Tea Elming
1DV018
Sept 2024

Question 4: Comparing UnionFind Algorithms

Introduction

Three different implementations of the UnionFind algorithm will be discussed in this section: QuickFind, QuickUnion, and WeightedUnion. Although the instructions were to compare two different implementations, I found the difference between QuickUnion and WeightedUnion to be interesting, and therefore wanted to include both of those in the comparison with QuickFind.

The comparisons are structured into four different components, where the algorithms are tested with both varying array sizes, as well as varying amounts of either finds or unions, depending on which functions are being tested. The comparisons are represented both graphically, with linear graphs as well as scatter graphs, and in tables.

Comparison 1: Execution Time vs. Array size ~ Finds

Expectation

Since the find operation itself only assesses an element in the array based on the index, it is expected that the execution time of this should be constant (*O(1)*). However, as we compare the QuickFind with the QuickUnion and the weightedUnion, we expect the QuickFind to be slightly faster than the other two.

Outcome

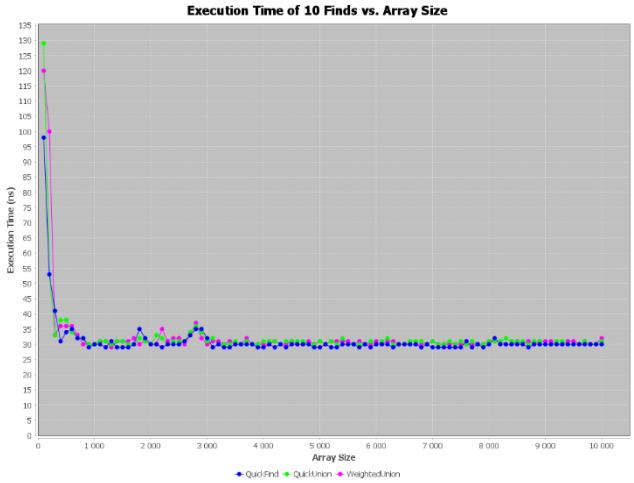

Figure 2 - Execution Time vs. Array Size for Finds (linear)

Figure 1 - Execution Time vs. Array Size for Finds (scatter)

Although there are some outliers, the graphs above show that the execution time is fairly constant. The averages are calculated from running the algorithms 10 times for each point, which is far from enough to ensure accuracy. This is something which will be a factor for all future graphs and tables as well, and will therefore not be mentioned again, but with better hardware these algorithms should be run significantly more times to achieve higher accuracy.

We can also see, with some difficulties, that the QuickFind (*Dark Blue*) is below the two other algorithms for the majority of the points. This aligns with our initial conclusions, as we expected it to be faster.

Below, the tables representing the varying times for different array sizes may be seen. The error margins are also based on the minimum and maximum values when running the algorithms 10 times and are therefore not very accurate. However, a common occurrence amongst the different algorithms is the high error margins for the smaller array sizes. This could be due to the fact that sometimes, the algorithm is more likely to “get lucky” and find what it is looking for very quickly, whereas there is a smaller chance for that as the array grows. It is also possible to see that the QuickFind is not actually faster for the smaller arrays, which is most likely due to the fact that method is less important the smaller the array is. However, it is also clear from the tables that the high values in the beginning are outliers, and that the QuickFind is slightly more efficient or at the same level as the others for the larger array sizes.

Tea Elming
1DV018
Sept 2024

Table Representation of Find-operations with Varying Array Sizes

QuickFind

Array Size	Avg Time (ns)	Error Margin (%)
100	378620.0	67.41
600	98200.0	7.24
1100	86190.0	8.77
1600	71430.0	2.98
2100	58610.0	12.37
2600	60700.0	9.06
3100	29130.0	3.71
3600	35420.0	5.79
4100	41520.0	3.81
4600	36540.0	4.54
5100	70290.0	14.88
5600	52960.0	10.1
6100	50120.0	11.09
6600	30740.0	4.52
7100	29690.0	3.77
7600	30290.0	3.53
8100	31950.0	3.04
8600	35900.0	12.48
9100	29610.0	3.04
9600	32250.0	3.16

