DSA Spring 2018 HW1

Zhongze Tang zt67

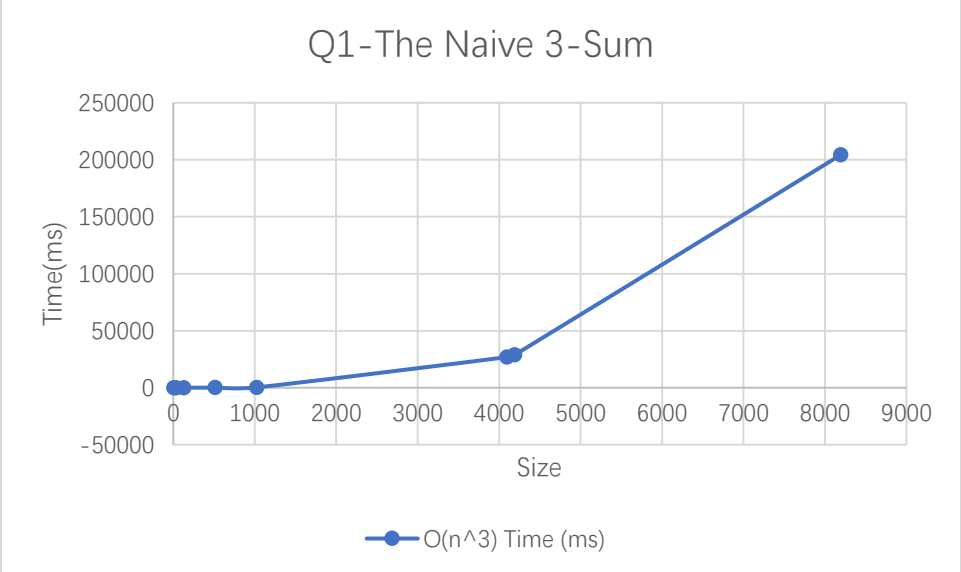
Note:
- For every Time(ms), I run the program 5 times and get the average value of running time (in nanosec) and convert the time to ms. You can find the raw data in sourcefile/Q*/RunTimeData/. * can be 1,2,4 or 5.

Q1

I run the program manually to get the run time data.

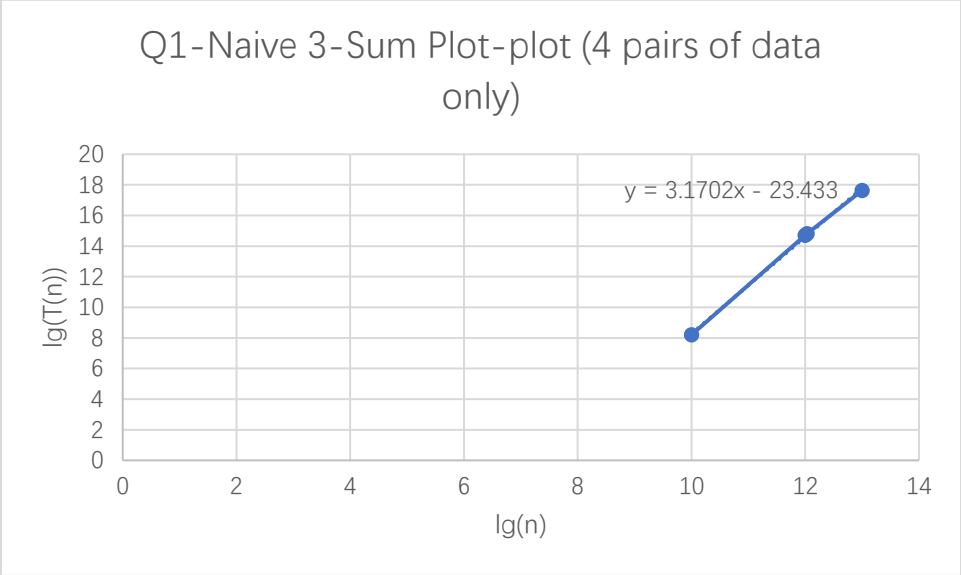
Size	O(n^3) Time (ms)	O(n^2lgn) Time (ms)
8	0.0360462	0.376964
32	1.14871	0.8583714
128	17.054031	3.6209404
512	69.7116648	17.5848682
1024	294.1584942	35.7357164
4096	26852.58152	297.7907658
4192	28854.05078	304.215905
8192	204232.0919	1131.787488

