DATA STRUCTURES AND ALGORITHMS

16:332:573

ECE RUTGERS SPRING 2018

HOMEWORK – 1

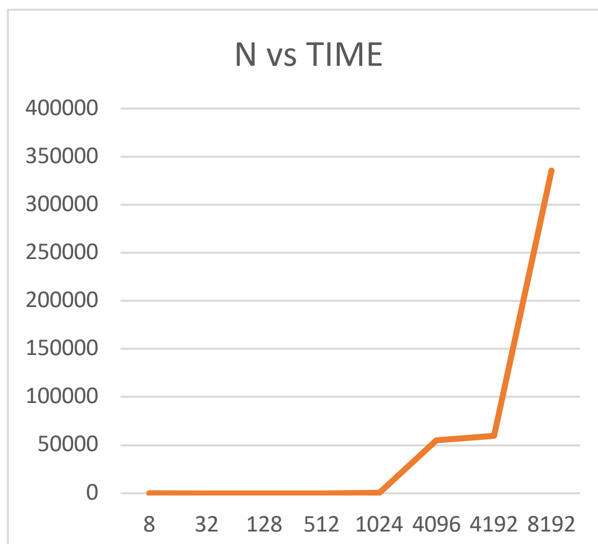
SOLUTION AND IMPORTANT POINTS

Q1: 3-sum problem

Naive implementation of 3-sum problem ($O(N^3)$)

This implementation also known as the brute-force algorithm was implemented and the time-cost is captured and logged and plotted. The order of this algorithm is $O(N^3)$ because the sum is calculated inside the 3-nested loop.

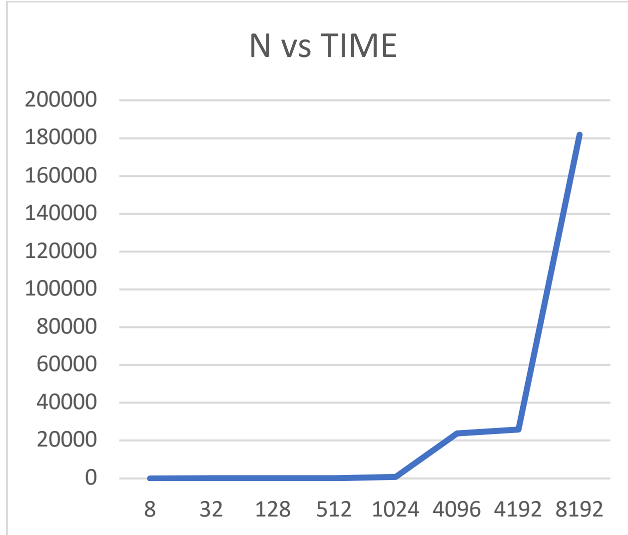
N	TIME(Naive) (in ms)
8	0.4
32	2.368
128	27.508
512	59.271
1024	334.343
4096	55197.534
4192	59705.585
8192	335568.364



Sophisticated implementation of 3-sum problem ($O(N^2 logN)$)

This implementation is also called binary-search based linearithmic algorithm was implemented and the time-cost is captured and logged and plotted. The order of this algorithm is $O(N^2 logN)$ because the sum is calculated inside the 2-nested loop with the third element searched using binary search whose summation gives the value 0.