QuickUnion

Array Size	Avg Time (ns)	Error Margin (%)
100	242670.0	66.26
600	88440.0	5.82
1100	124690.0	5.82
1600	109030.0	22.45
2100	58410.0	1.63
2600	54620.0	10.44
3100	36680.0	4.96
3600	30950.0	2.39
4100	45990.0	4.11
4600	38570.0	2.36
5100	40990.0	5.66
5600	56460.0	10.06
6100	57700.0	10.47
6600	58830.0	8.79
7100	34760.0	2.68
7600	34060.0	4.35
8100	32490.0	1.23
8600	31620.0	2.69
9100	32720.0	3.58
9600	31000.0	1.84

WeightedUnion

Array Size	Avg Time (ns)	Error Margin (%)
100	267120.0	65.09
600	79440.0	2.83
1100	114330.0	1.54
1600	136210.0	6.26
2100	127810.0	3.15
2600	83590.0	12.41
3100	41680.0	4.44
3600	41470.0	9.4
4100	51940.0	4.76
4600	62200.0	3.78
5100	52310.0	4.87
5600	66970.0	8.06
6100	103260.0	15.82
6600	73490.0	6.06
7100	75800.0	8.06
7600	77710.0	5.37
8100	50530.0	3.17
8600	41350.0	1.74
9100	47490.0	1.71
9600	54310.0	8.25

Comparative

Array Size	Avg QF (ns)	Avg QU (ns)	Avg WU (ns)
100	378620.0	242670.0	267120.0
600	98200.0	88440.0	79440.0
1100	86190.0	124690.0	114330.0
1600	71430.0	109030.0	136210.0
2100	58610.0	58410.0	127810.0
2600	60700.0	54620.0	83590.0
3100	29130.0	36680.0	41680.0
3600	35420.0	30950.0	41470.0
4100	41520.0	45990.0	51940.0
4600	36540.0	38570.0	62200.0
5100	70290.0	40990.0	52310.0
5600	52960.0	56460.0	66970.0
6100	50120.0	57700.0	103260.0
6600	30740.0	58830.0	73490.0
7100	29690.0	34760.0	75800.0
7600	30290.0	34060.0	77710.0
8100	31950.0	32490.0	50530.0
8600	35900.0	31620.0	41350.0
9100	29610.0	32720.0	47490.0
9600	32250.0	31000.0	54310.0

Figure 3 - Tables of Execution Times for Find-operations

Tea Elming
1DV018
Sept 2024

Comparison 2: Execution Time vs. Number of Finds ~ Finds

Expectation

Since we have already concluded that it takes a constant amount of time to conduct a find, we expect the graph to show a linear increase (*O(N)*) of execution time as the number of finds increases. We also expect the QuickFind method to be faster, and therefore have a less steep gradient, than the QuickUnion and WeightedUnion algorithms.

Outcome

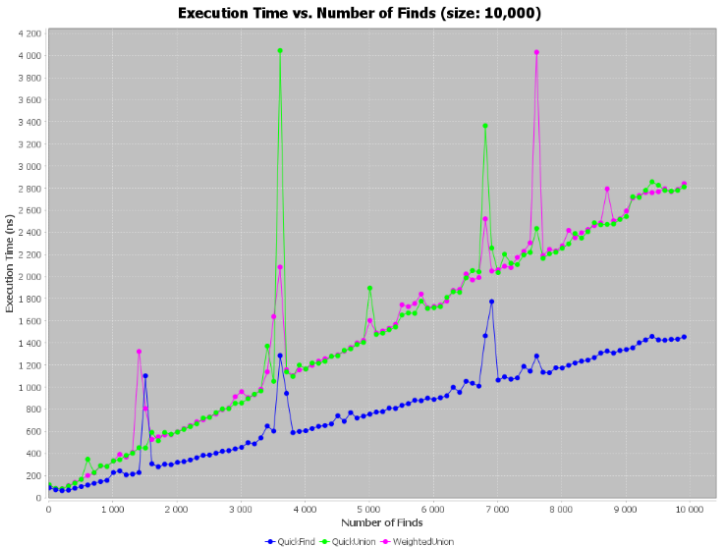

Figure 5 - Execution Time vs. Number of Finds (linear)