a) The Naïve 3-Sum



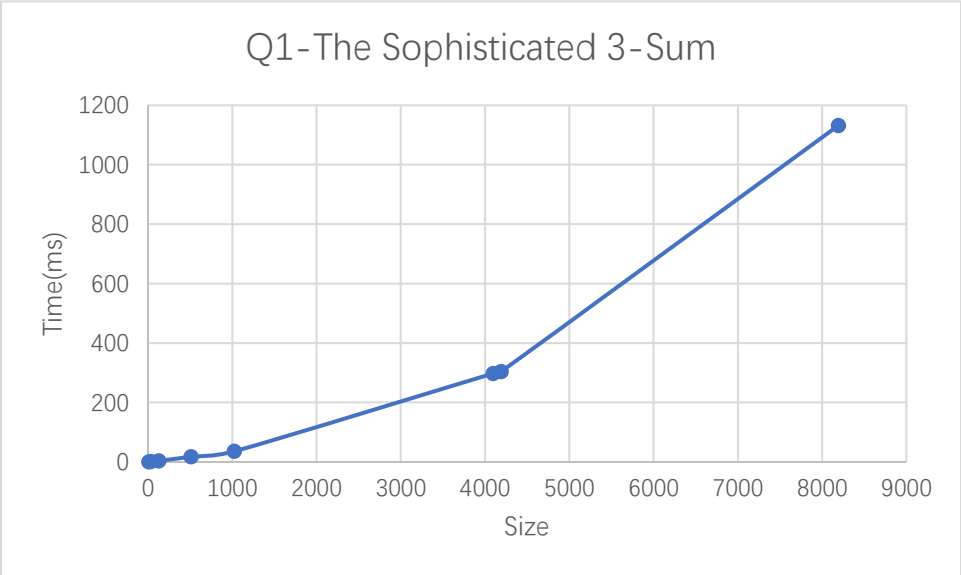
In the first algorithm (brute force, the naive one), it's a $O(n^3)$ algorithm according to the implementation. I try to use Log-log plot but

find that the slope is only 2.14, far away from 3. I suppose that it is because when the data is small, the data size will not be the main thing that affects the running time. But if I use the last four pairs of data only (1024, 4096, 4192, 8192), it performs well, and the slope is 3.14, which indicates it's a O(n^3) algorithm.



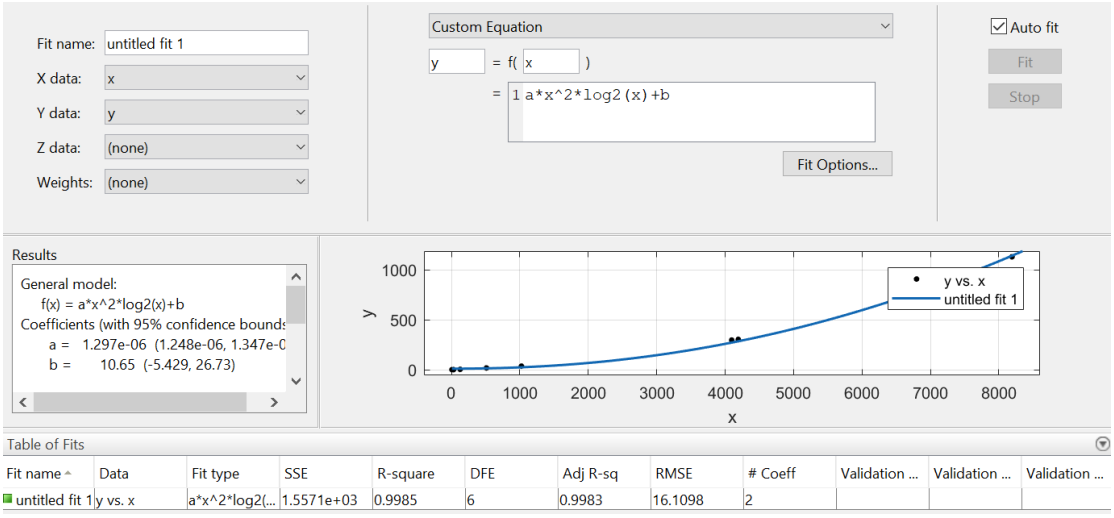
I try doubling hypothesis as well. For data size 4096 and 8192, their running time are 26852ms and 204232ms, respectively. 204232/26852 = 7.605, and $\log_2 7.605 = 2.93 \approx 3$, so the algorithm is a $O(n^3)$ one.