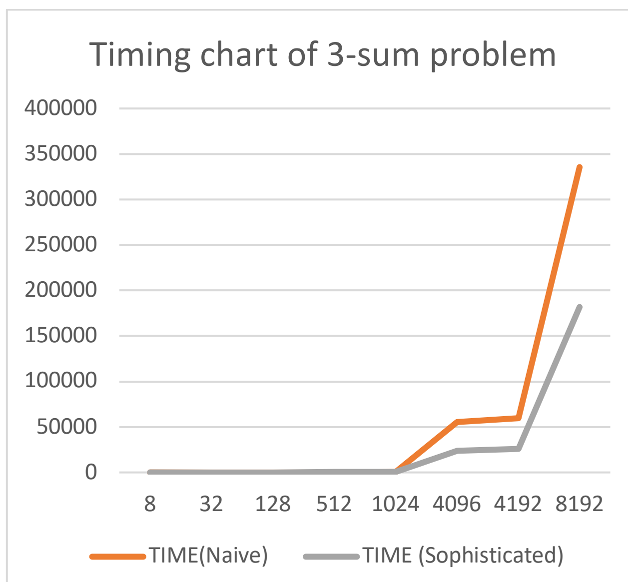
N	TIME (Sophisticated) (in ms)
8	1.269
32	6.659
128	37.541
512	232.054
1024	763.733
4096	23837.512
4192	26025.181
8192	181814.616



<u>Comparison of the Naïve and Sophisticated algorithms</u>

We see that for lower value of N, the time taken by the naïve algorithm to solve the 3-sum problem is the least. But as the order of N increases, the sophisticated algorithm shows better performance by consuming lesser time because it has lower growth rate than the naïve algorithm.

N	TIME (Naive) (in ms)	TIME (Sophisticated) (in ms)
8	0.4	1.269
32	2.368	6.659
128	27.508	37.541
512	59.271	232.054
1024	334.343	763.733
4096	55197.534	23837.512
4192	59705.585	26025.181
8192	335568.364	181814.616



Important points of Q1:

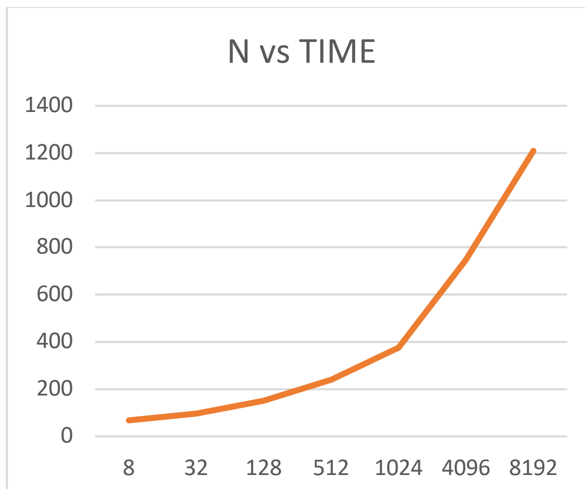
1. **Graphs, Tables and Comparison between the two implementations**
2. **Inclusion of time complexity in the analysis. So, $O(N^3)$ for naïve based approach and $O(N^2 \log N)$ for sophisticated approach.**

Q2: Union Find algorithms

(i) Quick Find

The quick find implementation is the slowest with respect to the other two implementations. In quick find, the find operation is O(1), a constant growth rate. But the union operation is expensive. For each union operation, we must go through n array elements to ensure successful implementation i.e. is O(N). Thus, for M union functions, the growth rate would be O(MN), which is a square function and expensive, with a worst case of $O(N^2)$

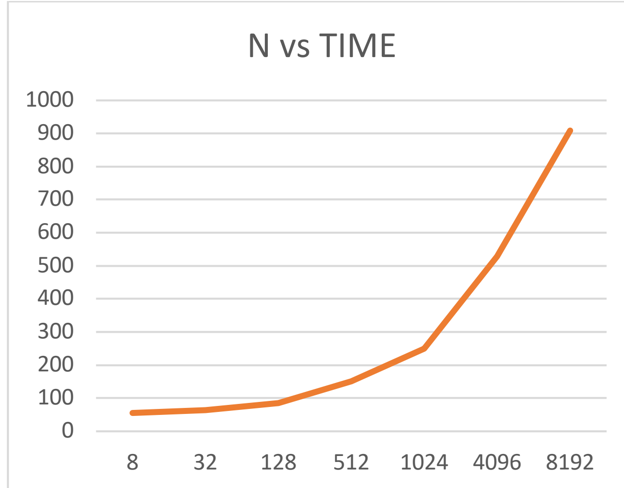
N	TIME (in ms)
8	67.82
32	96.062
128	149.858
512	241.287
1024	375.192
4096	747.392
8192	1209.469



