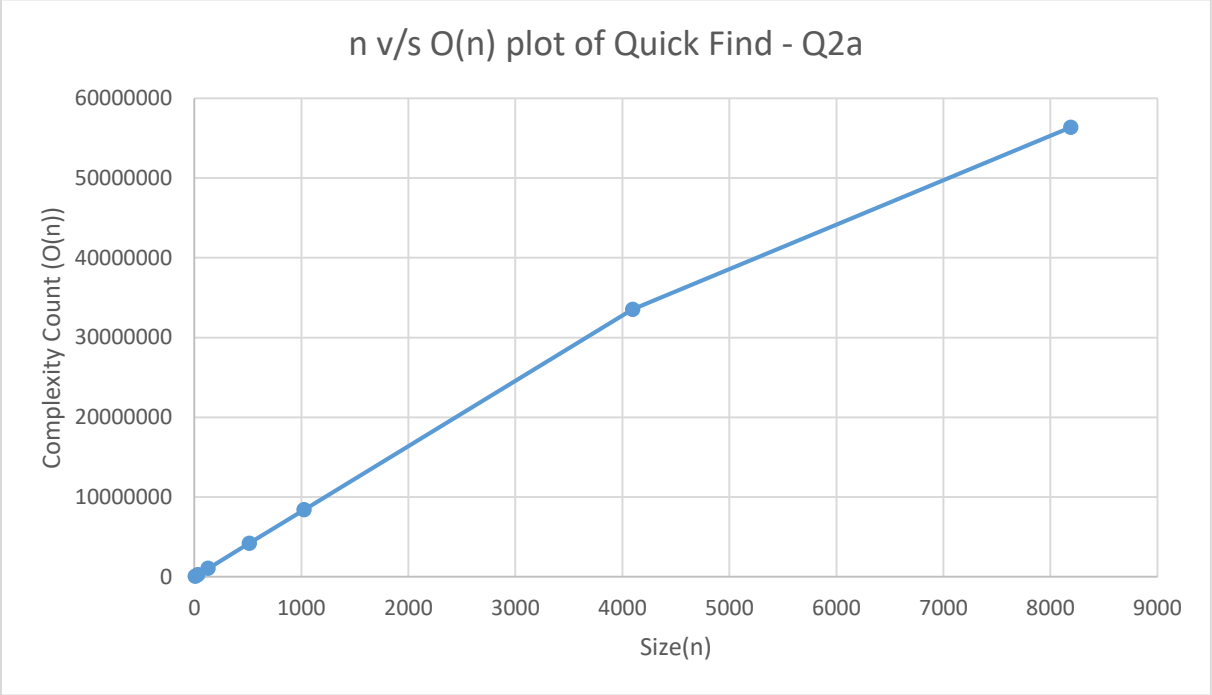
2a. Quick Find

Results: Time complexity for different size and also the Log-Log values of Time complexity and size

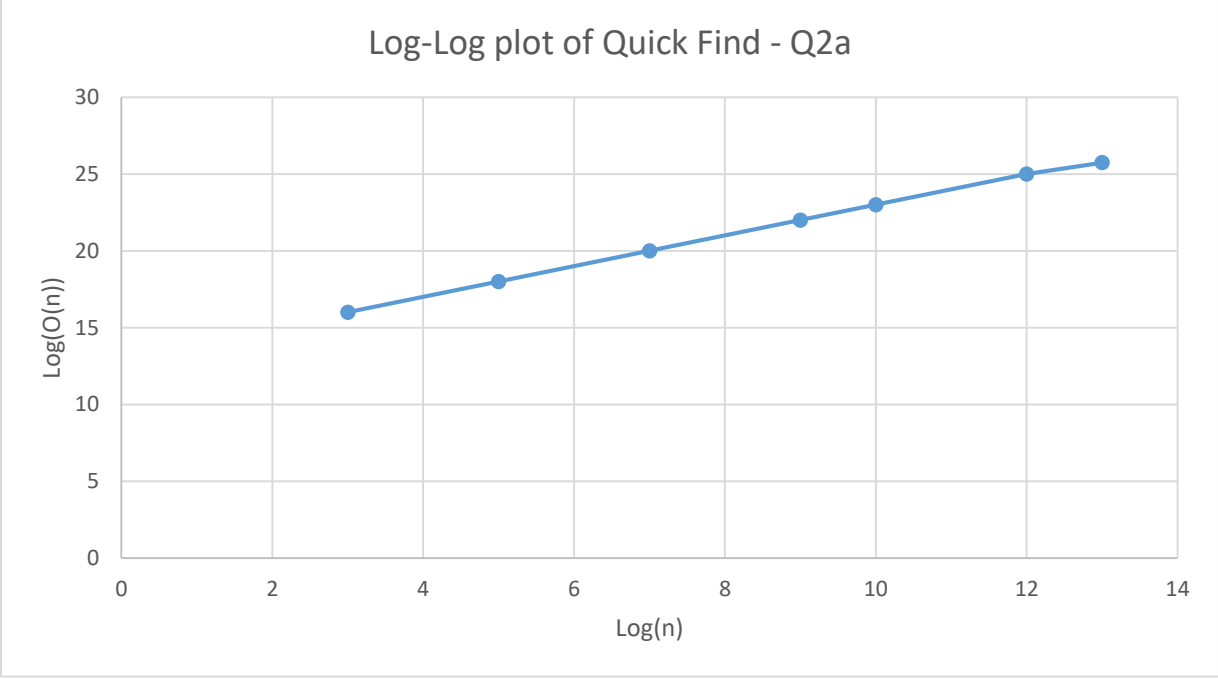
Size (n)	Time Complexity in terms of Complexity Counter(O(n))	Log(n)	Log(O(n))
8	65544	3	16.00018
32	262176	5	18.00018
128	1048700	7	20.00017
512	4194820	9	22.00018
1024	8389630	10	23.00018
4096	33525800	12	24.99877
8192	56352800	13	25.74798

