DSA Homework 1

Ruiyu Zhang | rz213

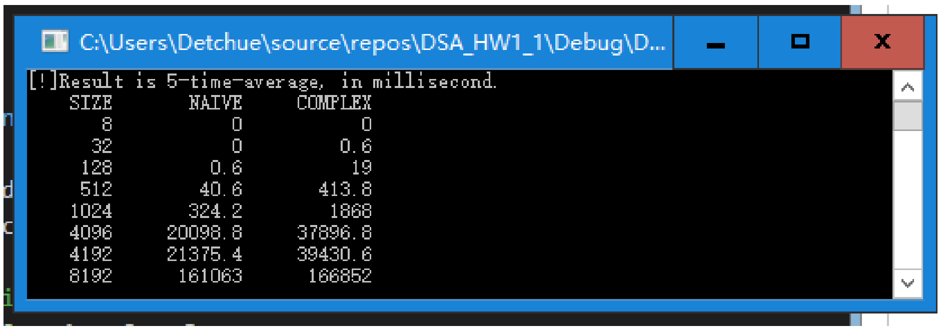
Q1. We discussed two versions of the 3-sum problem: A "naive" implementation (O(N^3)) and a "sophisticated" implementation (O(N^2 lg N)). Implement these algorithms. Your implementation should be able to read data in from regular data/text file with each entry on a separate line. Using Data provided under resource (hw1-1.data.zip) to determine the run time cost of your implementations as function of input data size. Plot and analyze (discuss) your data.

A:

[!]Cpp file is attached along in Sakai.

All results are **5-time-calculation-average.** Time shown in "millisecond".

	8	**32**	**128**	**512**	**1024**	**4096**	**4192**	**8192**
naive	0	0	0.6	40.6	324.2	20098.8	21375.4	161063
complex	0	0.6	19	413.8	1868	37896.8	39430.6	166852

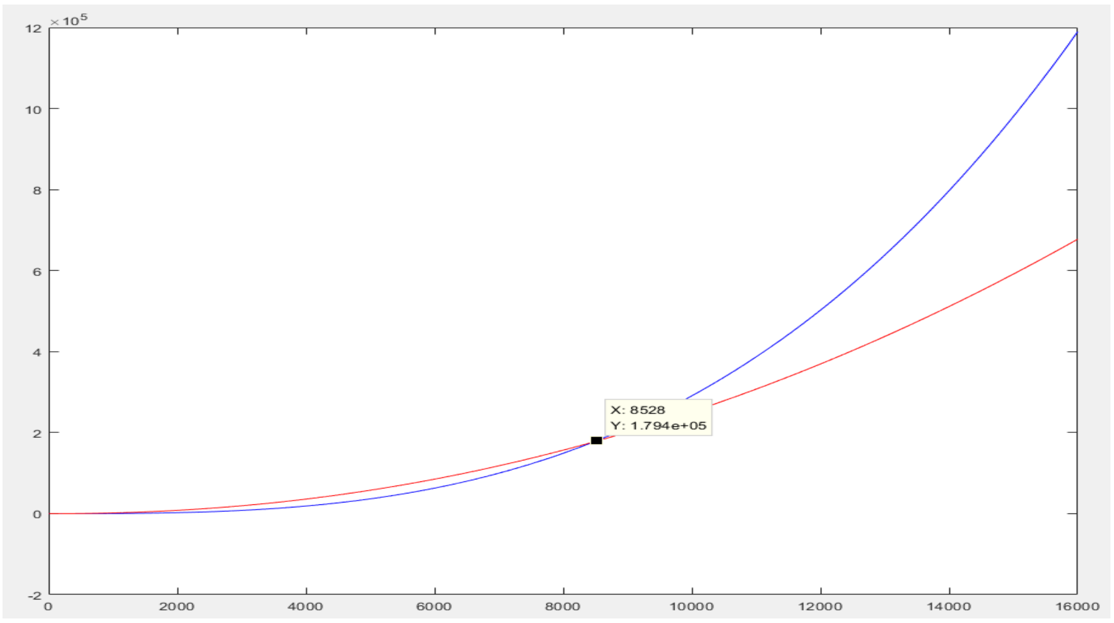


```
[!]Result is 5-time-average, in millisecond.
    SIZE        NAIVE      COMPLEX
       8            0            0
      32            0          0.6
     128          0.6           19
     512         40.6        413.8
    1024        324.2         1868
    4096      20098.8      37896.8
    4192      21375.4      39430.6
    8192       161063       166852
```

Naïve algorithm $T=0.0000002901N^3+0.000004341N^2+0.02493N-8.26$

Complex algorithm $T=0.0001901N^2 * \log_2(N)-0.131N+14.99$

#estimated based on MatLab2017b Curve Fitting Tool



Blue: Naïve Red: Complex X: N(data size); Y: T(time consumption)

#Plotted based on MatLab2017b Curve Fitting Tool and Plot Tool.

