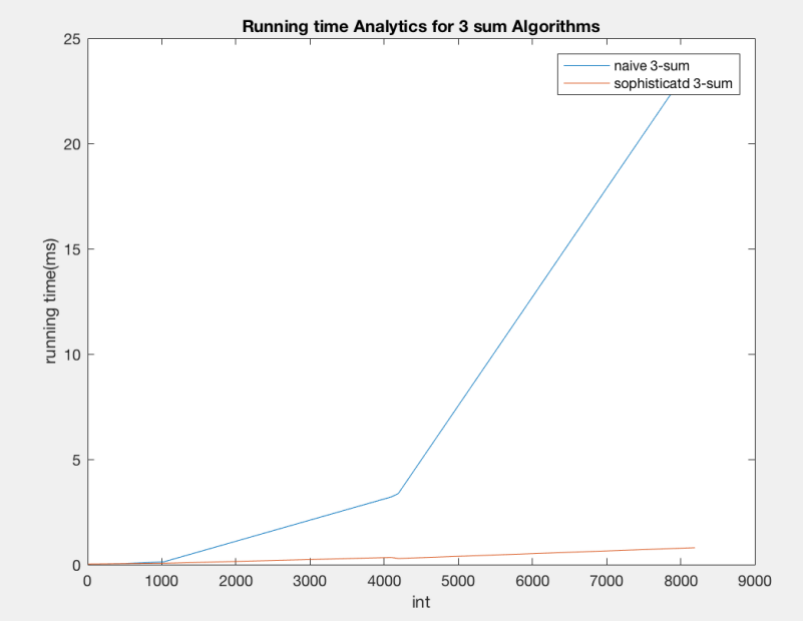
HW1

(Data Structure and Algorithms)

Name: Xiaoyi Tang
RUID: 174008332

Q1:

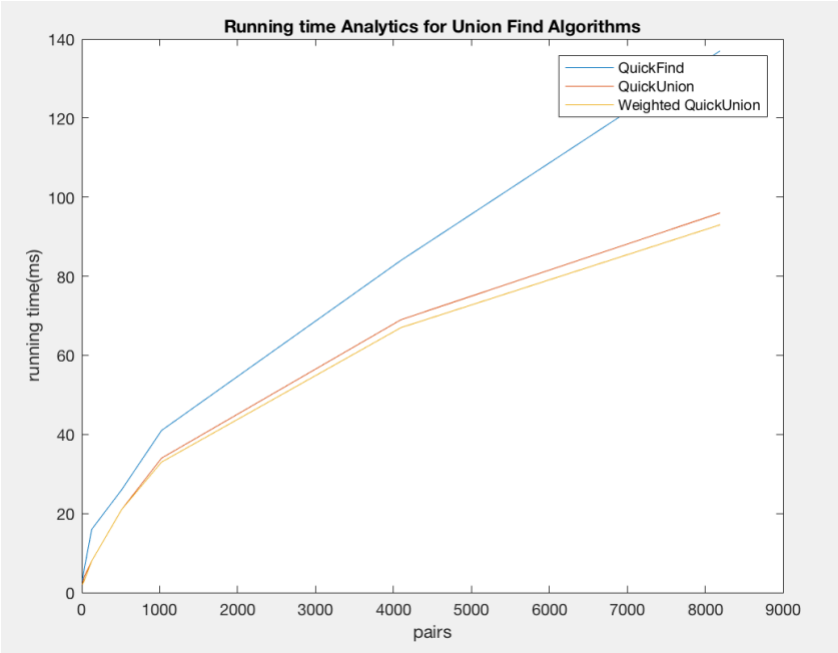


- Q1_a:
  - In this program, I used three for loops and made one calculate and one judgement in the most inside loop, so the time complexity for my program should be
    $$F(N) = 2N^3$$
  - As we can see from the graph (the blue line), as the size of input increased, the running time also increased approximately as the function
    $$O(N) = cN^3$$

- Q1_b:
  - In this program, I used two for loops and one binary search before that, in the binary search, I made one calculate and one judgement, so the time complexity of my program is
  - $F(N) = 2N^2 logN$
  - As we can see from the graph (the red line), as the size of input increased, the running time also increased approximately as the function
    $$O(N) = cN^2 logN$$

Codes: threesum_naive.java and threesumbinary.java in folder Q1

Q2.



- Q2_a:
  - In quick find algorithm, the time complexity of my program is as below:

$$F(N) = Mtimes\ union$$
$$= M * N$$

    where M represents the pairs and N represents the nodes
  - As a result, with the increasing pairs, the running time also increases in order $O(n)$. When the number of pairs is too small, the error is relatively big, as we can see from the graph, the running time line becomes linear gradually.
- Q2_b:
  - In quick find algorithm, the time complexity of my program is as below:

$$F(N) = Mtimes\ union$$
$$= M * N$$

    where M represents the pairs and N represents the nodes
  - As a result, with the increasing pairs, the running time also increases in order $O(N)$
- Q2_c:
  - In quick find algorithm, the time complexity of my program is as below:

$$F(N) = Mtimes\ union$$
$$= MlgN$$

  - As a result, with the increasing pairs, the running time also increases in order $O(lgn)$

Q3.

- Q1_a:

$$F(N) = time\ for\ reading\ data + time\ for\ operations$$
$$= N + N(N-1)(N-2) * 2$$

  - And we also know that $G(N) = cN^3$ and $F(N) < G(N)$.
  - If we choose $c = 2$, we can get that $N > 1$, as a result, $N_c = 1$.

- Q1_b:

$$F(N) = time\ for\ reading\ data + time\ for\ operation$$
$$= N + N^2 lgN * 2$$

  - And we also know that $G(N) = cN^2 lgN$ and $F(N) < G(N)$.
  - If we choose $c = 3$, we can get that $N > 2$, as a result, $N_c = 2$.

- Q2_a:

$$F(N) = time\ for\ finding\ and\ union$$
$$= (1 + N) * M$$

  - And we also know that $G(N) = N$ and $F(N) < G(N)$.
  - If we choose $c = M + 1$, we can get that $N > 1$, as a result, $N_c = 1$.

- Q2_b:

$$F(N) = time\ for\ finding\ and\ union$$
$$= (lgN + N) * M$$

  - And we also know that $G(N) = N$ and $F(N) < G(N)$.
  - If we choose $c = M + 1$, we can get that $N > 2$, as a result, $N_c = 2$.

- Q2_c:

$$F(N) = time\ for\ finding\ and\ union$$
$$= (lgN + N) * M$$

  - And we also know that $G(N) = N$ and $F(N) < G(N)$.
  - If we choose $c = M + 1$, we can get that $N > 2$, as a result, $N_c = 2$.

Q4.

- In my implementation, I traverse the data for only one time, and during that, I will find the biggest and smallest value at the same time.
- Therefore, the worst case is that I traverse each data for once and determine that whether it is the biggest or smallest. So the time complexity is $O(n)$.
- Code: FarthestPair.java

Q5.

- In my implementation, I traverse the sorted array for only once, and inside the only for loop, I used two pointers to traverse all data bigger than current data. I will calculate the sum of three data and move the two pointers according to the absolute value between current sum and 0
- Therefore, the worst case is that I traverse each data for twice totally, so the time complexity is $O(n^2)$.
- Code: FasterThreeSum.java