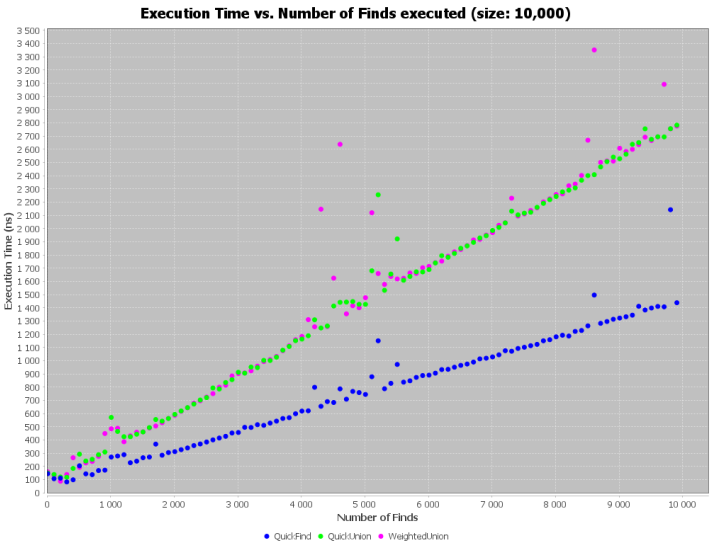

Figure 4 - Execution Time vs. Number of Finds (scatter)

The benefits of the QuickFind-method become more apparent as we increase the number of find-operations performed, rather than the array size. Although we still have the odd outliers, we can clearly see that the QuickFind (*dark blue*) has a far less steep gradient, and it appears to be almost twice as fast for the higher numbers of finds (from *approx. 8000+*). It is interesting to note that the outliers seem to fall around the same values on the x-axis, which leads us to ponder upon why that is. As mentioned before, this is not accurately executed as we would need to run it thousands, if not millions, of times to achieve a somewhat accurate output, especially when working which such small time periods. However, our graphs are enough to tell us that we have a linear increase (*O(N)*), with the QuickFind-method being the most efficient.

Tea Elming
1DV018
Sept 2024

Table Representation of Find-operations with Varying Numbers of Finds

QuickFind

Number of Finds	Avg Time (ns)	Error Margin (%)
10	137329.0	24.57
510	55245.0	0.83
1010	39789.0	1.1
1510	75712.0	0.43
2010	66011.0	0.15
2510	92724.0	1.11
3010	179361.0	0.86
3510	202113.0	0.8
4010	140646.0	0.59
4510	147849.0	0.78
5010	133012.0	0.55
5510	141147.0	0.44
6010	153330.0	0.55
6510	154463.0	0.78
7010	169686.0	0.43
7510	237002.0	0.88
8010	198448.0	6.39
8510	182586.0	0.34
9010	182241.0	0.17
9510	231527.0	1.23

QuickUnion

Number of Finds	Avg Time (ns)	Error Margin (%)
10	104095.0	21.38
510	62569.0	24.73
1010	72926.0	1.1
1510	109281.0	0.39
2010	123365.0	0.98
2510	142463.0	0.93
3010	247355.0	0.75
3510	256764.0	0.79
4010	229924.0	0.7
4510	206287.0	0.56
5010	273213.0	4.77
5510	223068.0	0.45
6010	242582.0	0.4
6510	245323.0	0.4
7010	307443.0	0.32
7510	310008.0	0.8
8010	303183.0	0.55
8510	324803.0	0.41
9010	352869.0	0.5
9510	360595.0	0.32

WeightedUnion

Number of Finds	Avg Time (ns)	Error Margin (%)
10	129630.0	17.01
510	51531.0	2.13
1010	97113.0	0.8
1510	123355.0	11.44
2010	118297.0	0.55
2510	145940.0	1.28
3010	203345.0	0.71
3510	284362.0	0.89
4010	286848.0	0.65
4510	235507.0	1.08
5010	236068.0	0.46
5510	225121.0	0.83
6010	245740.0	0.55
6510	276061.0	0.42
7010	319092.0	0.86
7510	305091.0	0.83
8010	300269.0	0.42
8510	325474.0	0.52
9010	376290.0	0.43
9510	342346.0	0.29

