

Linnaeus University

1DV516 – Algorithms and data structures Assignment 1

Student: Fredric Eriksson Sepúlveda
Student ID: fe222pa@student.lnu.se



Table of contents

Contents

Setup 3

Problem 4 4

Problem 7 9

Problem 8 15

Setup

I created a java class called GTM (general Time manager), and its rather flexible it allows for 2 types of testing by providing a size, max size and a sizeIntervalIncrease, and iterations the other values are for convenience ie having lines, or wanting to set up the first line of the CSV file.

You can see them in action in the main class.

I utilize strategy patterns because I read that callable classes might affect performance and since setting up strategy patterns makes the program easier to manipulate for future assignments

The key to this class is located in the Measurers folder where I create the measurers for future experiments in general in there I create the class and set different parameters, but the key is that they release an ArrayList of the values wanted.

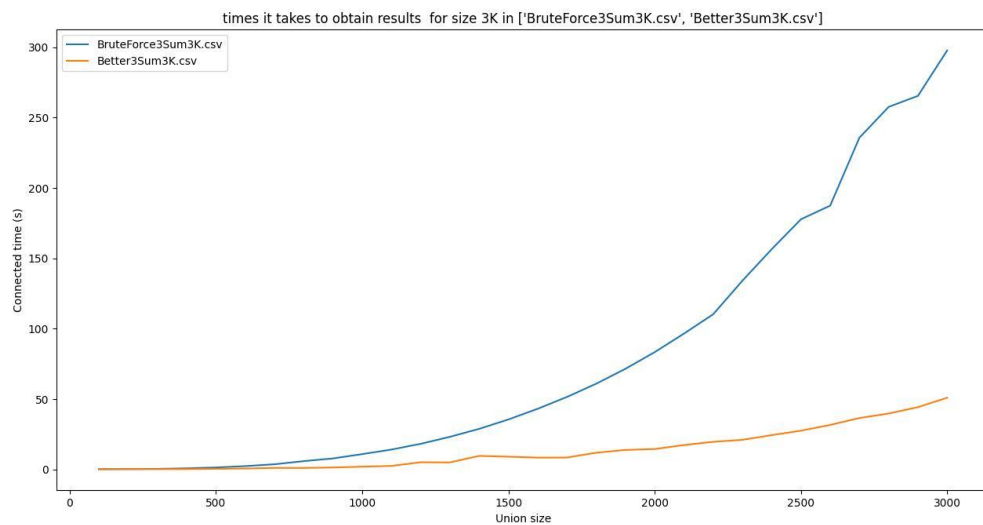
Problem 4

Aim: run 3Sum for different array sizes and measure the time it takes, then Empirically determine the growth.

Questions: Does it match your expectations? why/ why not?

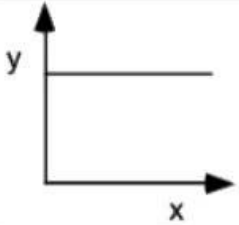
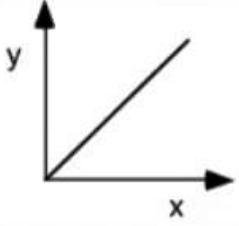
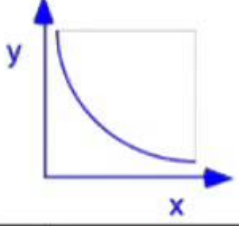
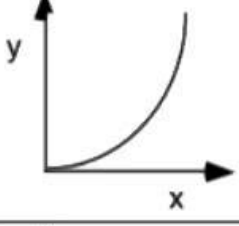
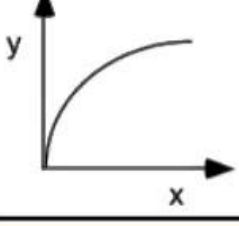
I will present how I obtain the growth by trying to linearize the table and obtain the gradient in that way

This is how the values look like in a graph for a max value of 3000



Obtaining growth for BruteForce or the naïve version of 3sum

We start by linearizing it, to do that we take all values of y and turn them into $(\log_2(y))$, the reason is this: if we assume that the above graph has one of the following shapes:

| Graph shape | Written relationship | Modification required to linearize graph | Algebraic representation |
|---|--|---|-------------------------------------|
|  | As x increases, y remains the same. There is no relationship between the variables. | None | $y = b$, or y is constant |
|  | As x increases, y increases proportionally. Y is directly proportional to x . | None | $y = mx + b$ |
|  | As x increases, y decreases. Y is inversely proportional to x . | Graph y vs $\frac{1}{x}$, or y vs x^{-1} | $y = m\left(\frac{1}{x}\right) + b$ |
|  | Y is proportional to the square of x . | Graph y vs x^2 | $y = mx^2 + b$ |
|  | The square of y is proportional to x . | Graph y^2 vs x | $y^2 = mx + b$ |