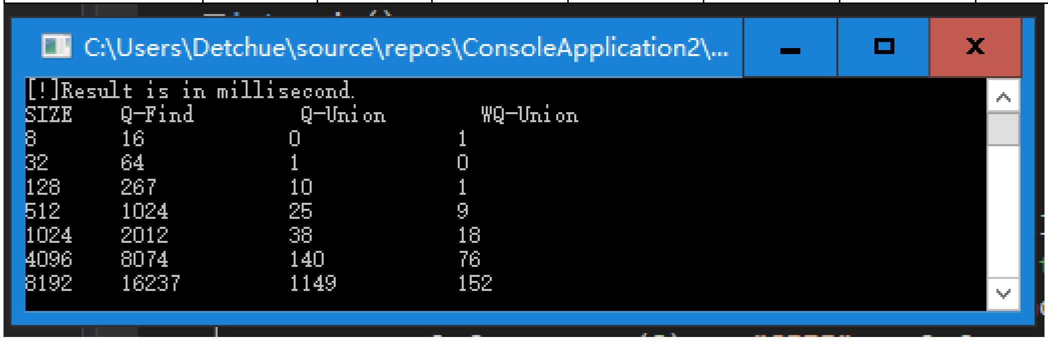
Q2. We discussed the Union-Find algorithm in class. Implement the three versions: (i) Quick Find, (ii) Quick Union, and (iii) Quick Union with Weight Balancing. Using Data provided here (hw1-2.data.zip, under resources) determine the run time cost of your implementation (as a function of input data size). Plot and analyze your data. Note: The maximum value of a point label is 8192 for all the different input data set. This implies there could in principle be approximately 8192 x 8192 connections. Each line of the input data set contains an integer pair (p, q) which implies that p is connected to q.

A:

[!]Cpp file is attached along in Sakai.

Time shown in "ms"

	8	**32**	**128**	**512**	**1024**	**4096**	**8192**
Quick Find	16	64	267	1024	2012	8074	16237
Quick Union	0	1	10	25	38	140	1149
Weight Q_U	1	0	1	9	18	76	152



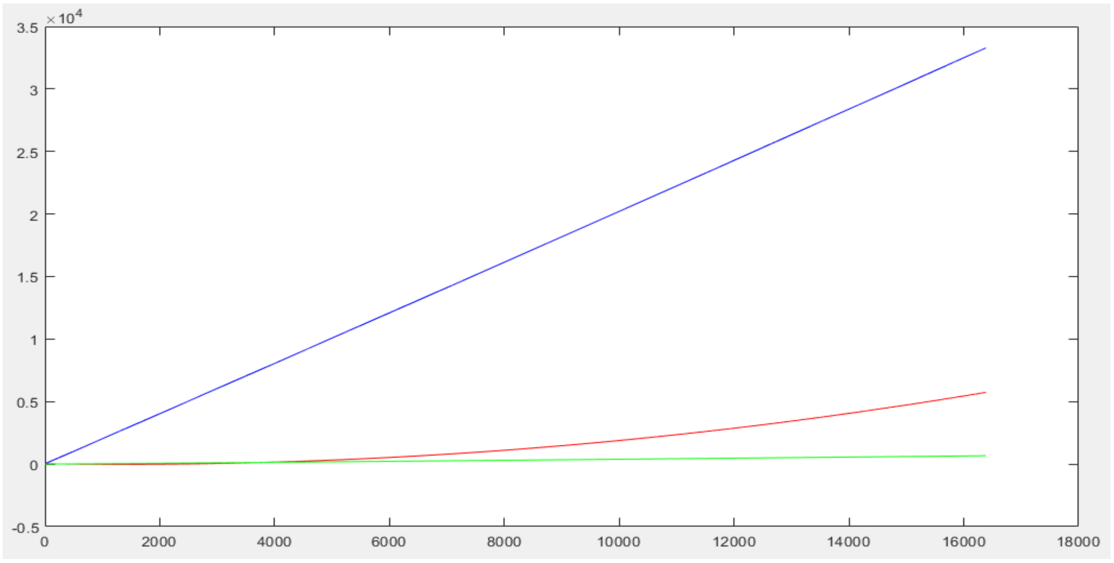
```
[!]Result is in millisecond.
SIZE    Q-Find          Q-Union         WQ-Union
8       16              0               1
32      64              1               0
128     267             10              1
512     1024            25              9
1024    2012            38              18
4096    8074            140             76
8192    16237           1149            152
```

