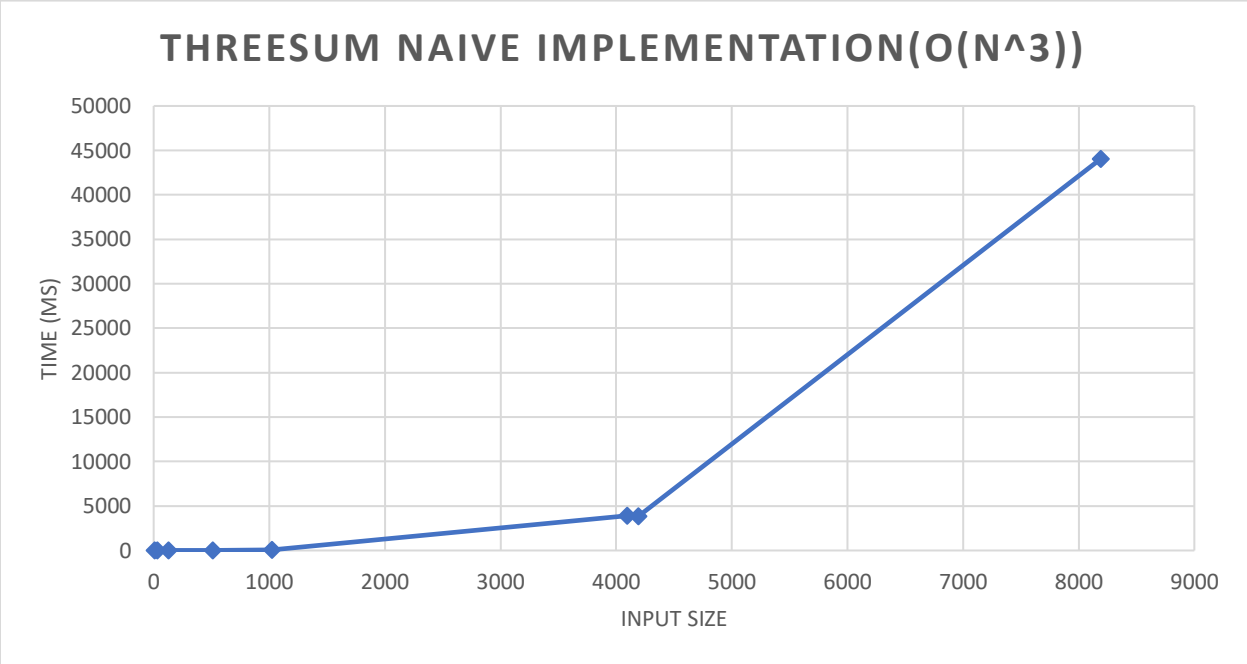
Project Report

Assignment 1

Q1 - > a)

Analysis of Naïve Implementation of O(N^3)

Plot :