Source:” <https://sites.google.com/site/apphysics1online/appendices/2-data-analysis/graph-linearization>”

We can make assumptions about the graph itself, in this case, we have the structure of $y = mx^2 + b$

If we assume we do not know its x^2

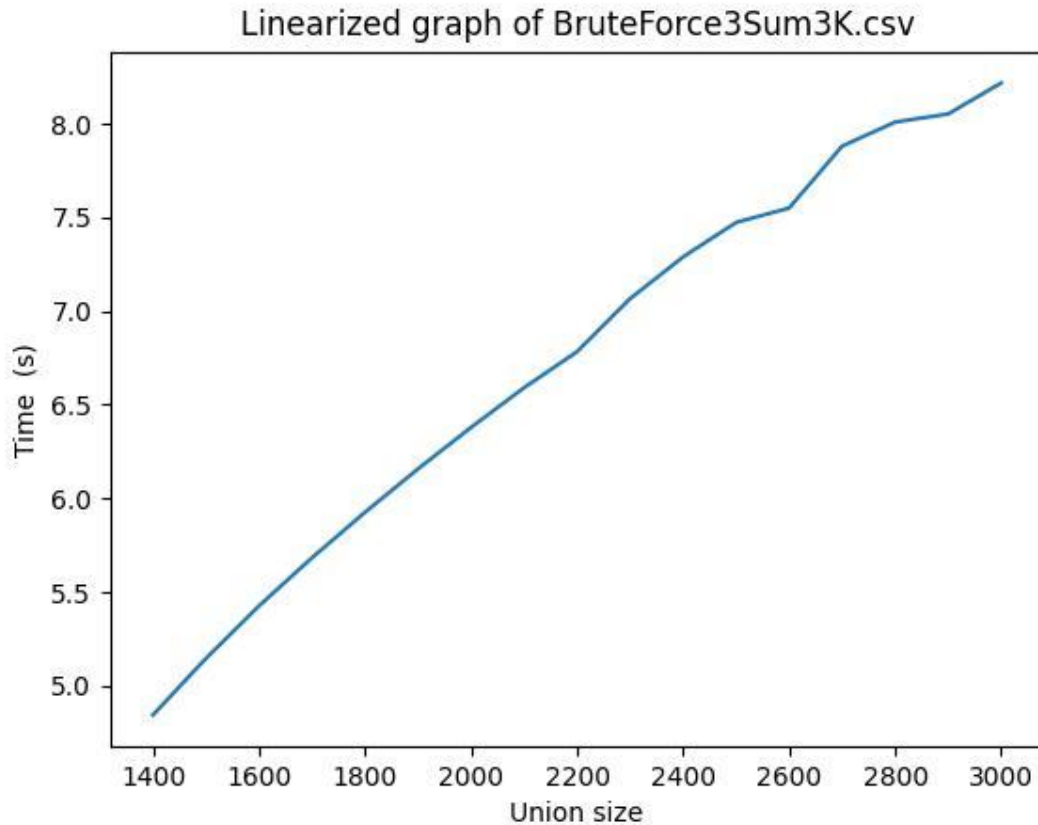
We have : $y = mx^n + b$

We assume $b = 0$

And so its $y = mx^n$, or for easier readability $y = a \cdot x^{**}b$

$$\text{Log}_2(y) = b \cdot \text{log}_2(x) + a$$

This resembles $y = mx + b$ and so it can be said it is linearized



And so it is

The next step is to calculate the slope by taking two values

