# Assignment 1

**Name:Tian Zhou**

**NETID:tz164**

## Q1:

| N | lgN | naive T/s | sophisticated T/s |
|---|---|---|---|
| 8 | 3 | 0 | 0 |
| 32 | 5 | 0. 005 | 0. 004 |
| 128 | 7 | 0. 41 | 0. 074 |
| 512 | 9 | 24. 135 | 1. 106 |
| 1024 | 10 | 192. 546 | 4. 461 |
| 4096 | 12 | | 74. 35 |

According to the data we get from the program,we plot two figures.

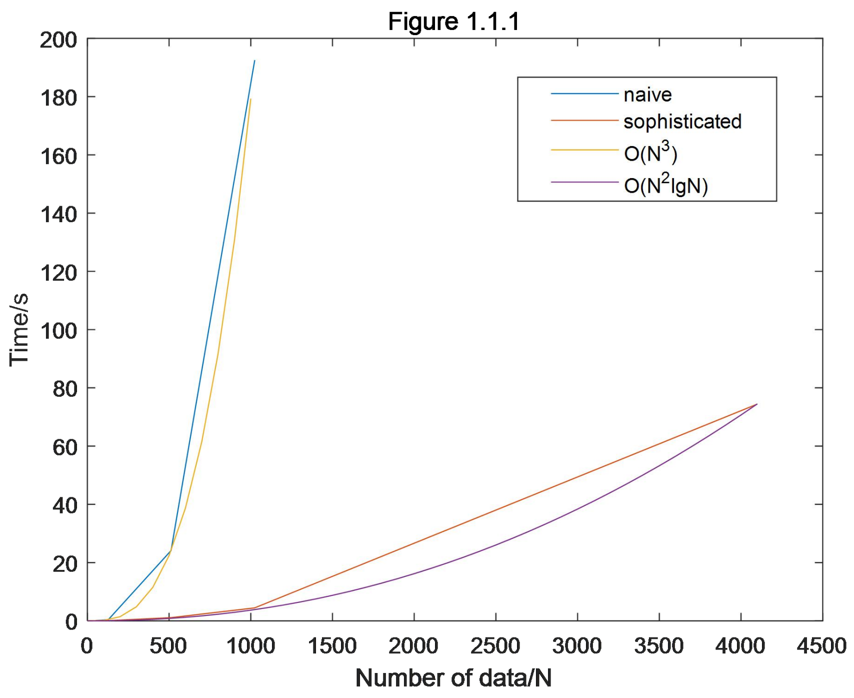


Figure 1.1.1: In this figure, the x-axis is the number of data(size of data) and the y-axis is the time cost for the implementation. Then we draw two functions **($T=N^3/5.6*10^5$)** and **($T=N^2lgN/2.7*10^6$)** to compare two lines of data below. Then we can assume that the "naive" implementation is $O(N^3)$ and the "sophisticated" implementation is $O(N^2 \lg N)$.

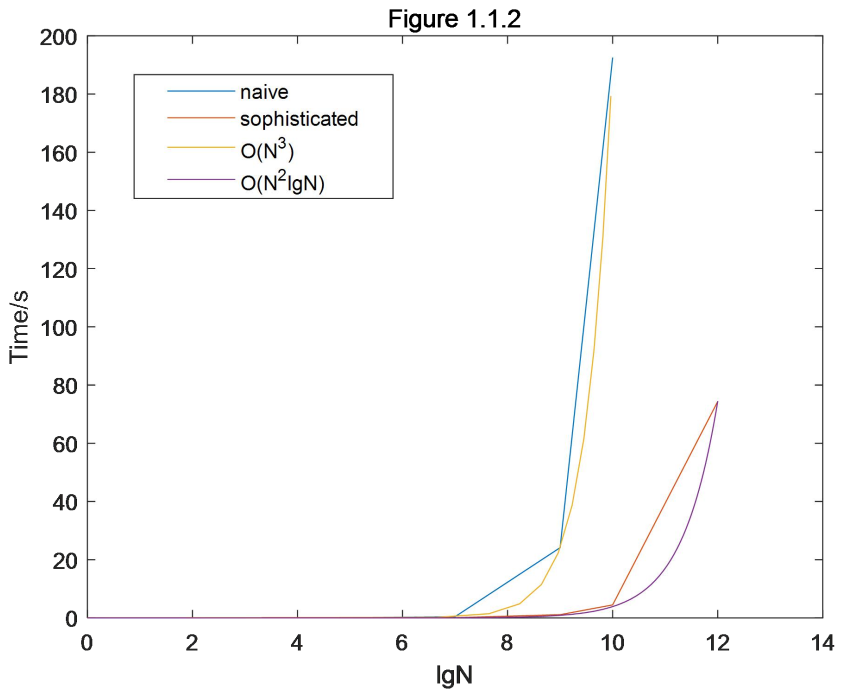


Figure 1.1.2: Also, we draw another figure with x-axis represent lgN(N is the number of data).