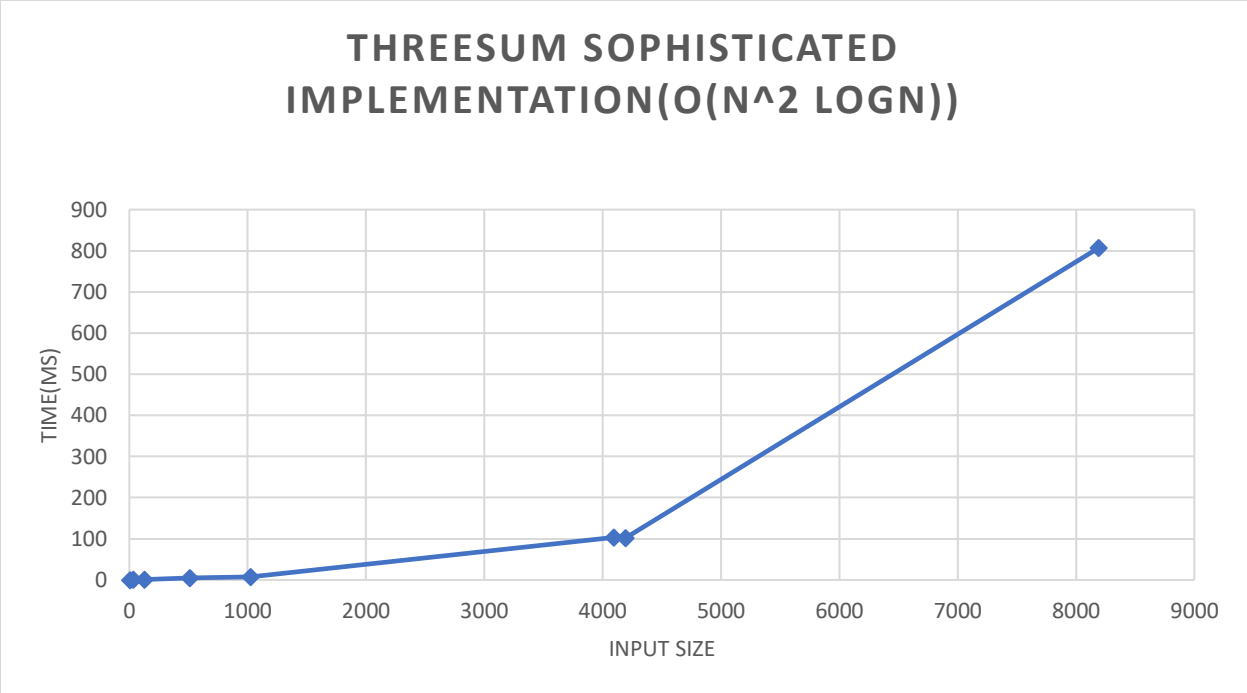


From the above plot of Input size vs time cost of execution, we can interpret that the execution of input sizes of 8,32,128 & 512 is almost linear and for input sizes 1024,1096, 8192 the three-sum algorithm takes almost double time for execution. For 8192 inputs the time is 44065 µs.

Since the three-sum problem is looking for the pairs of three numbers that sum to zero and the supplied inputs have no pair that sums up to zero, so it becomes the worst case for all input sizes. Also, theoretically we can say that the complexity of algorithm is O(N^3) so the graph must be exponential. This is proved as input size is directly proportional to time cost and graph seems to be growing exponentially.

b) Analysis of sophisticated Implementation of O(N^2 lg N)

Plot :



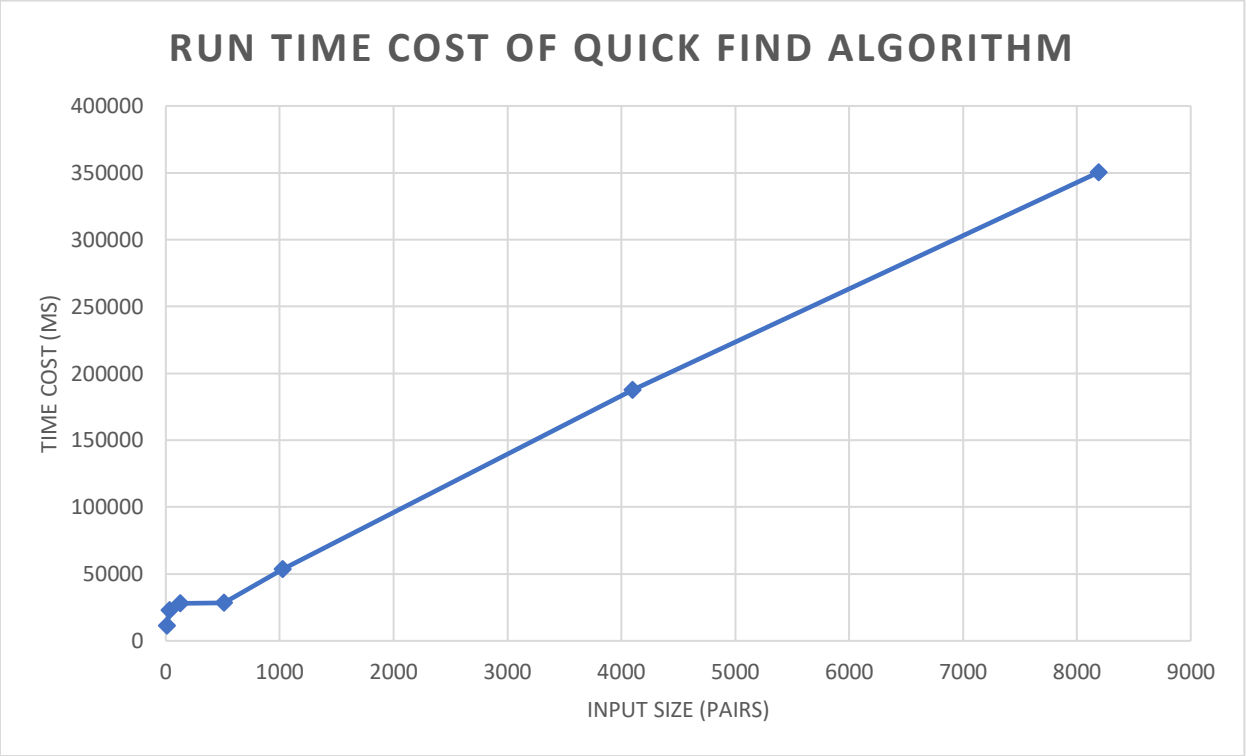
From above graph we can interpret that the cost of time execution is less than that of the brute force implementation. The graph shows exponential growth with input sizes. Since the inputs depicts worst case scenario of no pair of three that sums to zero the algorithm runs for the worst case and still performs better than the naïve implementation because of the sorted input array to the algorithm.