From the csv file = BruteForce3Sum3K.csv

$$\text{Log}_2(257.51) = b \cdot (\text{log}_2(2240)) + c$$

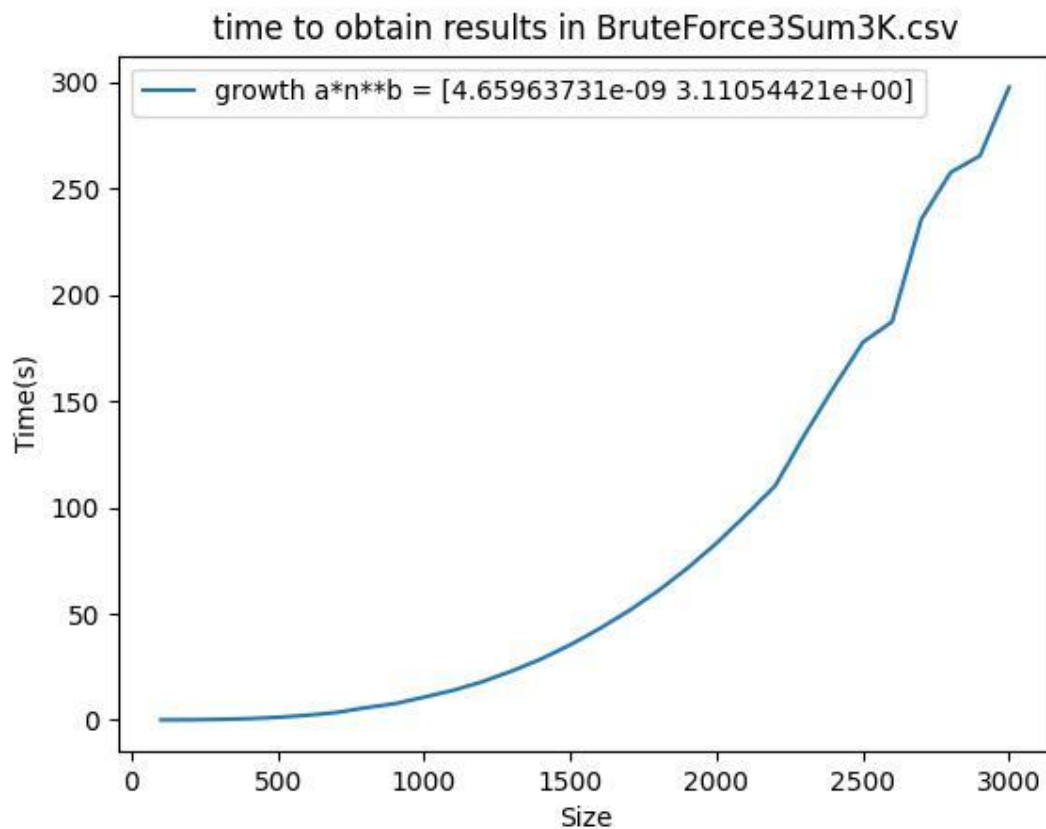
$$\text{Log}_2(110.13) = b \cdot (\text{log}_2(1760)) + c$$

$$B = \text{log}_2(257.51) - \text{log}_2(110.13) / \text{log}_2(2240) - \text{log}_2(1760)$$

$$B = 8 - 6.78 / 11.13 - 10.78 = 3.4857142857142857142857142857143 \text{ around } 3.5$$

$$c = \text{log}_2(257.51) - 3.5(\text{log}_2(2240)) = 8 - 38.955 = -30.96$$

for the next graphs in this section, I will use curve-fit since I demonstrated I know how to manually obtain the growth will use