Comparative

Number of Finds	Avg QF (ns)	Avg QU (ns)	Avg WU (ns)
10	137329.0	104095.0	129630.0
510	55245.0	62569.0	51531.0
1010	39789.0	72926.0	97113.0
1510	75712.0	109281.0	123355.0
2010	66011.0	123365.0	118297.0
2510	92724.0	142463.0	145940.0
3010	179361.0	247355.0	203345.0
3510	202113.0	256764.0	284362.0
4010	140646.0	229924.0	286848.0
4510	147849.0	206287.0	235507.0
5010	133012.0	273213.0	236068.0
5510	141147.0	223068.0	225121.0
6010	153330.0	242582.0	245740.0
6510	154463.0	245323.0	276061.0
7010	169686.0	307443.0	319092.0
7510	237002.0	310008.0	305091.0
8010	198448.0	303183.0	300269.0
8510	182586.0	324803.0	325474.0
9010	182241.0	352869.0	376290.0
9510	231527.0	360595.0	342346.0

Figure 6 - Tables of Execution Times for Varying Numbers of Find-operations

Tea Elming
1DV018
Sept 2024

Comparison 3: Execution Time vs. Array size ~ Unions

Expectations

The predictions for the union-operations are not quite as straight forward as they were for the finds-operations. The implementation of the unions is what really sets the different algorithms apart, and it becomes more apparent as we begin to look at the different array sizes. The QuickFind is expected to be the slowest as it iterates through the entire array for each updated union, and we expect it to grow linearly as the number of elements in the array increase. The QuickUnion is expected to be significantly better than the QuickFind, but it is mainly reliant on the find operation, which grows linearly. Since the Union-operation itself only updates one point once it has been found, the execution time for that is constant (*O(N)*). However, since we have a constant number of unions, and therefore finds, when looking at the increase in array size, we can expect the graph to take a constant amount of time. The weighted union is an improvement on the quick union, but slightly more complicated to predict. Since we have implemented a tree-like structure (without using TreeMap in Java, but rather through are implementations of ranks), we can assume a worst-case scenario of a logarithmic growth (*O(logN)*). However, we have some significant limitations in our testing, such as the fact that we only conduct 99 unions, and the array sizes grow from 100 to 10,000. Keeping in mind that the binary log for 100 is approximately 6.644, and the binary log for 10,000 is approximately 13.288, it will almost appear constant on our graphs. Had we increased our array size to 1,000,000, the $\log_2(N)$ would have been just below 20, and had we increased it to a billion ($N=10^9$), $\log_2(N)$ would have been just below 30. Hence, due to the limitations of our testing, the graphs for the WeightedUnion will appear constant.

Outcome

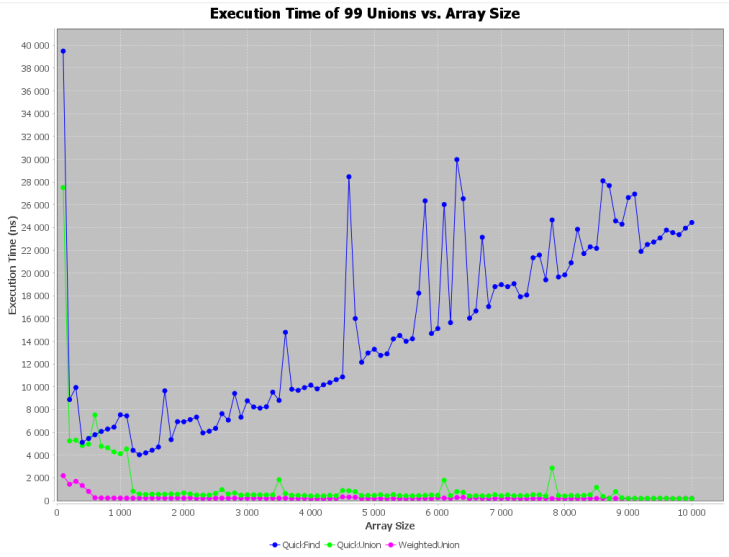

Figure 7 - Execution Time vs. Array Size of 99 Unions (linear)