Comparing both algorithms, we can say that the graphs are almost same, but the only difference is the execution time. which is very less in the O(N^2logN) implementation because of the absence of an extra loop to go through for finding the pair.

Q2 - >

Analysis of Quick Find Algorithm:

Plot :

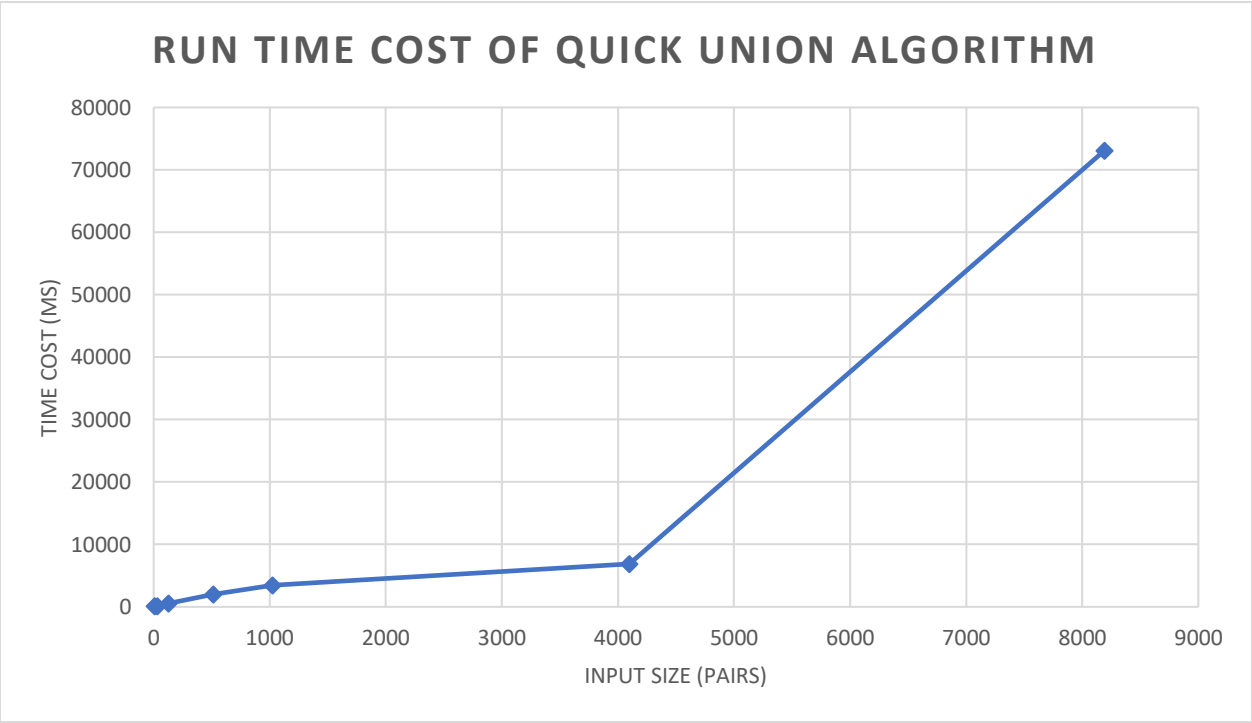


From the above graphs we can see that the graph has increasing upward trend with the input data pairs and the execution time. Time cost is directly proportional to number of input pairs. For the input pairs of 8184 the union operation takes less time while for the 1314 components union takes more time for merge operation. When there are many components its easy to form pairs , so this requires less time and in the other case when the there are less components algorithm needs to traverse to root and connect to the other parent where the pair needs to be formed.