Growth of the graph shown in the curve fit

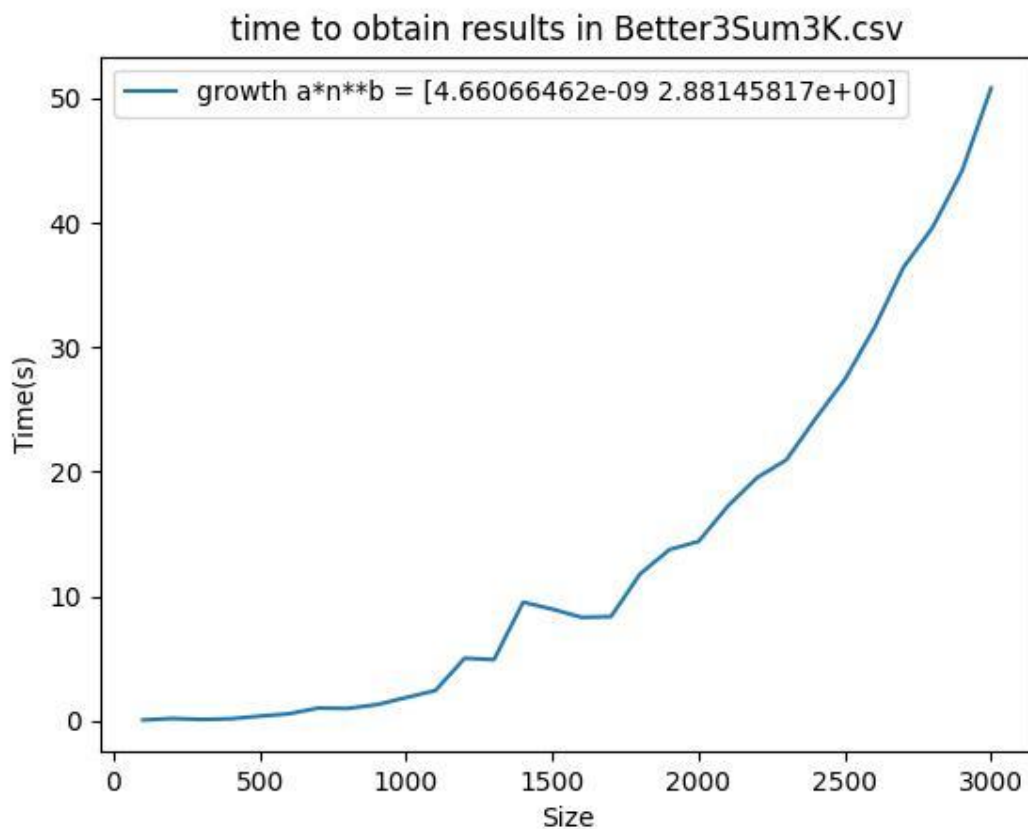
Better3sum

The difference is that I avoid iterating extra values by assuming that we already went through them before

```
public List<String[]> ThreeSum() {
    List<String[]> results = new ArrayList<String[]>();
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            for (int k = j + 1; k < size; k++) {
```

As can be seen here the next for loop is dependent on the previous index + 1

There is no much interest in obtaining growth for this 3sum so I will just provide the curve fit one



The difference between both graphs is the growth per size however both have the same growth structure

Answers:

Naive3sum did follow my expectations because the sum method contains 3 for loops inside and because of that I expected around cubic time complexity (it's more complicated than that but iterating through the list while iterating through does gives the idea that its cubic)

Better3sum did follow my expectations because it also contains 3 for loops and in addition, I expected it to be faster. Because by assuming that iterating through once, you would end up going through the same values so taking them out would provide us faster iterations

Problem 7

objective

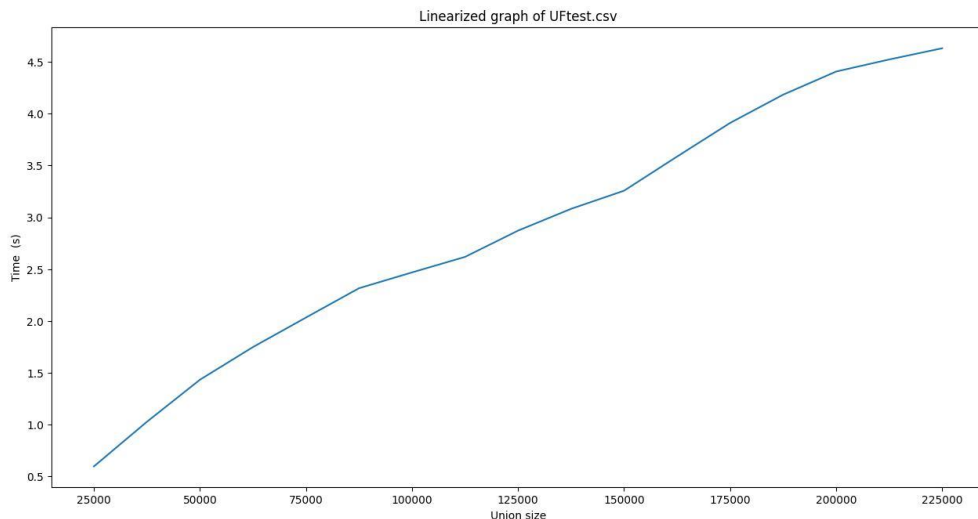
Run your two implementations of UnionFind for various array sizes and unions. Measure the time it takes and uses these to empirically determine how they grow.

Questions Does it match your expectations?
Why/why not?