$T(Q\_Find)=(1.72e-6)*N^2+2.002*N+23.46$

$T(Q\_Union)=(2.548e-5)*N^2-0.06881*N+27.03$

$T(Weight\_Q\_U)=0.002901*N*\log_2 N+1.707$

#estimated based on MatLab2017b Curve Fitting Tool



Blue: Quick Sort; Red: Quick Union; Green: Weighted Quick Union

X: N(data size); Y: T(time consumption).

#Plotted based on MatLab2017b Curve Fitting Tool and Plot Tool.

Q3. Recall the definition of "Big Oh" (where F(N) is said to be in O(g(N), when F(N) < c (g(N)), for N > Nc) . Estimate the value of Nc for both Q1 and Q2. More important than the specific value, is the process and reasoning your employ.

A:

For Q1:

Now that we have fitting functions for both algorithms for Q1, shown below:
Naïve algorithm $F_1(N) = 0.0000002901N^3+0.000004341N^2+0.02493N-8.26$
Complex algorithm $F_2(N) = 0.0001901N^2 * \log_2(N)-0.131N+14.99$

Now we put N=1 into both functions, and apply absolute values for negative parts, we have:
$C_1' = 0.0000002901+0.000004341+0.02493+8.26=8.2849346311$
$C_2'=15.121$

This is almost a random set of numbers, but one thing is for sure: **$F_1(1)<C_1'$, $F_2(1)<C_2'$**.
We employ these C's as c this time, we can then make sure equation "F(N) < c (g(N)), for N > Nc" stands as long as we find a specific and suitable Nc for it.

We now have
$g(N)_1=C_1'*N^3=8.2849346311N^3$
$g(N)_2=C_2'* N^2\log_2N=15.121N^2\log_2N$
All we have to proof is that $F_1(N)< g(N)_1$ and $F_2(N)< g(N)_2$
We find that
$g(N)_1 - F_1(N) = 8.284934341N^3-0.000004341N^2-0.02493N+8.26$
$g(N)_2 – F_2(N) = 15.1208099 N^2 * \log_2(N)+0.131N-14.99$
which are apparently both increasing functions.

As we already know that $g(1)_1 - F_1(1)> 0$ and $g(1)_2 – F_2(1)> 0$, We can tell that:
For $C_1=0.0249346311$ and $C_2=14.99$, Nc=1 should work for both.

For Q2:

IT'S ALL THE SAME LOGIC THAT WE HAVE IN SOLUTION ABOVE
We have for Q2 functions below:
$F(Q\_Find)=(1.72e-6)*N^2+2.002*N+23.46$
$F(Q\_Union)=(2.548e-5)*N^2-0.06881*N+27.03$
$F(Weight\_Q\_U)=0.002901 *N*\log_2N+1.707$

We put N=1 into both functions, and apply absolute values for negative parts, we have:
$C_1=25.46200172$, $C_2=26.96121548$, $C_3=1.707$
For N=1

With all the same reasoning process that I'm not repeating on, we get Nc as below:
$C_1=25.46200172$, $C_2=26.96121548$, $C_3=1.707$, Nc=1 works for all three.

Q4: Farthest Pair (1 Dimension): Write a program that, given an array a[ ] of N double values, find a farthest pair: two values whose difference is no smaller than the difference of any other pair (in absolute value). The running time of the program should be LINEAR IN THE WORST CASE.