b) The Sophisticated 3-Sum



In the second one (“sophisticated”), use the same method, we can get that 1131/297 = 1.93 ≈ 2, so I consider it as a $O(n^2 lgn)$ one.

Moreover, I use the Curve Fitting Tool in MATLAB to fit the data of it. The result is $y = 1.297*10^{-6}*x^2*lgx+10.65$. It is showed as below.



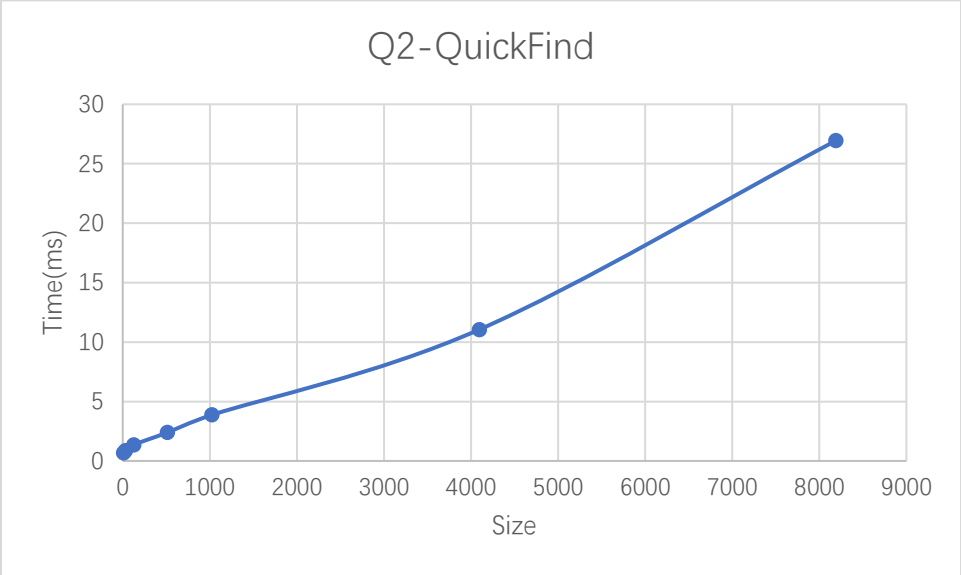
Q2

I write a 5-times loop to calculate the average running time of each algorithm.

The QuickUnion algorithm seems faster than the QuickFind, becase it doesn’t need to go through the entire array to change the root when connecting pairs. And the WeightedQuickUnion is fastest because it shortens the time to find root.