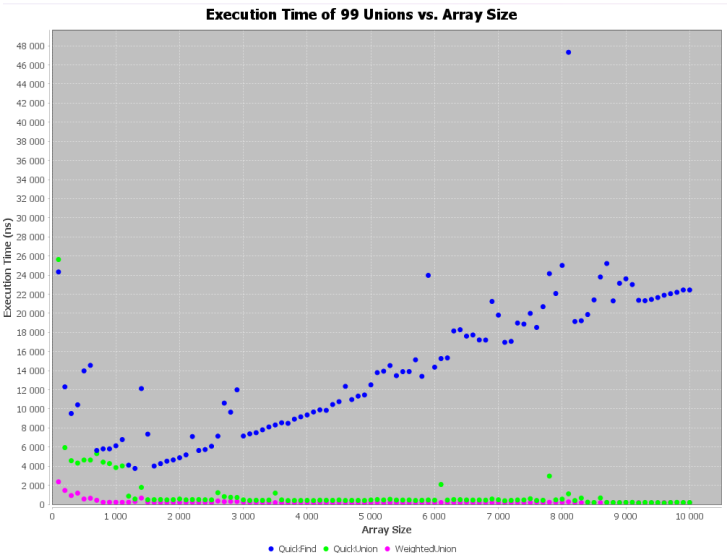

Figure 8 - Execution TIme vs. Array Size of 99 Unions (scatter)

Tea Elming
1DV018
Sept 2024

As can be seen from the graphs above, the QuickFind method appears to have a linear growth, whereas the QuickUnion and WeightedUnion appear constant, with the exceptions of their outliers. It is interesting to not that the QuickFind seems to have significantly more outliers than the other two, something which most likely depends on the fact that it is far more resource-intensive and therefore has a bigger risk of getting temporarily stuck at times. It is also interesting to note that both the QuickUnion and the WeightedUnion appear to have the majority of their outliers when the array size is smaller, which is interesting as it may suggest that these methods improve their efficiency, or at least their consistency in timing, with larger array sizes.

Tea Elming
1DV018
Sept 2024

Table Representation of Union-operations with Varying Array Sizes

QuickFind

Array Size	Avg Time (ns)	Error Margin (%)
100	286050.0	62.34
600	54790.0	9.24
1100	99010.0	6.83
1600	90370.0	0.84
2100	86650.0	1.05
2600	135240.0	8.7
3100	184020.0	8.36
3600	172320.0	6.69
4100	138390.0	2.42
4600	136950.0	2.4
5100	228200.0	8.86
5600	182820.0	2.07
6100	167660.0	0.78
6600	257560.0	5.94
7100	200040.0	2.58
7600	188130.0	0.22
8100	211210.0	2.81
8600	228720.0	2.96
9100	271410.0	5.18
9600	371500.0	8.55

QuickUnion

Array Size	Avg Time (ns)	Error Margin (%)
100	204520.0	51.11
600	50030.0	11.31
1100	69820.0	11.19
1600	42400.0	3.3
2100	38790.0	1.11
2600	93480.0	6.78
3100	41970.0	5.53
3600	44590.0	16.64
4100	55630.0	23.75
4600	28840.0	7.73
5100	49270.0	5.89
5600	35900.0	12.65
6100	19570.0	2.25
6600	18330.0	2.95
7100	18180.0	3.74
7600	18930.0	1.48
8100	22370.0	5.32
8600	28410.0	5.14
9100	24700.0	5.26
9600	37240.0	2.74

WeightedUnion

Array Size	Avg Time (ns)	Error Margin (%)
100	198040.0	61.62
600	68100.0	6.68
1100	56120.0	13.13
1600	46680.0	2.19
2100	70860.0	9.92
2600	116380.0	18.8
3100	27950.0	4.4
3600	44340.0	6.63
4100	34430.0	7.73
4600	30820.0	4.8
5100	46140.0	8.39
5600	56780.0	12.38
6100	47540.0	6.27
6600	30400.0	1.02
7100	28910.0	3.25
7600	28830.0	1.39
8100	32580.0	2.7
8600	44290.0	5.15
9100	41180.0	3.64
9600	59180.0	4.44