A:

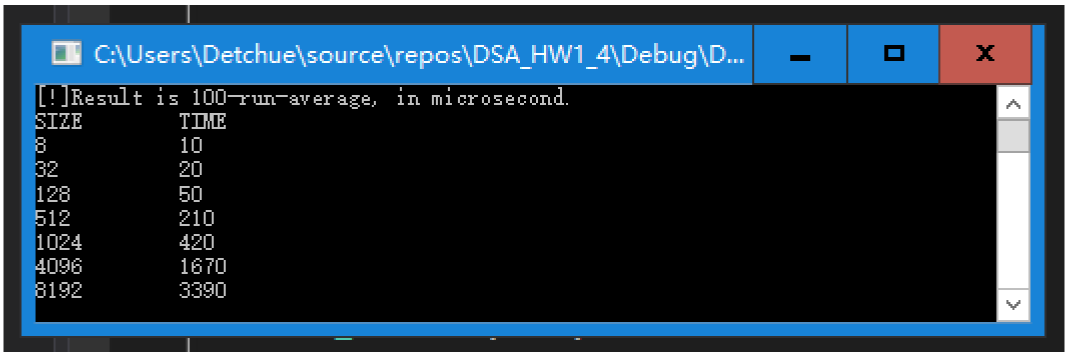
The main method is just try to find the biggest and smallest element.(cpp attached)

Result Test:

Now we run the farpair code just to make sure is really is a linear algorithm. Test data is what we have used in Q1, "8int", "32int", etc. (#4192 not included, I don't like that number)

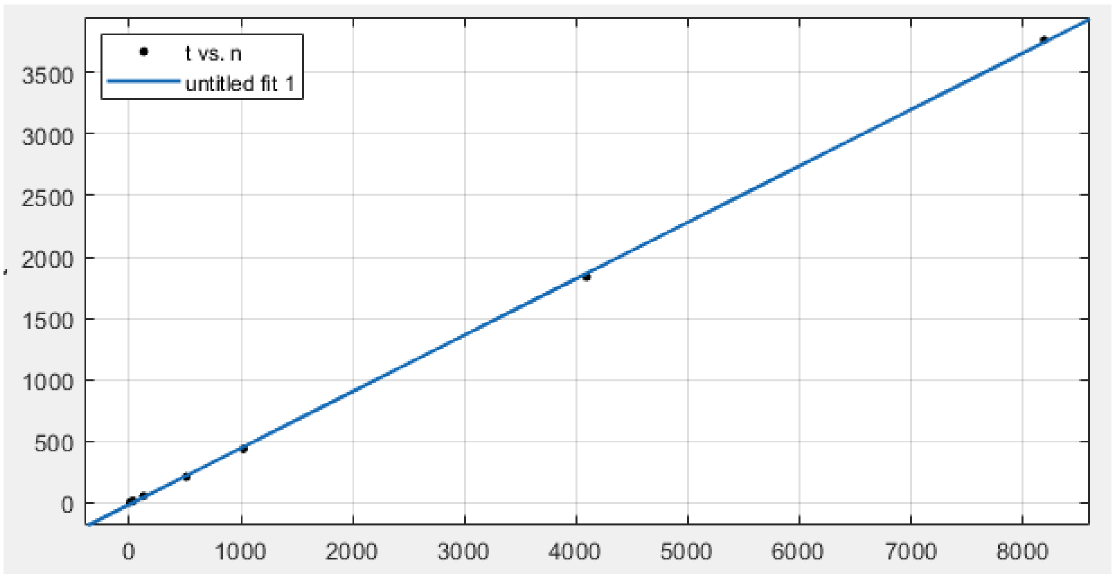
Time shown in "μs"

	8	**32**	**128**	**512**	**1024**	**4096**	**8192**
Farpair	10	20	50	210	420	1670	3390



```
[!]Result is 100-run-average, in microsecond.
SIZE        TIME
8           10
32          20
128         50
512         210
1024        420
4096        1670
8192        3390
```

T(farpair)=0.4593N-15.19



#Estimated and plotted based on MatLab2017b Curve Fitting Tool and Plot Tool.

Q5. Faster-est-ist 3-sum: Develop an implementation that uses a linear algorithm to count the number of pairs that sum to zero after the array is sorted (instead of the binary-search based linearithmic algorithm). Use the ideas to develop a quadratic algorithm for the 3-sum problem.

A:

Linear Algorithm at O(n) for 2-sum in sorted array (Also, cpp file is attached)

The method is to start scanning from both sides, we add number pairs from bigger and small size and adjust location based on whether the result is greater than 0 or smaller than 0.