Graphs:

1. Size(n) v/s Complexity Counter ((O(N)) Plot



2. Log(n) v/s Log(O(n))) Plot



Explanation:

- This is Quick Find algorithm whose time complexity includes cost of Find and Union Operators which is O(1) and O(n).

- But, since we find and perform union for "M" given input pairs with the maximum value of a point label i.e. N = 8192 for all the different input data set, the time complexity comes to O(M) + O(M*N) which is approximately **O(M*N)** for larger values of N i.e. we consider only cost of **UNION** operation.
- In the worst-case situation, if M = N i.e. input data pairs are 8192 then O(n) = **O(n^2)**.
- For better understanding and verification of the results, the second graph i.e. Log-Log plot can be referred with the values present in the above table.
- Theoretically, Log(O(n)) = Log(n) for a particular pair in dataset where O(n) = n. Thus, the equation is a straight line of form Y = mX where m is slope and it is 1 (theoretically)
- But, practically the value of slope is (say for n = 1024 and 4096):
  (25-23)/(12-10) = 1 which is very close to theoretical slope i.e. 1
- Thus, the results with the graph clearly indicates that the Time complexity for the Quick Find algorithm is **O(n)** in average case and **O(n^2)** in worst case situation.

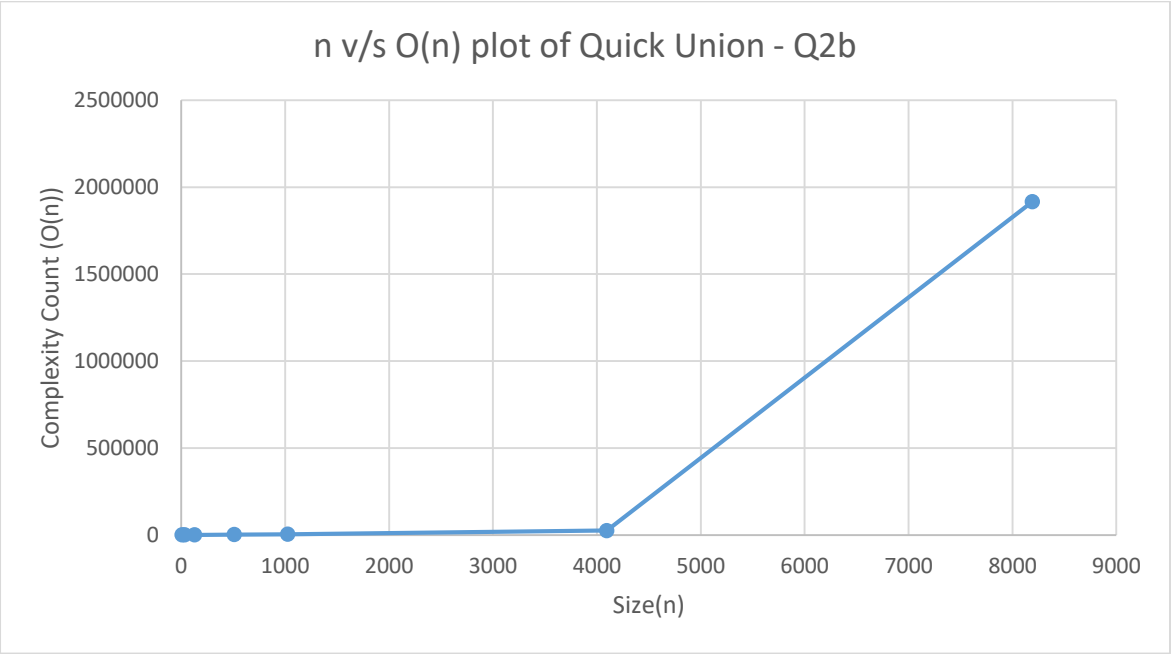
2b. Quick Union

Results: Time complexity for different size and also the Log-Log values of Time complexity and size

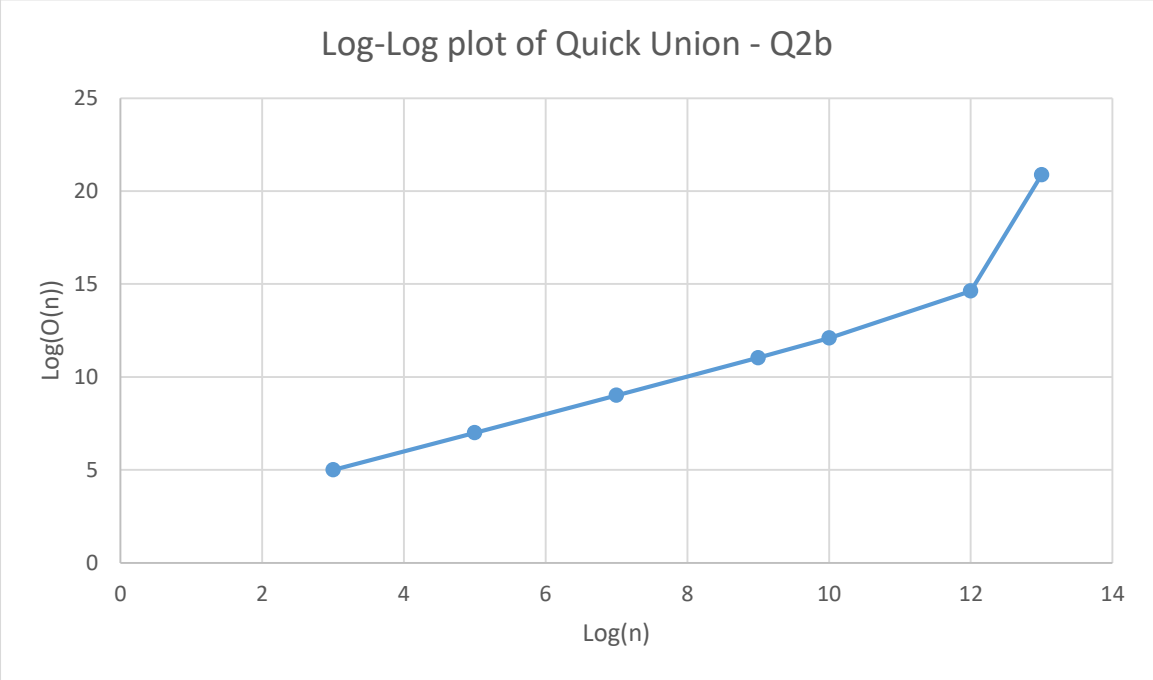
Size (n)	Time Complexity in terms of Complexity Counter(O(n))	Log(n)	Log(O(n))
8	32	3	5
32	128	5	7
128	516	7	9.0112
512	2104	9	11.039
1024	4362	10	12.091
4096	25313	12	14.628
8192	1916710	13	20.87

Graphs:

1. Size(n) v/s Complexity Counter ((O(N)) Plot



2. Log(n) v/s Log(O(n))) Plot



Explanation:

- This is Quick Union algorithm whose time complexity includes cost of Find and Union Operations which is indeed cost of finding the Root i.e. 2O(n) and 2O(n) since root for the pairs will be calculated in Find and Union operations.
- But, since we find and perform operations for "M" given input pairs with the maximum value of a point label i.e. N = 8192 for all the different input data set, the time complexity comes to 2O(M*N) + 2O(M*N) which is approximately **4O(M*N)** for larger values of N i.e. cost of finding ROOT in Find and Union operations.
- In the worst-case situation, if M = N i.e. input data pairs are 8192 then O(n) = **O(n^2)** (ignoring 4 which is constant).
- For better understanding and verification of the results, the second graph i.e. Log-Log plot can be referred with the values present in the above table.
- Theoretically, Log(O(n)) = Log(n) for a particular pair in dataset where O(n) = n. Thus, the equation is a straight line of form Y = mX where m is slope and it is 1 (theoretically)
- But, practically the value of slope is (say for n = 512 and 1024):
  (12-11)/(10-9) = 1 which is very close to theoretical slope i.e. 1
- Thus, the results with the graph clearly indicates that the Time complexity for the Quick Find algorithm is **O(n)** in average case and **O(n^2)** in worst case situation
- **NOTE**: We can see sudden spike in the graph for M = 8192 which is mainly because there are large number of connected pairs and hence cost of finding the root in find and union operations makes it larger.

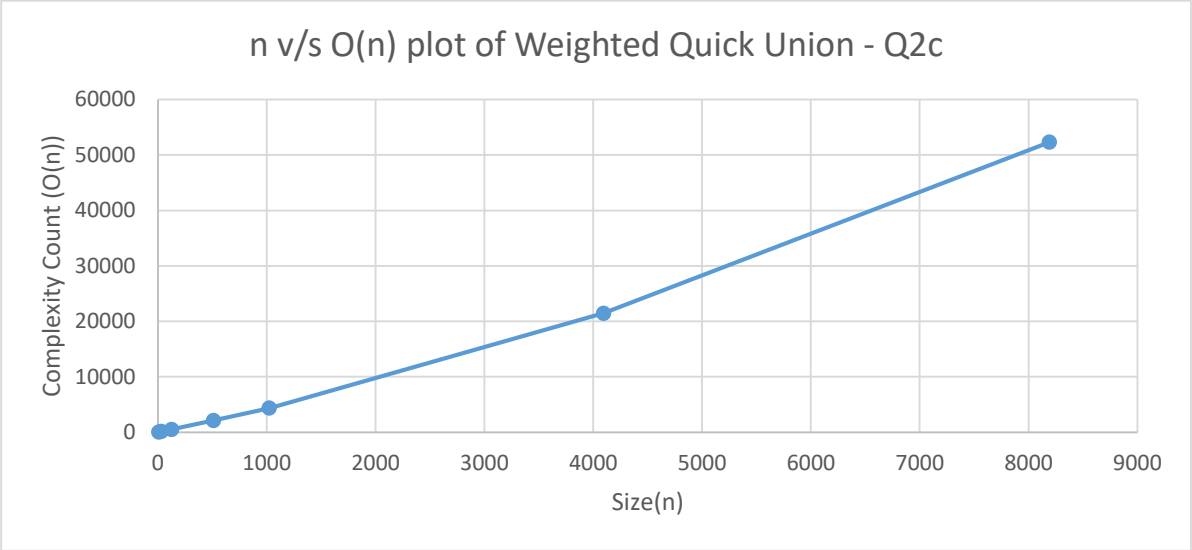
2c. Weighted Quick Union

Results: Time complexity for different size and also the Log-Log values of Time complexity and size

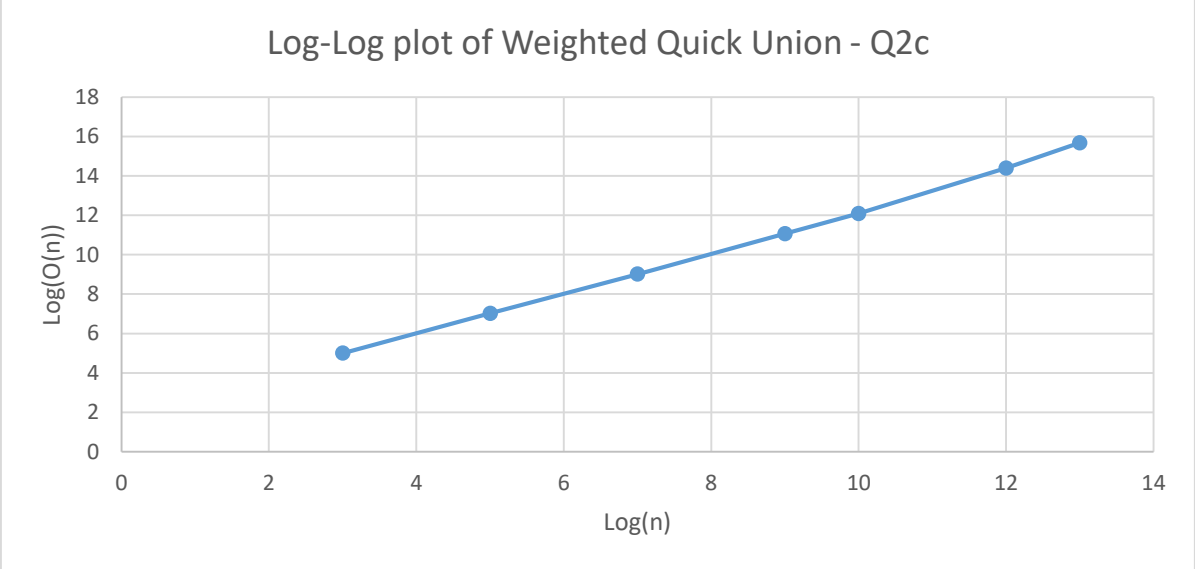
Size (n)	Time Complexity in terms of Complexity Counter(O(n))	Log(n)	Log(O(n))
8	32	3	5
32	130	5	7.022368
128	514	7	9.005625
512	2138	9	11.06205
1024	4354	10	12.08813
4096	21454	12	14.38896
8192	52315	13	15.67494