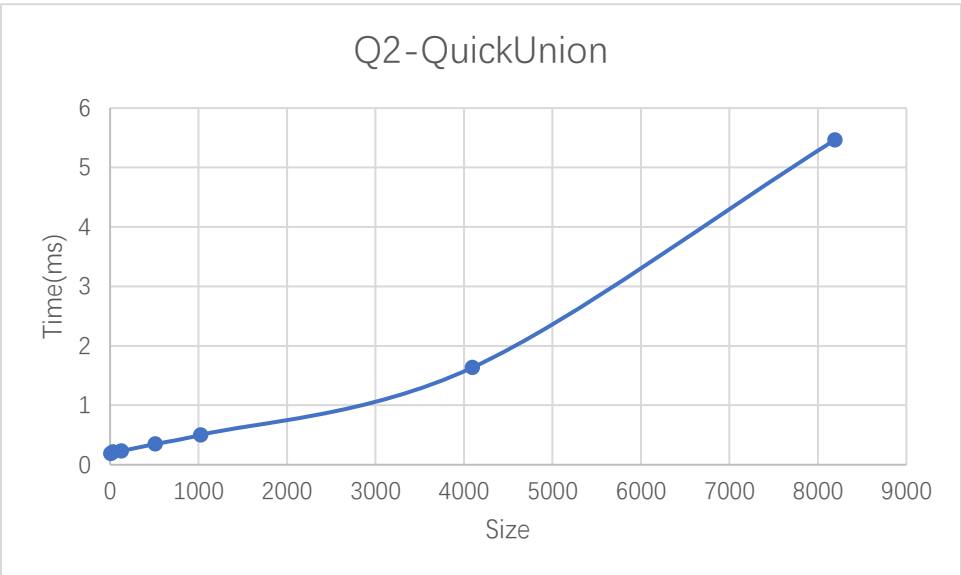
Size	QuickFind(ms)	QuickUnion(ms)	WQuickUnion(ms)
8	0.674939	0.190575	0.260204
32	0.869127	0.214305	0.261682
128	1.361126	0.231876	0.274738
512	2.390531	0.34995	0.388541
1024	3.871127	0.501113	0.579692
4096	11.054957	1.6357	1.470332
8192	26.948044	5.465034	2.085249

a) QuickFind



It’s an algorithm, whose order is O(M*N), where M stands for the input size and N is 8192 always. To be specific, the order of union() is O(N), where N is the maximum value of point labels. And the union() will run M times at most, where M is the number of input pairs, so the order of it is O(M*N). In the worst condition, it’s O(N^2).

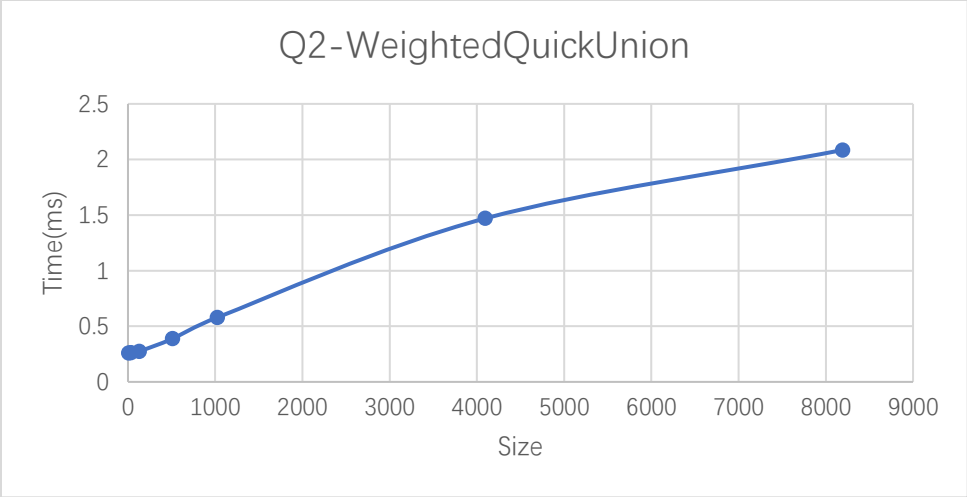
b) QuickUnion



I think it’s an algorithm whose order is O(M*N) as well, where M stands

for the input size and N is 8192 always. The order of union() is the height of the tree, but consider the cost of finding the root, it is O(N). Again it runs M times, so the order of it is O(M*N). In the worst condition, it's O(N^2).