How the values look in a graph for different arrays and union sizes

Union Find (UF)

Same procedure as in 3Sum, we linearize the graph size 250000



While this graph might not look like the most linear version of it, it still does have some usefulness

We just have to assume that this might not be the most accurate value for growth.

$$\log_2(23.01) = b \cdot \log_2(212500) + c$$

$$\log_2(8.47) = b \cdot \log_2(137500) + c$$

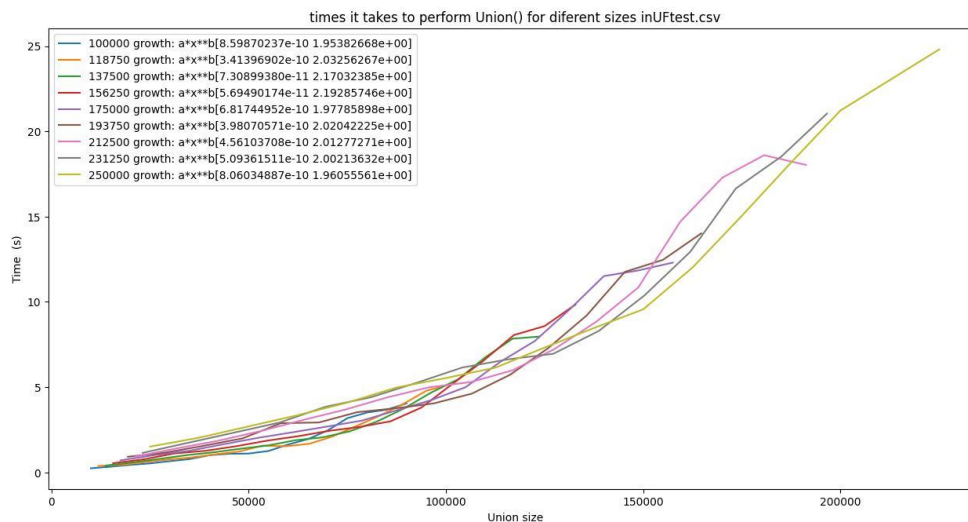
$$B = \log_2(23.01) - \log_2(8.47) / \log_2(212500) - \log_2(137500)$$

$$B = 4.52 - 3.08 / 17.70 - 17.1 = 1.44 / 0.6 = 2.4$$

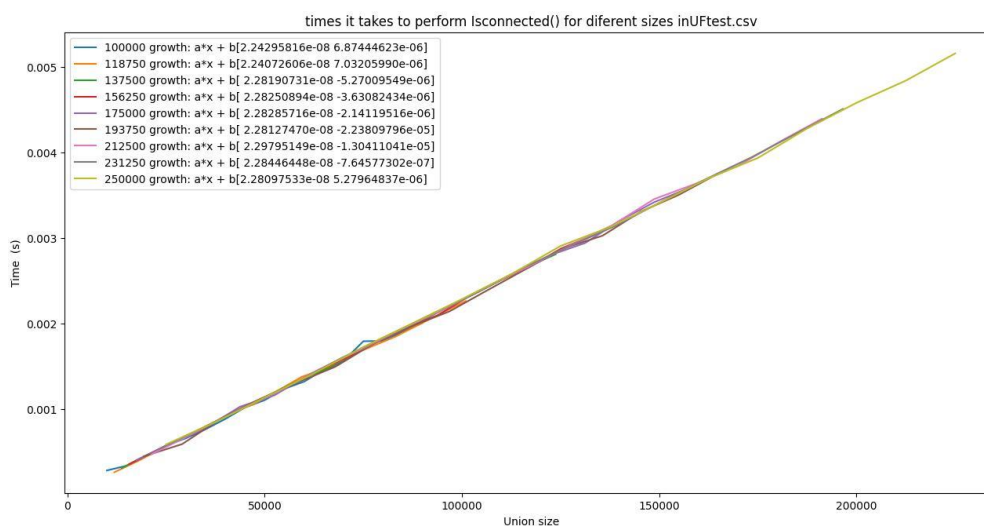
$$C = \log_2(43.23) - 2.4(\log_2(212500)) = 5.43 - 2.4(17.70) = -37.05$$

$$Y = (2^{-37.05})(x^{2.4})$$

Roughly quadratic estimation



The values for n vary between 1.96 to 2.1 according to curve fit values
 Curve fit values are not too far from the estimated one, so it would be safe to say that UF has a time complexity of $O(n^2)$