Graphs:

1. Size(n) v/s Complexity Counter ((O(N)) Plot



2. Log(n) v/s Log(O(n))) Plot



Explanation:

- This is Quick Weighted Union algorithm whose time complexity includes cost of Find and Union Operations which is indeed cost of finding the Root of the nodes.
- In weighted union, we assign the root of smaller tree to the larger tree, hence the cost of finding the root will be cost of depth of the tree i.e. **log(n)**.
- Since root for the pairs will be calculated in Find and Union operations 4 times, complexity will be 2log(n) + 2log(n) for find and union.
- But, since we find and perform operations for "M" given input pairs with the maximum value of a point label i.e. N = 8192 for all the different input data set, the time complexity comes to 2O(M*log(N)) + 2O(M*log(N)) which is approximately **4O(M*log(N))** for larger values of N i.e. cost of finding ROOT in Find and Union operations.
- In the worst-case situation, if M = N i.e. input data pairs are 8192 then **O(n) = O(nlog(n))** (ignoring 4 which is constant).
- Since the dataset is not the worst case, it is very difficult to analyze and verify the graph with the theoretical ones.
- But, from the graphs we can see that the algorithm is linearithmic with **O(n) = nlog(n)** in the worst case situation.

Conclusion:

- Quick Find and Quick Union are slow compared to Quick Weighted Union.
- The cost for union in Quick Find and Quick Union is O(n) while for Weighted Quick Union it is O(log(n)) because finding the root in Weighted Quick Union is calculating the depth of the node.

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.

Solution:

1a. Naïve Method:

Explanation:

- According to the logic implemented, cost of the algorithm is: N(N-1)(N-2)/6
  i.e. (N^3)/6 – (N^2)/2 + N/3
- Thus according to Tilde Notation, we will ignore lower order terms and hence our total cost of the implemented algorithm will be approximately: ~(N^3)/6 which is f(n)
- Now, for f(n) < c g(N), if c >= 1 then:
  (N^3)/6 < c * N^3 … where g(n) = N^3 (theoretical complexity).
- Thus, for all the values of N >=1, f(n) < cg(n) where c >=1
- But, since we require minimum 3 array elements for the algorithm to run once, the minimum value of N >= 3

1b. Sophisticated Method:

Explanation:

- According to the logic implemented, cost of the algorithm is: (N(N-1)*log(N))/2
  i.e. (N^2)Log(N)/2 – Nlog(N)/2

- Thus, according to Tilde Notation, we will ignore lower order terms and hence our total cost of the implemented algorithm will be approximately: ~(N^2)Log(N)/2 which is f(n)
- Now, for f(n) < c g(n), if c>=1 then:
  (N^2)Log(N)/2 < c * (N^2)Log(N) …where g(n) = (N^2)Log(N)(theoretical complexity).
- Thus for all the values of N >= 1, f(n) < c g(n) where c>=1
- But, since we require minimum 3 array elements for the algorithm to run once, the minimum value of N >= 3.