Comparative

Array Size	Avg QF (ns)	Avg QU (ns)	Avg WU (ns)
100	286050.0	204520.0	198040.0
600	54790.0	50030.0	68100.0
1100	99010.0	69820.0	56120.0
1600	90370.0	42400.0	46680.0
2100	86650.0	38790.0	70860.0
2600	135240.0	93480.0	116380.0
3100	184020.0	41970.0	27950.0
3600	172320.0	44590.0	44340.0
4100	138390.0	55630.0	34430.0
4600	136950.0	28840.0	30820.0
5100	228200.0	49270.0	46140.0
5600	182820.0	35900.0	56780.0
6100	167660.0	19570.0	47540.0
6600	257560.0	18330.0	30400.0
7100	200040.0	18180.0	28910.0
7600	188130.0	18930.0	28830.0
8100	211210.0	22370.0	32580.0
8600	228720.0	28410.0	44290.0
9100	271410.0	24700.0	41180.0
9600	371500.0	37240.0	59180.0

Figure 9 - Tables of Execution Times for Union-operations

Tea Elming
1DV018
Sept 2024

Comparison 4: Execution Time vs. Number of Unions ~ Unions

Expectations

The expectations for how the graphs will behave as the number of unions increase differs from that of the array increase. The predictions of calculations are similar for the QuickFind and WeightedUnion, however the QuickUnion will most likely differ. The QuickUnion is dependent on the find-operation, which means that if the tree-structure for the parents becomes too unbalanced, the find-operation will potentially begin to take a lot longer. The array-size limitation is 10,000, which is why the expectation might be that the QuickUnion will become to struggle, and therefore take longer, once it has connected more than half of the elements in the array.

Outcome

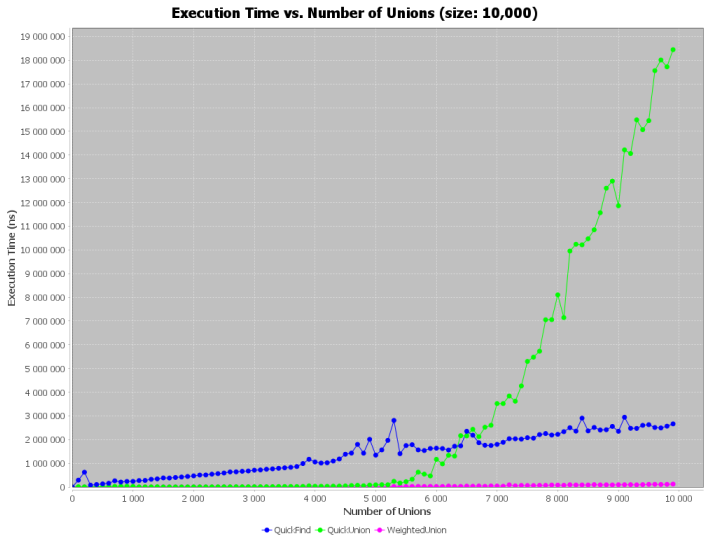

Figure 10 - Execution Time vs. Number of Unions (linear)

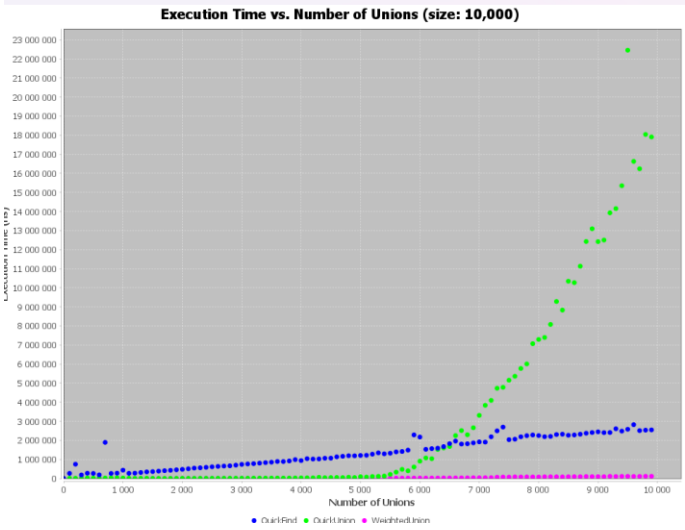

Figure 11 - Execution Time vs. Number of Unions (scatter)

As predicted, the QuickUnion began to take significantly longer time to execute once it reached approximately 6000. It is difficult to estimate if the growth from that point is linear or exponential, but it is a very steep gradient and not desirable. It is fascinating to compare it against the WeightedUnion, which appears to be very steady and executes without any major difficulties. It was slightly unexpected to see how much better the QuickFind performs than the QuickUnion from about 6500 and onwards, but that surprise stems mainly from the fact that it was unexpected that the QuickUnion would increase so rapidly.