(ii) Quick Union

The quick union is a better implementation of the union function. The find function is now expensive, which in the worst case is O(N). On the other hand, the union is faster for an average case. The number of find operations the root operation in an average case is less than N. Thus, for the average case, the m union operation is less than O(MN). But for the worst case, it is O(MN), specifically $O(N^2)$ for n union operations

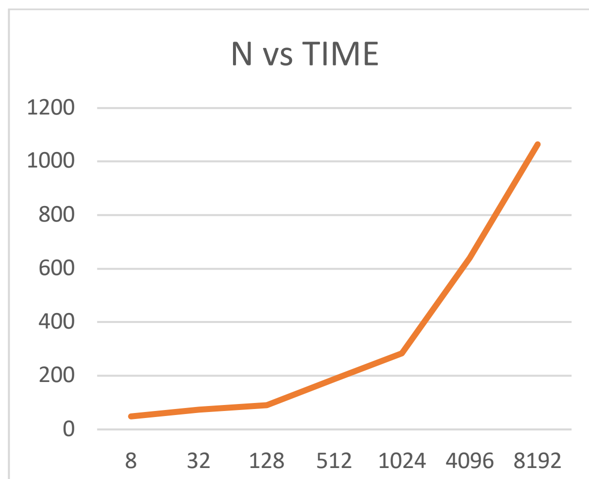
N	TIME (in ms)
8	55.42
32	63.938
128	84.45
512	150.142
1024	250.506
4096	528.749
8192	908.933



(iii) Weighted Quick Union

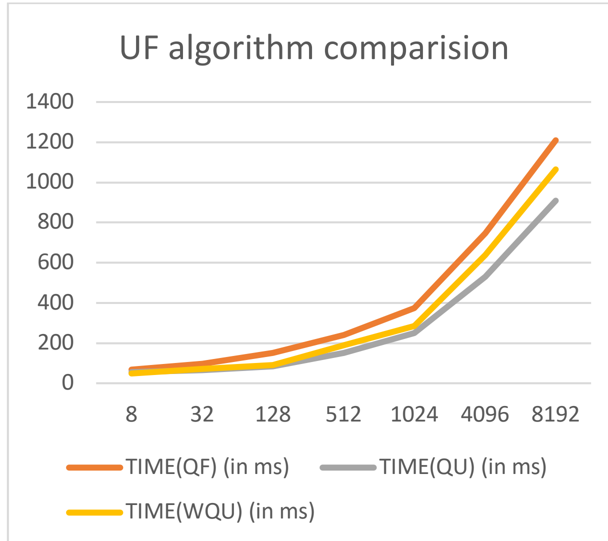
The weighted quick union is the fastest of the three. It decreases the average distance to finding a root. The union operation and the find operation on an average lead to O(logN).

N	TIME(in ms)
8	48.598
32	72.692
128	91.681
512	188.78
1024	283.895
4096	640.106
8192	1064.627



Analysis of all the three algorithms collectively:

N	TIME (QF) (in ms)	TIME (QU) (in ms)	TIME (WQU) (in ms)
8	67.82	55.42	48.598
32	96.062	63.938	72.692
128	149.858	84.45	91.681
512	241.287	150.142	188.78
1024	375.192	250.506	283.895
4096	747.392	528.749	640.106
8192	1209.47	908.933	1064.63



Important points of Q2:

1. **Graphs, Tables and Comparison between the two implementations**
2. **Inclusion of time complexity in the analysis in terms of big Oh notations as done above.**

Q4: Farthest Pair (1 Dimension)

The algorithm goes through the 1-D array once using a for loop to find the minimum and maximum value. Thus, it is a linear function with growth rate O(N), where n is the size of input data.