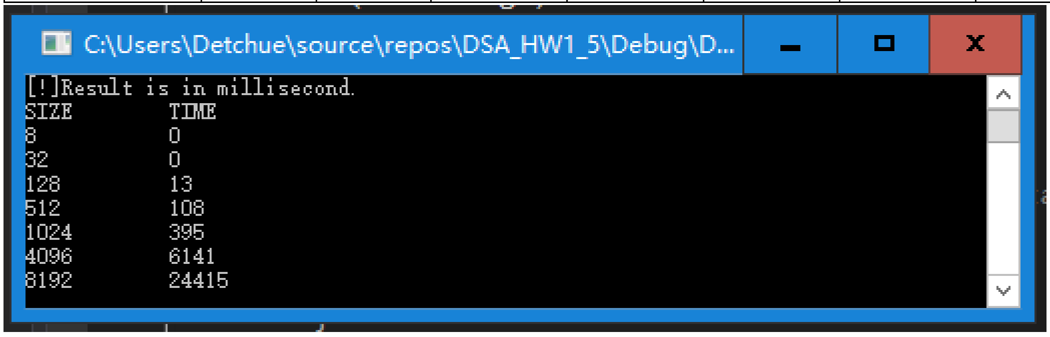
Quadratic Algorithm for 3-sum (Also, cpp file is attached)

The same method, just to put the algorithm above into a scanning loop, which makes complexity N time greater.

Result Test for 3-sum:

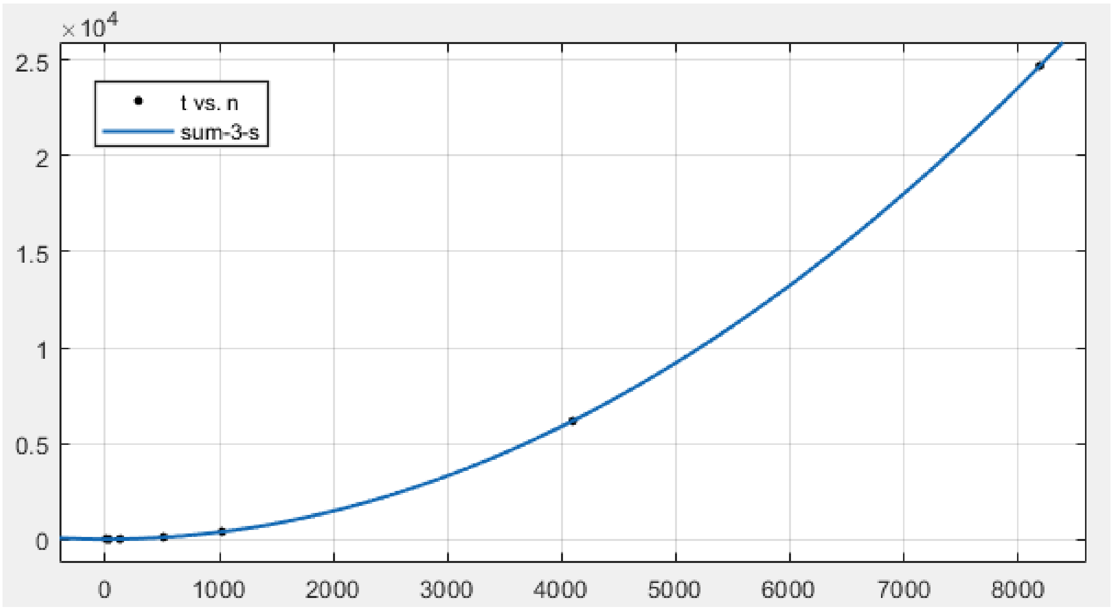
Now we run the 3-sum code just to make sure is really is a quadratic algorithm. Test data is what we have used in Q1, "8int", "32int", etc. (#4192 not included)

Time in "ms"	**8**	**32**	**128**	**512**	**1024**	**4096**	**8192**
3-sum	0	0	13	108	395	6141	24415



```
C:\Users\Detchue\source\repos\DSA_HW1_5\Debug\D...
[!]Result is in millisecond.
SIZE        TIME
8           0
32          0
128         13
512         108
1024        395
4096        6141
8192        24415
```

T(sorted 3-sum)=0.0003686N^2-0.004066N+2.622



X: N(data size); Y: T(time consumption)

#Estimated & plotted based on MatLab2017b Curve Fitting Tool and Plot Tool.