Complexity of the union operation in quick find is O(N) since it takes N steps for merge and for find operation is O(1) since the trees formed are flat and have less dept so the combine complexity cones out to be O(N) for the worst case.

Analysis of Quick Union Algorithm:

Plot :

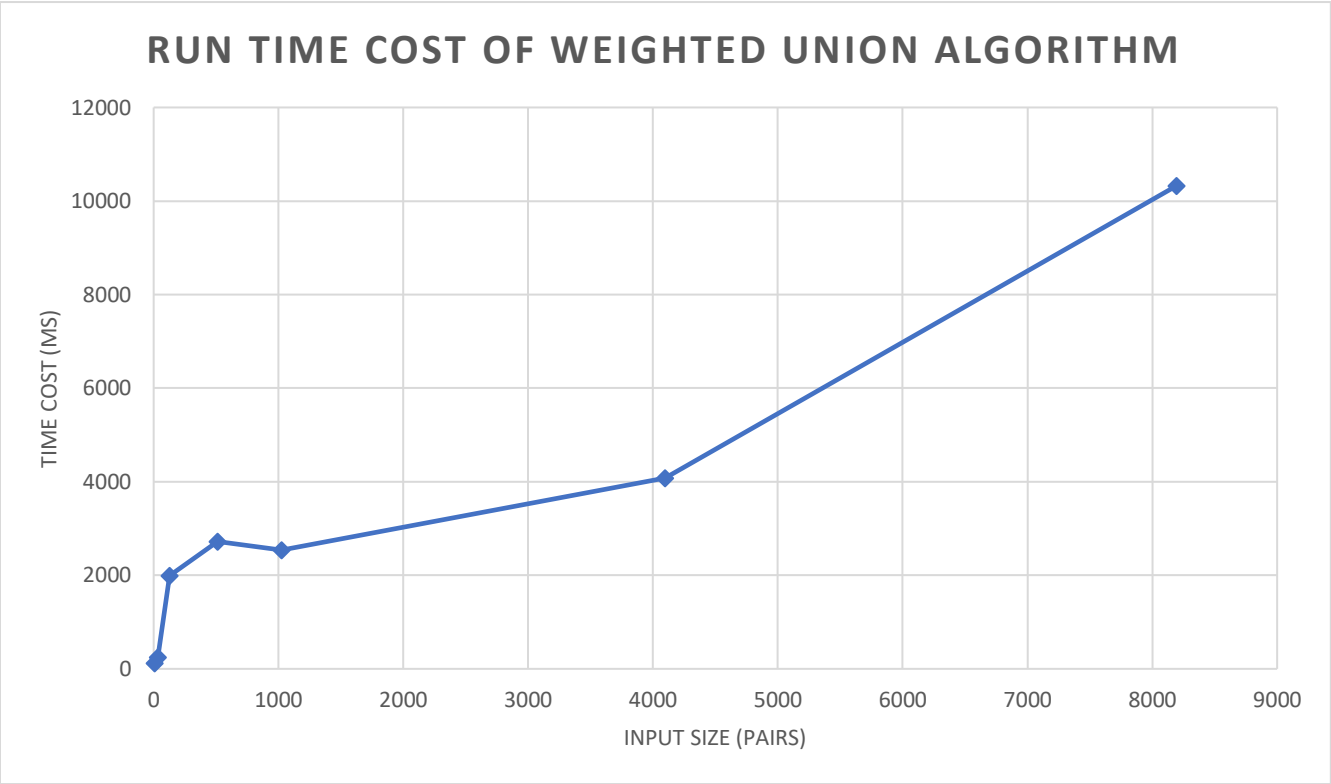


From the above graph we can see that it has the same increasing trend similar to quick find but the time cost is half of quick find, means quick union is efficient than quick find. In Quick union we have to change the root of the child nodes instead of changing the index of all the nodes to similar number. So, here the tree grows tall instead of flat in quick find.