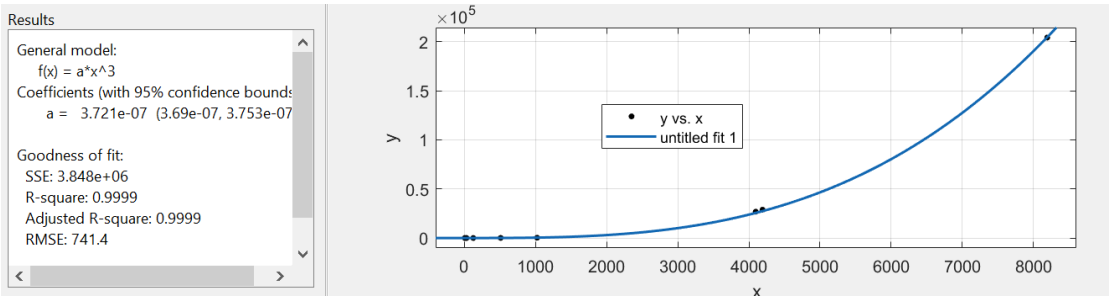
c) WeightedQuickUnion



The order of union() is O(lgN), and run at most M times is O(M*lgN). I cannot fit the curve well if I suppose g(N) = a*NlgN+b, I think it is because the test data are not the worst case. The worst case is linearithmic, which is O(NlgN).