## Q2:

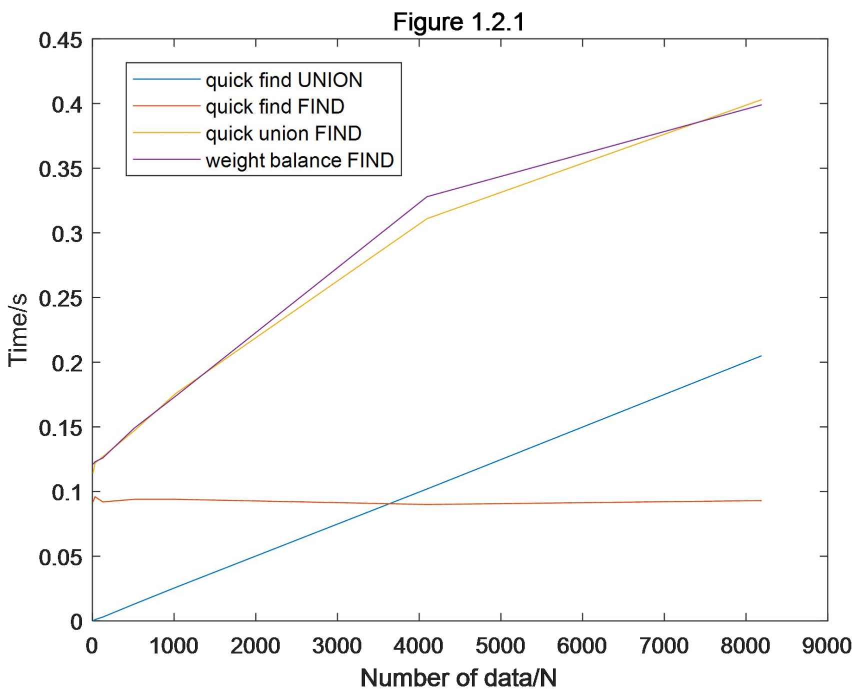
| N | qf UNION | qf FIND | qu FIND | wb FIND |
|---|---|---|---|---|
| 8 | 0 | 0.092 | 0.114 | 0.121 |
| 32 | 0.001 | 0.096 | 0.122 | 0.123 |
| 128 | 0.003 | 0.092 | 0.127 | 0.126 |
| 512 | 0.013 | 0.094 | 0.147 | 0.149 |
| 1024 | 0.026 | 0.094 | 0.176 | 0.174 |
| 4096 | 0.102 | 0.09 | 0.311 | 0.328 |
| 8192 | 0.205 | 0.093 | 0.403 | 0.399 |

*Time cost of UNION for quick union and quick union with weight balance is smaller than 0.001(both of them is a linear algorithm), so we could ignore it.

1. When we UNION the data by using quick find method, the program only run one step for each data of connection. So, the UNION of quick find is a linear algorithm.(O(N)).
2. When calling FIND function for quick find, the only thing we should do is to

compare whether the number of value in the array is the same. So , the cost of time which depending on the size of database should be a constant.

3. When calling FIND funcion for quick union and quick union with weight balance, I take the (hw1-1.data.zip) as a sample. As we can see in the figure below, it will cost more time for quick union and quick union with weight balance than quick find while calling the FIND function.



## Q3:

For Q1:

We choose the data from 1024int.zip.

We can assume that **($T=N^3/5.6*10^5$)** for naive implementation and **($T=N^2\lg N/2.7*10^6$)** for sophisticated implementation.

Let $c=1/5.6*10^5$, T(N) is always smaller than $cN^3$. (naive implementation)

Let $c=1/2.7*10^6$, T(N) is always smaller than $cN^2\lg N$. (sophisticated implementation)

Now,when for all N>0 , T(N)<c(g(N)) is satisfied. Nc is 0.

## Q4:

When we find the maximum value and minimum value in the array we could figure out the largest difference for a pair.