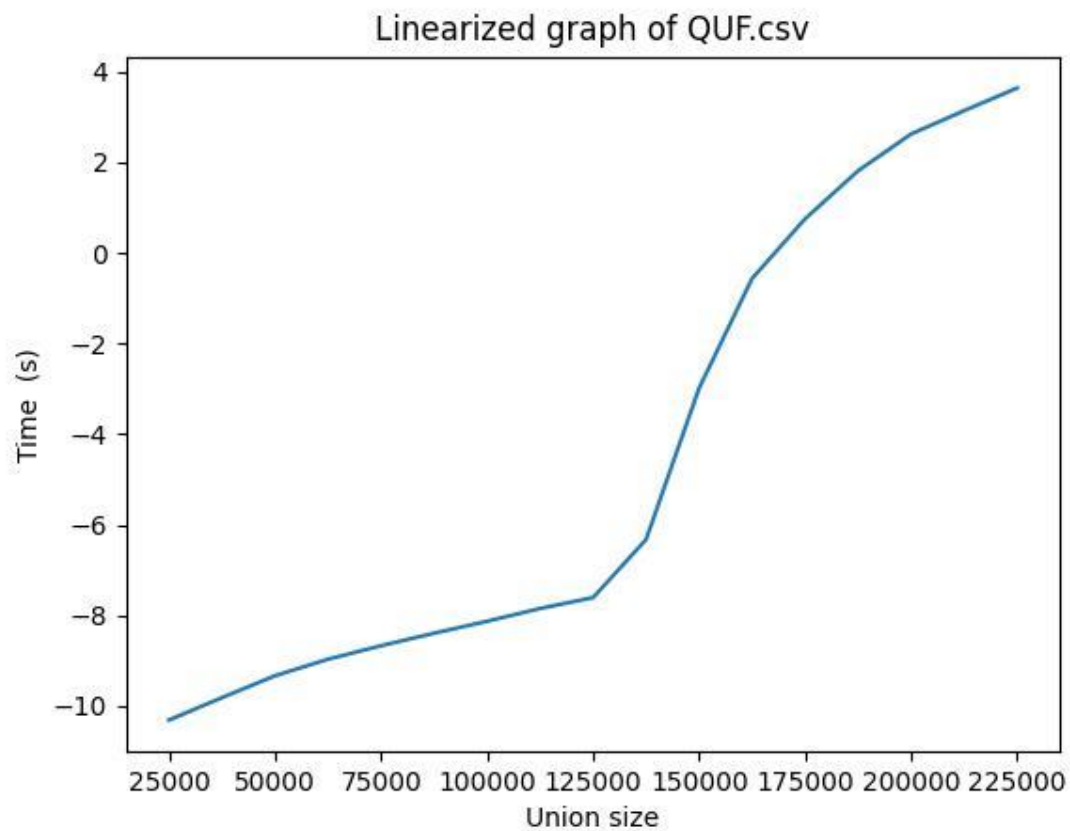


Almost constant time and increases because there are more union sizes

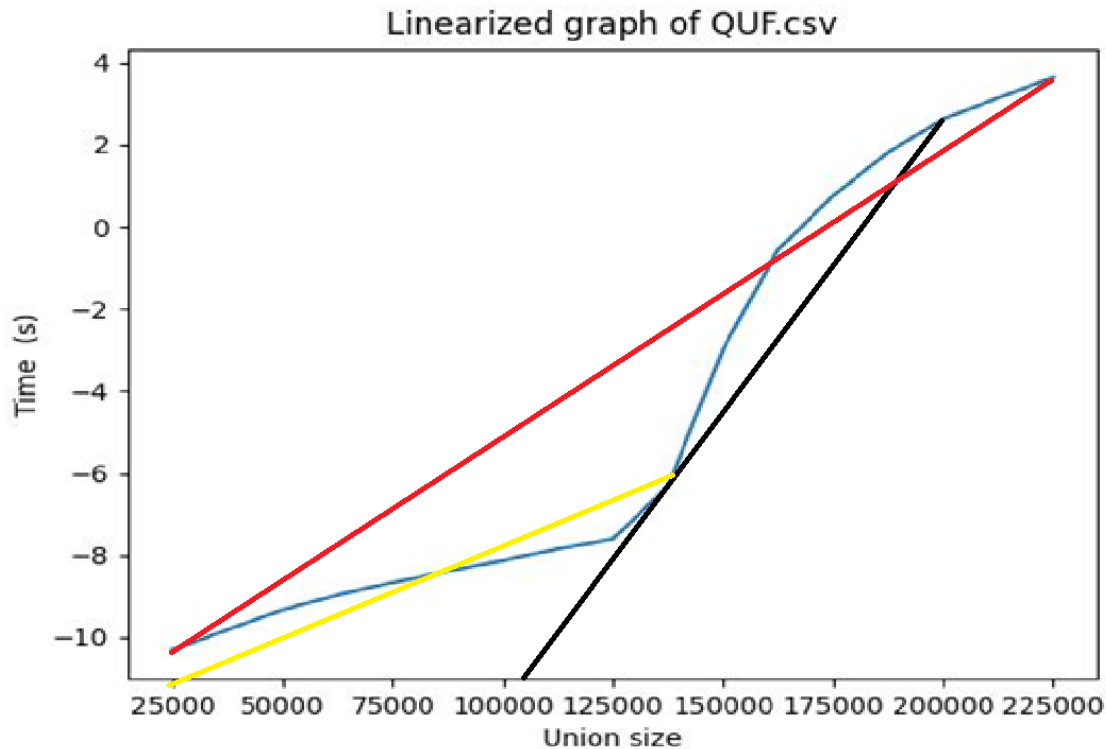
QuickUnionFind (QUF)