Important points of Q4:

1. **Correct and runnable code.**
2. **Discussion of time complexity in terms of Big Oh notation.**
3. **I have attached the code in the assignment folder. Have a look at it if there is any confusion.**

Q3: "Big Oh"

I used my TI-83 calculator to list the data points for the naive implementation and then used the cubic regression function to determine the best fit line for the data. I received the equation $0.167x^3 - 0.499x^2 + 0.333x + 1.71 \times 10^{-5}$. This means that we can find the values of c and N_C by making all of the exponents of the x variables be equal to 3 and disregarding all of the negative coefficients, so $0.167x^3 - 0.499x^2 + 0.333x + 1.71 \times 10^{-5}$ $\leq 0.167x^3 + 0.333x^3$, and from that, $0.167x^3 - 0.499x^2 + 0.333x + 1.71 \times 10^{-5} \leq 0.5x^3$, so the constant c in $f(n) \leq c(g(n))$ is equal to 0.5. Further, the value of N_C is then 1, as can be shown by substituting x with 1 in both of the equations: $0.167(3^3) - 0.499(3^2) + 0.333(3) + 1.71 \times 10^{-5} \leq 0.999(3^3) \Rightarrow 0.0010171 \leq 0.5$. For the sophisticated implementation, I also used the same calculation method, but using quadratic regression instead to obtain an equation of $7.989x^2 - 4585.799x + 816521.094$. Using this equation and knowing that the sophisticated implementation is $O(N^2 log(N))$, I used a guess and check method to try to modify the equation to obtain $0.63log_2(x)x^2 - 1585.799x + 816521.094$ as the best fit line while still being $O(N^2 log(N))$. Now, we get the equation $0.63log_2(x)x^2 - 1585.799x + 816521.094$ $\leq 0.63log_2(x)x^2 + 816521.094x^2$, and therefore $0.63log_2(x)x^2 - 1585.799x + 816521.094$ $\leq 816521.724x^2 log_2(x)$ where c is 816521.724 and N_C is 2 because if N_C were 1, log(1) is 0.

Using the same approach for Problem 2, for quick find, the equation using linear regression is $7.679x + 0.007$, for quick union, the equation is $8.668x - 0.005$, for weighted quick find, the equation is $7.608x + 0.032$, and finally for weighted quick union the equation is $8.757x - 0.015$. From this information, the values of c and N_C can be calculated as follows: for quick find, $7.679x + 0.007 \leq 7.679x + 0.007x$ so $7.679x + 0.007 \leq 7.686x$ and the value of c is 7.686 and N_C is 1. For quick union, $8.668x - 0.005 \leq 8.668x$, so the value of c is 8.668 and N_C is 1. For weighted quick find, $7.608x + 0.032 \leq 7.608x + 0.032x$ so $7.608x + 0.032 \leq 7.64x$, and the value of c is 7.64 and N_C is 1. Finally, for weighted quick union, $8.757x - 0.015 \leq 8.757x$, so the value of c is 8.757 and N_C is 1.

Important points on Q3:

1. **Analysis on how to get c, Nc.**
2. **Get values of c, Nc based on the analysis.**
3. **Some people have not used curve fitting to get c, Nc and have evaluated the values using the equations. Those answers are fine as well.**

Q5: Faster-est-ist 3-sum

For the two pair implementation, I iterated through the sorted number n in one pass, with two iterators at the beginning and at the end. I observed if sum is less than 0, thus we have needed a bigger value. Hence you mover the iterator in the beginning. Similarly, if sum is more than 0, thus we have needed a smaller value and you move the iterator at the end. This is results in an O(N) growth rate. Using that same thinking in the three-sum implementation, we again fix one value of the element and fix two iterators at the beginning and at the end. I observed if sum is less than 0, thus we have needed a bigger value. Hence you mover the iterator in the beginning. Similarly, if sum is more than 0, thus we have needed a smaller value and you move the iterator at the end. Thus, this leads to an $O(N^2)$ growth rate.

Important points:

1. **Correct and runnable code.**
2. **Discussion of time complexity in terms of Big Oh notation.**