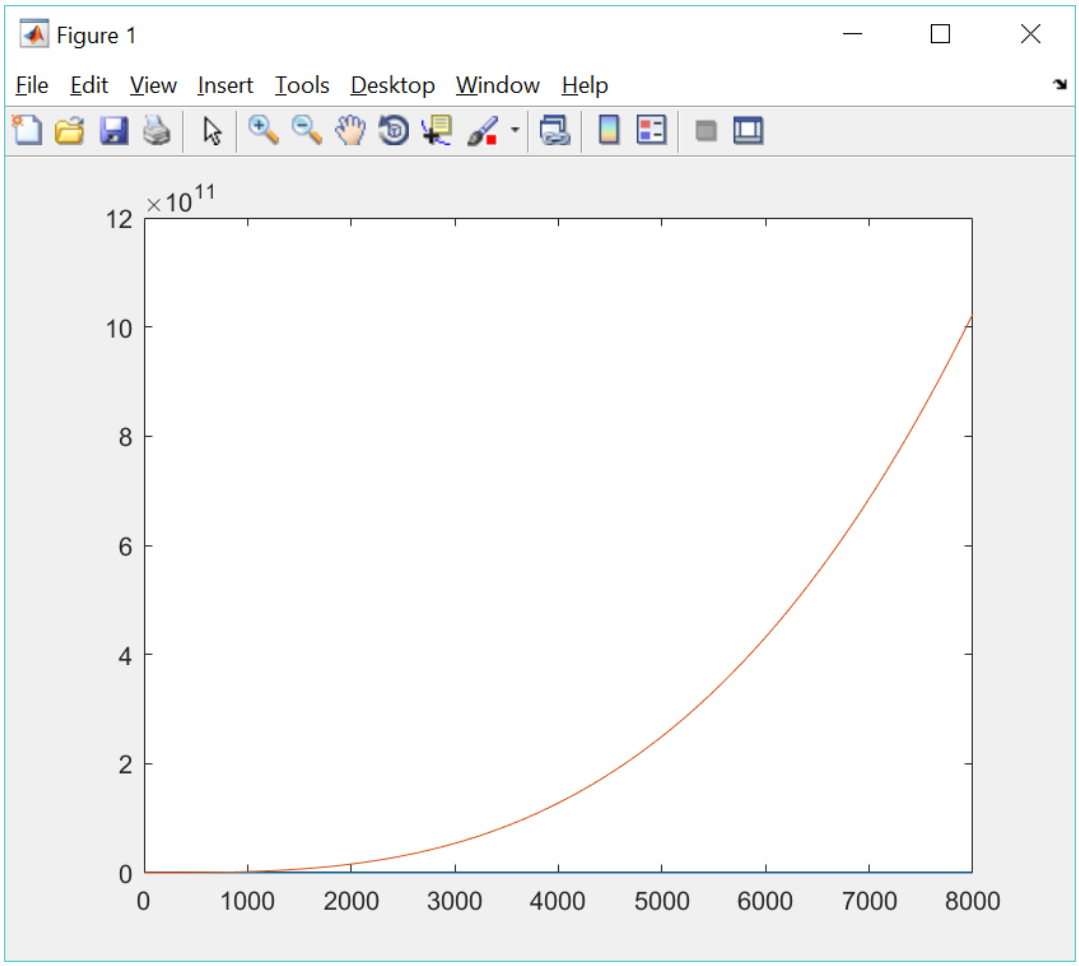
Q3

Q1_a) "Naïve" 3-Sum (O(N^3))

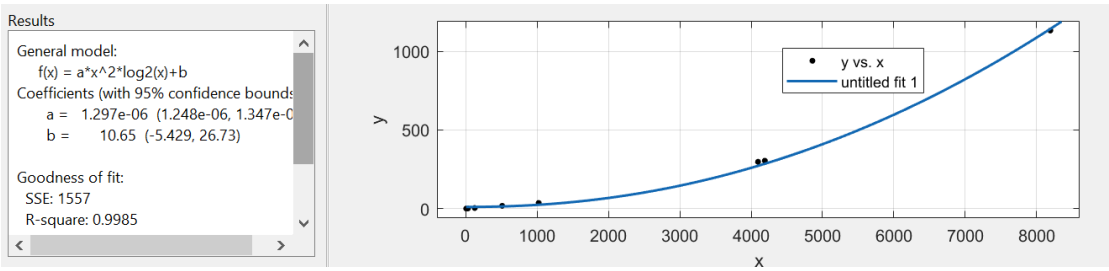


Suppose f(x) = 3.721e-07*x^3, that is, $F(N) = 3.721*10^{-7}*N^3$

Assume that g(N) = N^3 and c = 1. That is, find Nc that F(N)<c*g(N), for N>Nc. We can set Nc = 1.