For QUF is more complicated and to clarify, I will go through the same process to obtain growth as the previous Union find

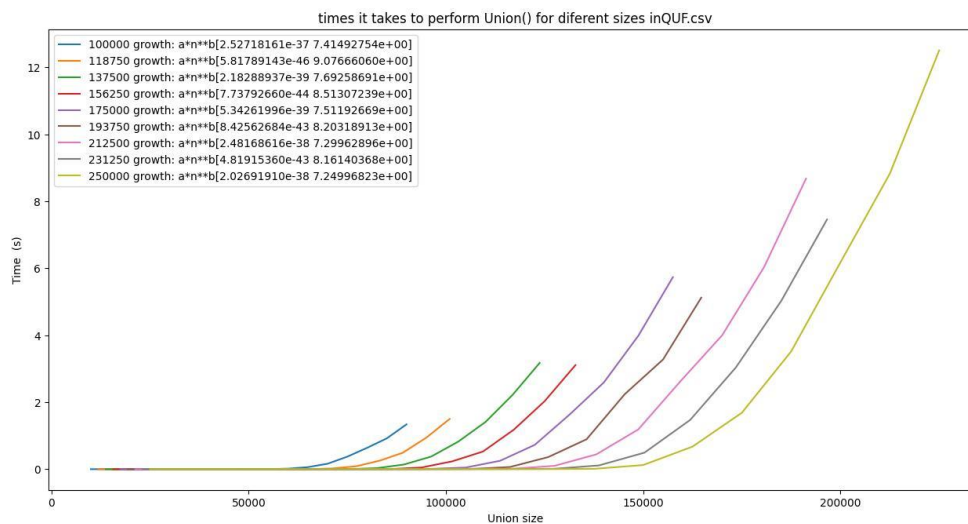
Linearization



Here is where we are getting some troubles with the linearization approach, this does not work for a linear graph, and picking values here would provide very different results based on what I pick
IE



So the linearization approach just does not work here because I will end up avoiding many points, and to prove this point further, running curve fit provides very ludicrous values



If we analyze the algorithm, we know, that this algorithm's running times are very dependent on the depth of the tree.

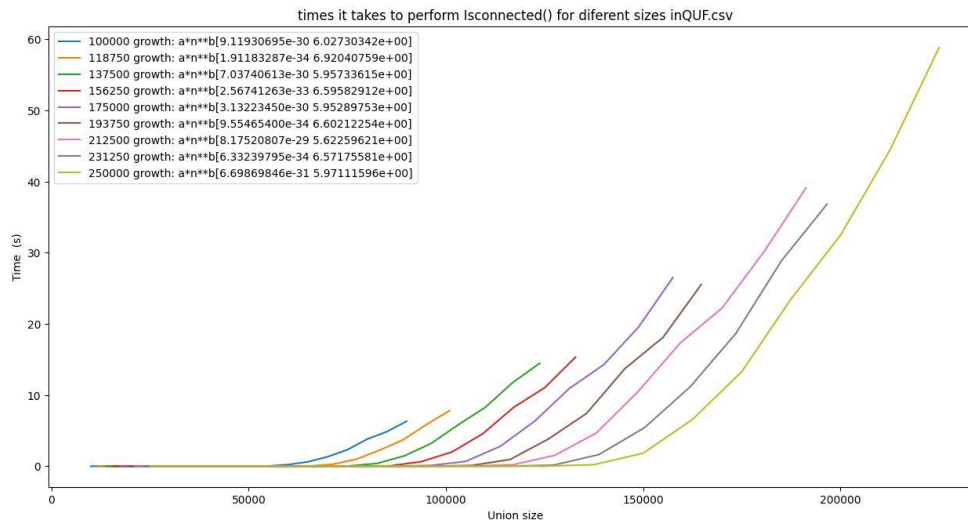
Theoretically, the range of running times varies between $[\theta(n), \theta(n \cdot \log(n))]$, and $\theta(n \cdot \text{depth})$ and this is because of how the tree structure works.