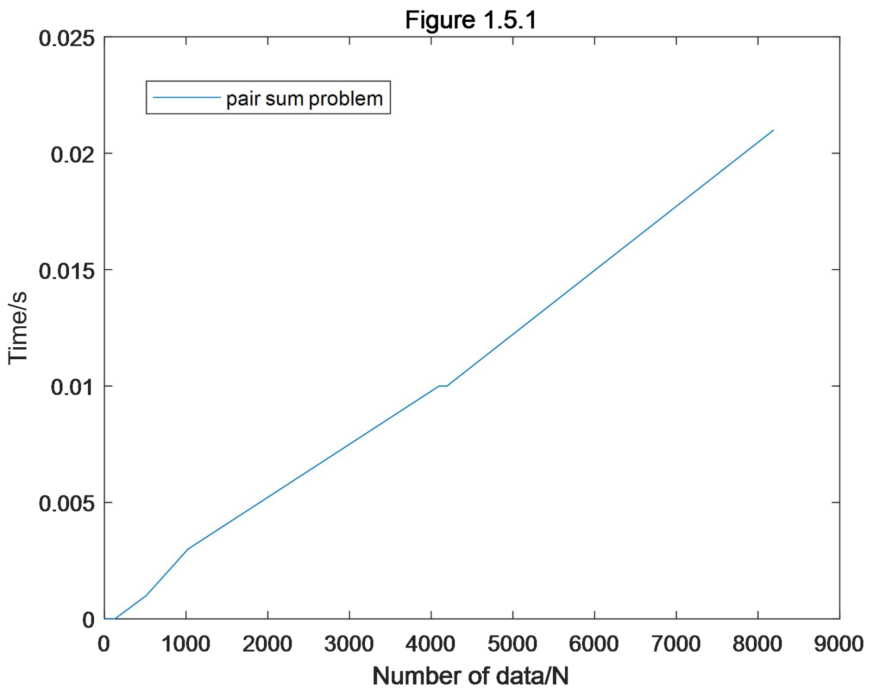
For each value in the array, we should know whether it's larger than the current maximum value or smaller than current minimum value. (Assume we compare with current maximum value first.)

For the best case, we will compare each value only once. In that case, the number of steps is N-1,which is linear algorithm.
For the worst case, we will compare each value twice with both current maximum value and minimum value. The number of steps is 2N-1, which is also linear algorithm.

## Q5:

**1) Count the number of pairs that sum to zero after the array is sorted.**
Solution: Each pair could be divided as part1 which is smaller than 0 and part2 which is larger than 0. We start searching part1 from minimum side and part2 from maximum side. When the sum of part1 and part2 is smaller than 0, part1 should move to next point that is larger than itself. On the contrary, part2 should also move to a smaller one when the sum is larger than 0. When the sum is equal to 0, both part1 and part2 move to next point.

The worst case: N-1(linear algorithm)
The best case: N/2(linear algorithm)

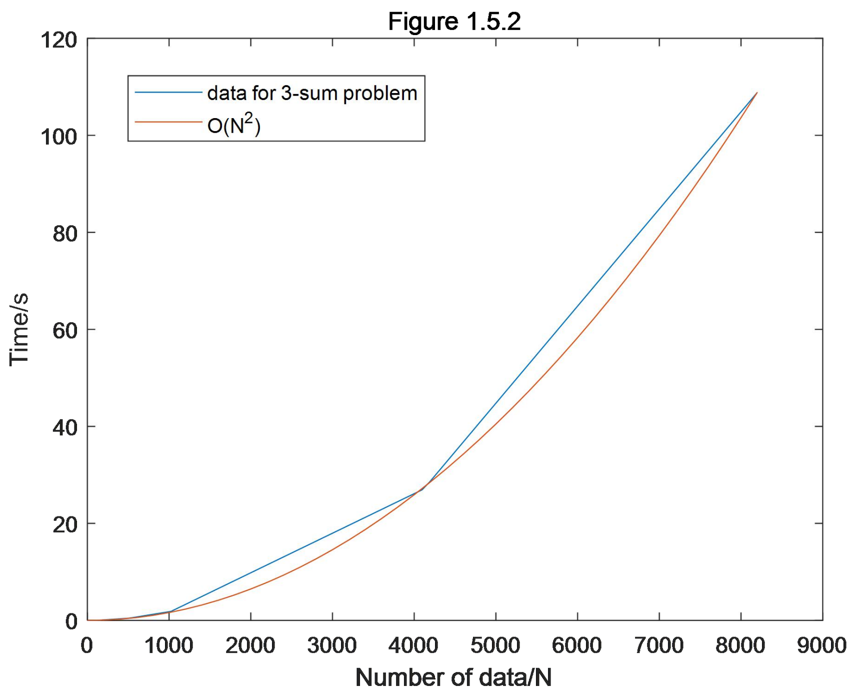


In the figure 1.5.1, it is obvious that the figure is a straight line which means the implementation is a linear implementation.

**2) A quadratic algorithm for the 3-sum problem.**
Given N sorted(assume ascending sort) values store in the array a[].
i. Pick up a pivot from a[0] to a[N].
ii. When the pivot is a[x], count the number of pairs that sum to -a[x] from a[x+1] to

a[N] by using the implementation introduced above.

The worst case: (N-1)(N-1) (quadratic algorithm)
The best case: (N-1)N/2 (quadratic algorithm)



In the figure 1.5.2, we compare the shape between the data and **$T(N)=n^2/6.2*10^5$**, we can see it’s fitting well.