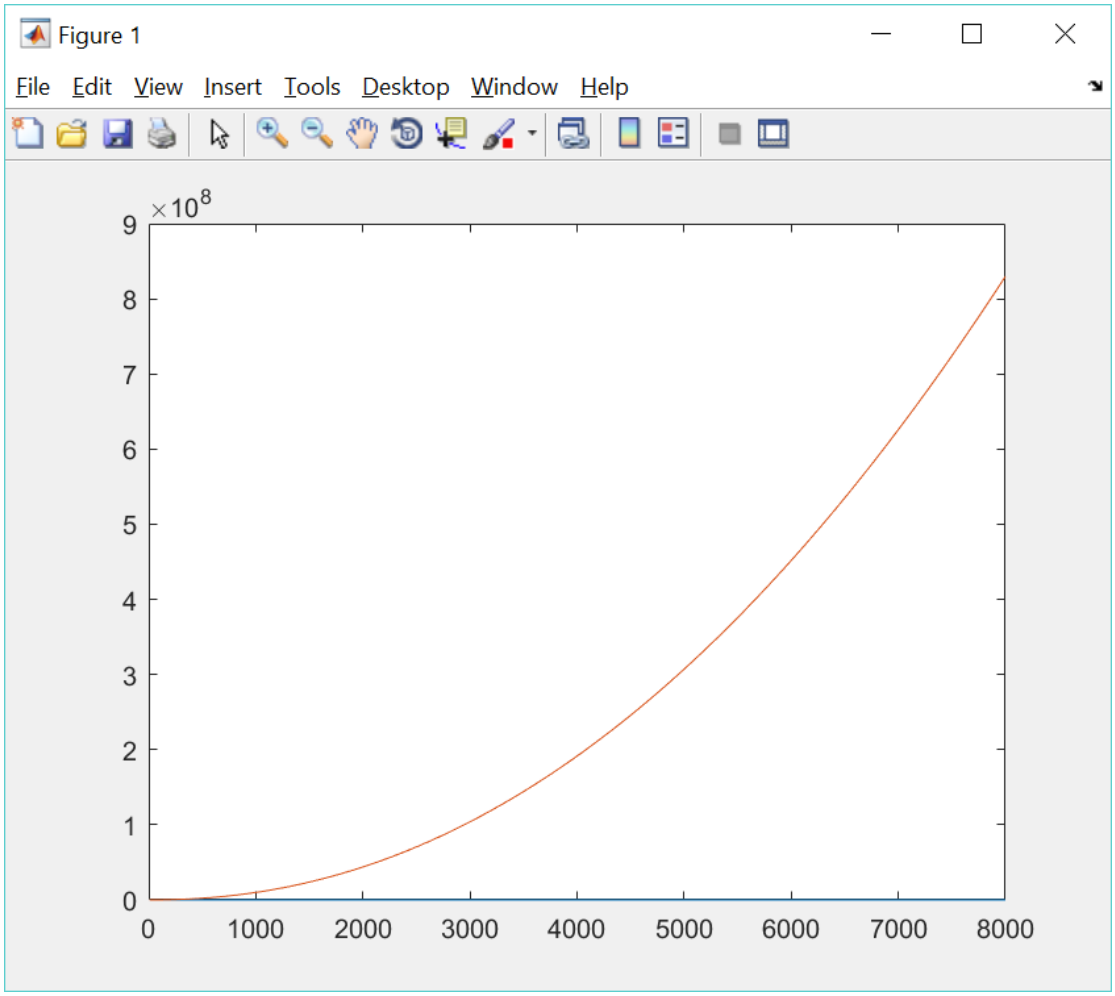


Q1_b) O(n^2lgn) 3-Sum

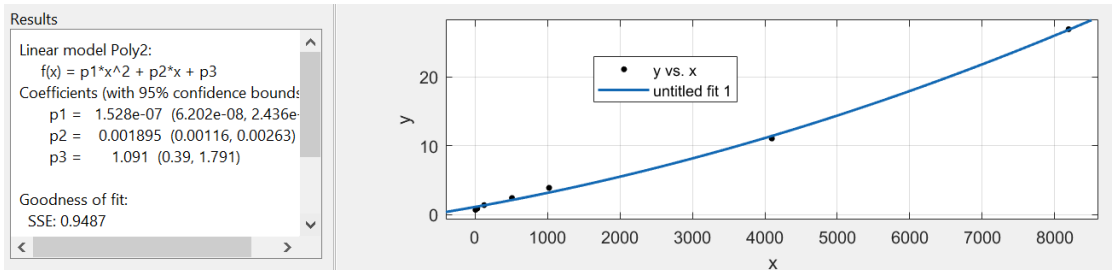


F(N) = 1.297*10^(-6) * N^2 *lg(N) + 10.65

Assume g(N) = N^2*lg(N), c = 2. As to get F(N)<c*g(N), where N>Nc, set Nc = 10.