2a. Quick Find:

Explanation:

- According to the logic implemented, cost of the algorithm includes find and union functions: M + M*N
  ..Where M is number of pairs given in the dataset and N is the maximum point value in a given pair.
- Thus, according to Tilde Notation, we will ignore lower order terms and hence our total cost of the implemented algorithm will be approximately: ~(M*N) which is f(n)
- In the worst case situation, f(n) = N^2 because M=N (number of pairs are same as maximum value of a pair)
- Now, for f(n) < c g(n), if c>=1 then
  N^2 < c * N^2 … where g(n) = N^2 (Theoretical complexity)
- Thus, for all the values of N>=1, f(n) < c g(n) where c>=1

2b. Quick Union:

Explanation:

- According to the logic implemented, cost of the algorithm includes root function called by find and union functions: 2(M*N) + 2(M*N)
  ..Where M is number of pairs given in the dataset and N is the maximum point value in a given pair.
- Thus, according to Tilde Notation, total cost of the implemented algorithm will be approximately: ~(M*N) which is f(n) (ignoring the constant 4)
- In the worst case situation, f(n) = (N^2) because M=N (number of pairs are same as maximum value of a pair)
- Now, for f(n) < c g(n), if c>=1 then
  N^2 < c * N^2 … where g(n) = N^2 (Theoretical complexity)
- Thus, for all the values of N>=1, f(n) < c g(n) where c>=1

2c. Weighted Quick Union:

Explanation:

- According to the logic implemented, cost of the algorithm includes root function called by find and union functions: 2(M*log(N)) + 2(M*log(N))
  ..Where M is number of pairs given in the dataset and N is the maximum point value in a given pair.
- Thus, according to Tilde Notation, total cost of the implemented algorithm will be approximately: ~(M*Log(N)) which is f(n) (ignoring the constant 4)
- In the worst case situation, f(n) = (NLog(N)) because M=N (number of pairs are same as maximum value of a pair)
- Now, for f(n) < c g(n), if c>=1 then
  NLog(N) < c * NLog(N) … where g(n) = NLog(N) (Theoretical complexity)

- Thus, for all the values of N>=2, f(n) < c g(n) where c>=1.
- N>=2 is required because base of Log is 2 and hence to get non-zero positive value, N>=2 so that N^2Log(N) will be non-zero.

Q4. Farthest Pair (1 Dimension): Write a program that, given an array a[] of N double values, find a farthest pair: two values whose difference is no smaller than the difference of any other pair (in absolute value). The running time of the program should be LINEAR IN THE WORST CASE.
Solution:
Explanation:

- The complexity for the implemented algorithm is linear i.e. of the order N where N is number of array elements.
- The logic implemented is that we assign the lowest and highest values to the first element of array and then traverse through the array so that we get true values of lowest and highest after comparison.
- Once, the lowest and highest values are determined we can then subtract to find the farthest absolute distance.
- Since, we traverse the array only once for finding the lowest and highest values, the cost is N i.e. size of array. Thus, time complexity is **O(n)** i.e. linear.

Q5. Faster-est-ist 3-sum: Develop an implementation that uses a linear algorithm to count the number of pairs that sum to zero after the array is sorted (instead of the binary-search based linearithmic algorithm). Use the ideas to develop a quadratic algorithm for the 3-sum problem.
Solution:
5a. Linear Algorithm:
Explanation:

- The complexity for the implemented algorithm is linear i.e. of the order N where N is number of array elements.
- NOTE: I have implemented Bubble Sort to sort the array before finding the sum of pairs and hence we will ignore the complexity since question mentions finding only the cost of algorithm to find the sum of pairs.
- The logic implemented is that we keep track of the left index and right index each initialized to the first index of array and last index of array and hence they will point to the minimum and maximum values of array
- We can add the elements of current indexes and if the sum is greater than 0 than we move to the next lower value else if the sum is lower than 0 then we move to the next higher value.
- If the sum is 0, we move to the next lower and higher value and increment the counter.
- Since, we traverse the array only once for finding the pairs, the cost is N i.e. size of array in worst case situation. Thus, time complexity is **O(n)** i.e. linear.

5b. Quadratic Algorithm:

Explanation:

- The complexity for the implemented algorithm is quadratic i.e. of the order N^2 where N is number of array elements.
- NOTE: I have implemented Bubble Sort to sort the array before finding the sum of pairs and hence we will ignore the complexity since question mentions finding only the cost of algorithm to find the three sum of pairs.
- The logic implemented is for each value of array we find the complement of it in the rest of the array such that sum of current array element and the pair will be 0. For finding the pair of number, we will use the Linear algorithm mentioned in above section 5a.
- Since, we traverse the array twice – one for each array element and other for finding the pairs, the cost is N^2 in worst case situation. Thus, time complexity is **O(n^2)** i.e. quadratic.