Tea Elming
1DV018
Sept 2024

Table Representation of Union-operations with Varying Numbers of Unions

QuickFind

Number of Unions	Avg Time (ns)	Error Margin (%)
0	350860.0	54.69
500	3033380.0	29.61
1000	3030670.0	3.63
1500	4371220.0	3.63
2000	5642350.0	2.66
2500	6517490.0	0.83
3000	8149010.0	1.76
3500	9276320.0	3.42
4000	9847300.0	1.01
4500	1.520369E7	1.6
5000	1.599214E7	2.73
5500	1.661375E7	1.48
6000	1.61044E7	0.91
6500	1.921276E7	2.08
7000	1.979681E7	1.55
7500	2.377759E7	5.15
8000	2.454743E7	3.86
8500	2.446386E7	1.52
9000	2.789283E7	2.06
9500	2.70064E7	2.39

QuickUnion

Number of Unions	Avg Time (ns)	Error Margin (%)
0	249090.0	53.18
500	204960.0	22.84
1000	301210.0	9.92
1500	240040.0	3.46
2000	161230.0	3.44
2500	232160.0	3.72
3000	272380.0	1.62
3500	366220.0	3.33
4000	445520.0	11.07
4500	682310.0	3.48
5000	1067580.0	2.5
5500	2136640.0	10.95
6000	7733200.0	6.19
6500	2.562241E7	5.34
7000	3.54833E7	3.5
7500	5.347991E7	4.05
8000	8.024178E7	2.51
8500	1.0389962E8	3.51
9000	1.4147681E8	2.43
9500	1.9772992E8	3.45

WeightedUnion

Number of Unions	Avg Time (ns)	Error Margin (%)
0	284350.0	39.61
500	241250.0	16.34
1000	314580.0	3.87
1500	300790.0	4.33
2000	181780.0	2.43
2500	217180.0	3.61
3000	305360.0	3.67
3500	276990.0	6.1
4000	284730.0	1.09
4500	425910.0	2.5
5000	622080.0	1.82
5500	813860.0	4.01
6000	892440.0	1.22
6500	1160440.0	5.45
7000	1174500.0	2.04
7500	1078930.0	1.14
8000	1233850.0	3.64
8500	1359170.0	1.99
9000	1410330.0	1.64
9500	1961390.0	2.72

Comparative

Number of Unions	Avg QF (ns)	Avg QU (ns)	Avg WU (ns)
0	350860.0	249090.0	284350.0
500	3033380.0	204960.0	241250.0
1000	3030670.0	301210.0	314580.0
1500	4371220.0	240040.0	300790.0
2000	5642350.0	161230.0	181780.0
2500	6517490.0	232160.0	217180.0
3000	8149010.0	272380.0	305360.0
3500	9276320.0	366220.0	276990.0
4000	9847300.0	445520.0	284730.0
4500	1.520369E7	682310.0	425910.0
5000	1.599214E7	1067580.0	622080.0
5500	1.661375E7	2136640.0	813860.0
6000	1.61044E7	7733200.0	892440.0
6500	1.921276E7	2.562241E7	1160440.0
7000	1.979681E7	3.54833E7	1174500.0
7500	2.377759E7	5.347991E7	1078930.0
8000	2.454743E7	8.024178E7	1233850.0
8500	2.446386E7	1.0389962E8	1359170.0
9000	2.789283E7	1.4147681E8	1410330.0
9500	2.70064E7	1.9772992E8	1961390.0

Figure 12 - Tables of Execution Times for Varying Numbers of Unions

Tea Elming
1DV018
Sept 2024

Summary

Based on the tests above, the WeightedUnion appears to be the best alternative if only one can be implemented. There was not a significant difference in execution time for the find-operations, but it was an impressive difference for the Union-operations, where the WeightedUnion was far more reliable and outperformed the other two options.