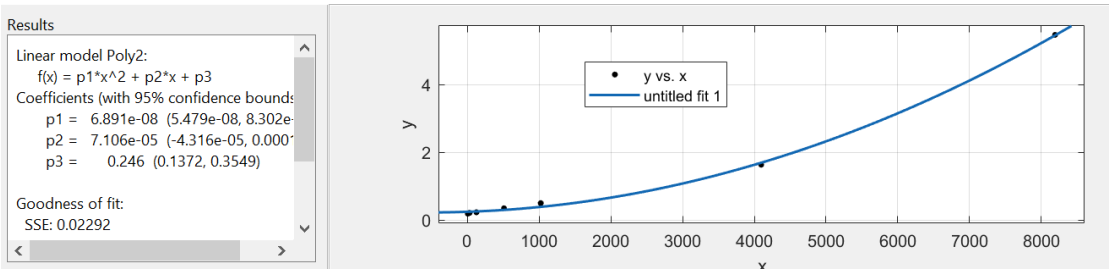


Q2_a) QuickFind



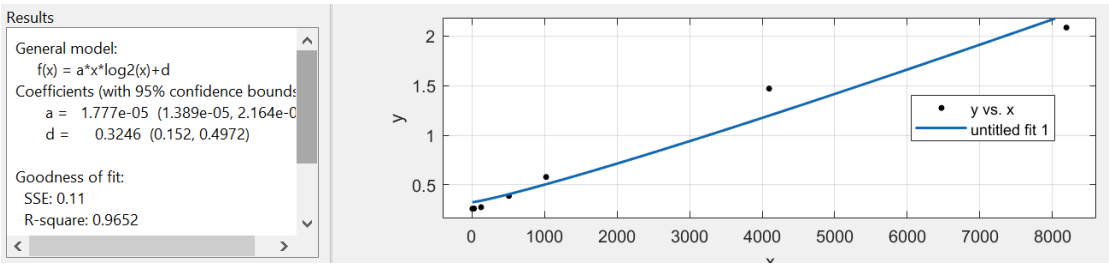
F(N) = 1.528*10^(-7)*N^2 + 0.001895*N + 1.091.
Suppose g(N) = N^2, c = 2. Get Nc = 2.

Q2_b) QuickUnion



F(N) = 6.891*10^(-8)*N^2+7.106*10^(-5)*N+0.246
Suppose g(N) = N^2, c = 2. Set Nc = 2.

Q2_c) WeightedQuickUnion

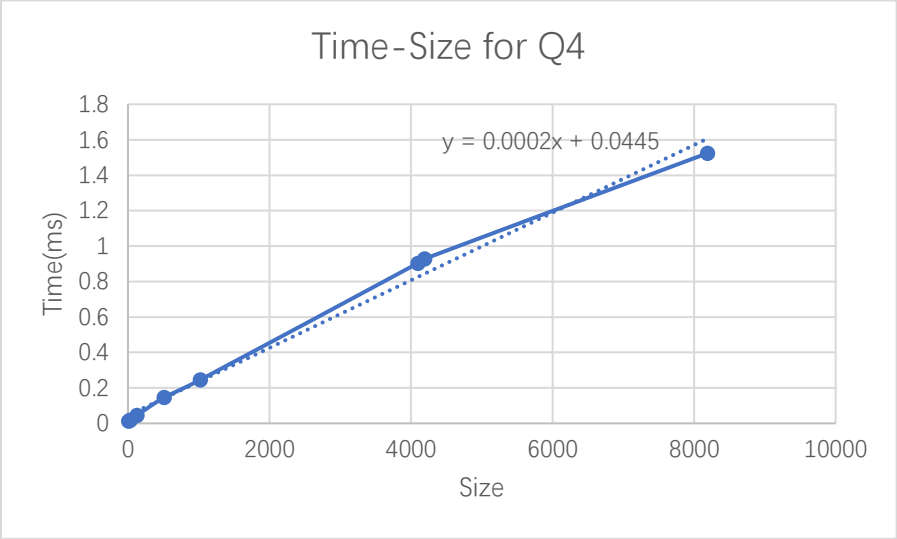


F(N) = 1.777*10*(-5)*N*lgN + 0.3246
Suppose g(N) = N*lgN, c = 2. Set Nc = 2.

Q4

The algorithm just goes through all the data one time to find the maximum one and the minimum one. I use the data from Q1. It is obvious that it's a linear algorithm.

Size	Time(ms)
8	0.012234
32	0.0175712
128	0.0456526
512	0.1463188
1024	0.246328
4096	0.9034492
4192	0.9284104
8192	1.5244424



Q5

I take the duplicate triplets into consideration. That is, {-1, 0, 0, 1} will return 2, not 1. I use the data from Q1 to test the algorithm. Using 4092 and 8192, 89.86/24.63 = 3.65, $\log_2 3.65 = 1.87 \approx 2$, so it could be a O(n^2) algorithm.

Size	Time(ms)
8	0.012152
32	0.103129
128	1.412363
512	5.314117
1024	8.34814
4096	24.63125
8192	89.86426