Every time for performing a union operation we have to perform Find operation. So, complexity of union operation is O(N) (which includes the cost of find operation) and for find operation is also O(N) so the overall complexity is O(N) for the worst case.

Analysis of Weighted Quick Union Algorithm:

Plot :



Weighted quick union is the improved version of quick union, from the graph we can interpret that the time cost is less than that of quick find & quick union. It has a upward trend , time increases with the increase in input data size. Here in weighted union size of tree is tracked , smaller tree always goes below the larger tree So , in the 8184 inputs, we have formed the initial trees, which requires less time. when coming down to 1134 components, trees needs to be balanced and there are too many trees to look for, hence the time is increased.

So, the complexity of union and find operation comes out to be log(N) as the trees are balanced with larger sized trees at top and small sized trees at bottom.

Q3 ->

Analysis of Big(O):

Let f and g be functions from positive numbers to positive numbers. f(n) is O(g(n)) if there are positive constants C and k such that: f(n) ≤ C g(n) whenever n > k f(n) is O(g(n)) ≡ ∃C ∃k ∀n (n > k → f(n) ≤ C g(n)) . Choose values for Nc that stratifies n > k implies f(n) ≤ C g(n).

f(n) = O(g(n)) means there are positive constants c and k, such that 0 ≤ f(n) ≤ cg(n) for all n ≥ k. The values of c and k must be fixed for the function f and must not depend on n.

In Q1 ,

For calculating the Nc for Three Sum problem using Brute Force there are three nested 'for' loops each costing the time for 'n'. So, overall it comes to be (n*n*n) N^3.

Three sum with sophisticated implementation has 2 nested loops costing (n*n) and we are performing binary Search in the nested loop costing worst case for (log n).

The estimated Nc value for Q1 comes to be O(n^3 + n^2 log n) which comes out to be O(n^3).

In Q2,

There are three algorithms Union Find, Quick Find, Weighted Quick Union.

i) Quick Find :
In this algorithm, find operation just returns the index, and union operation merges the index to form tree. In union operation we are using one nested 'for' loop which costs O(n) for union and for find operation it takes constant time for traversing and finding the index through the array, so it comes out to be O(1).
Combined Nc value for this would be [O(1) + O(n)] that comes out to be O(n)

ii) Union Find:

In this algorithm, the union operation costs N computational steps which includes finding of N steps of find operation. So, the combined Nc of this algorithm will be O(n + n) equals to O(n).

iii) Weighted Quick Union:
In this algorithm, it is similar to union-find the union operation includes the complexity of logN with the find operation. Both the operation costs LogN.
Combined Nc value of this algorithm is O(logN).

So, in Q2, Nc comes out to be O(n + n + logn) which equals to O(n).

Q4 ->

Analysis of Farthest Pair :

```
Result : Farthest pair are numbers:1.0 95.0
Time cost is 41
```

In this algorithm an array of size 100 has been generated randomly, after that running a for loop for finding minimum and maximum from the array to get the farthest pair with highest difference.

There are two 'for' loops one for generating a random array and another for finding min and max of the array . So, the complexity comes out to be O(n + n) for which the overall complexity comes out to be O(n). which shows that the complexity is linear in worst case.

Q5 ->

Analysis of Faster3sum using HashMap

```
Output :
Time cost is 0 ms
Time cost is 1 ms
Time cost is 5 ms
Time cost is 20 ms
Time cost is 30 ms
Time cost is 80 ms
Time cost is 62 ms
Time cost is 183 ms
```

Complexity : O(N^3)

Faster3sum using HashMap gives us the best result for 3sum problem. The loops ends to the worst case for performing quadratic algorithm. The output is the time

cost for iterating over the inputs supplied respectively like from 8,31,128 to 8192 inputs. For 8192 inputs the time cost is 183 milli seconds which is worst case time execution when there is no three-sum pair that equals to zero. The time cost is less than that of the naïve implementation of O(N^3) for the same inputs supplied. Which proves that HashMap performs better than the regular 3sum using brute force.