n : because it has to iterate for n and the union() method gives constant.

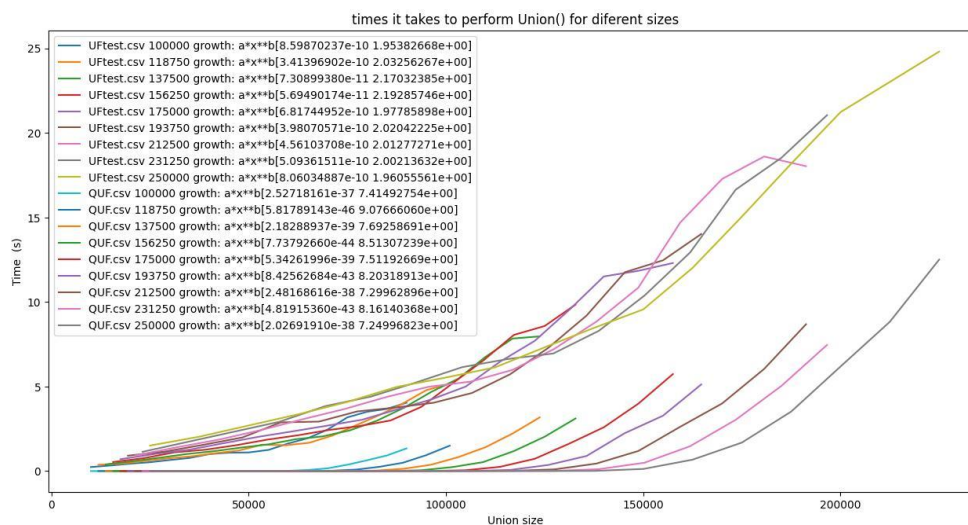
$n \cdot \log(n)$: iterate n and union() method takes $\theta(\log(n))$.

$N \times \text{depth}$: which at worst can be n^2 , iterate n and `union()` takes $\theta(d \times n)$

If we see the `isConnected` table we see the table having the same property of being dependent on the depth of a tree



Both Union in the same graph



Questions:

UF served as a good way of learning time complexities because by just looking at the union method there is just one for loop and because of that I expected that in paper it should be linear, however, it turned out to be quadratic and the reason for that is that the method union itself takes $O(n)$ but invoking Union n times makes it quadratic.

UF is connected and provides almost constant values I think it follows my expectations since we are just taking two values where we know their location in the list

QUF in a way did follow my expectations because I did not believe I would get a static growth that I can pinpoint. Because it's very dependent on the depth in the tree and this happens for both `union()` and `isConnected()` methods.

Problem 8

Problem 8, the report should also contain your experiment setup and findings from that problem.

In monteCarloSim.java I illustrate much of my thought process, however, in short words, I thought through all possible components and made a difference between it being a normal connect or a side connect.

Of Course, I am using quickUnionFind to get the answer because we are interested in the connected component at the start being connected to the component at the end.

Findings

```
less specific but overall oriented test  
average overall: 0.590485108231829 and average of stdev: 0.01504234454562744  
hyper specific test  
average of n = 10: 0.5897170282971703  
standard deviation of n = 10 is: 0.07442250949147106
```

I made two types of tests, overall and hyper-specific

The aim overall is to make fewer iterations per size but have a broader amount of sizes
While Hyper specifics aim is to take only 1 size (10 in this case) and run it for very high values and see if there are any differences there.

As can be seen, both methods provided me with very similar values, which means that unless the randomness (ie how Thread.localrandom works) affects my methodology, 0.59 is the answer to the percolation