MATHEMATICS
EXTENDED ESSAY

Efficiency comparison between the Binary algorithm and Merge Algorithm in COVID Pool Testing

Word Count: 3890

May 2021

Contents

1 Introduction

1.1 Pool Testing Introduction

Currently, COVID-19 has spread all over the world. The only way to stop this spreading is to quarantine the people with positive testing results, which can interrupt the transmission of the virus. However, most of the country can't conduct such a great amount of individual tests every day.

Therefore, to solve this problem, many countries start to adopt pool testing to accelerate testing's efficiency. For example, Andaman and Nicobar have been the first place in India to implement 'pool testing' to overcome the shortage of test kits in the country (BusinessInsider, 2020).

Pool testing means to mix several people's saliva together into one test kit. If the result is negative, this means none of people in the mixing sample is affected. When the result is positive, it means someone in the group has been affected. We need to test each one in the group to find out the people been affected. This approach increases the number of individuals that can be tested using the same amount of resources (FDA, 2020).

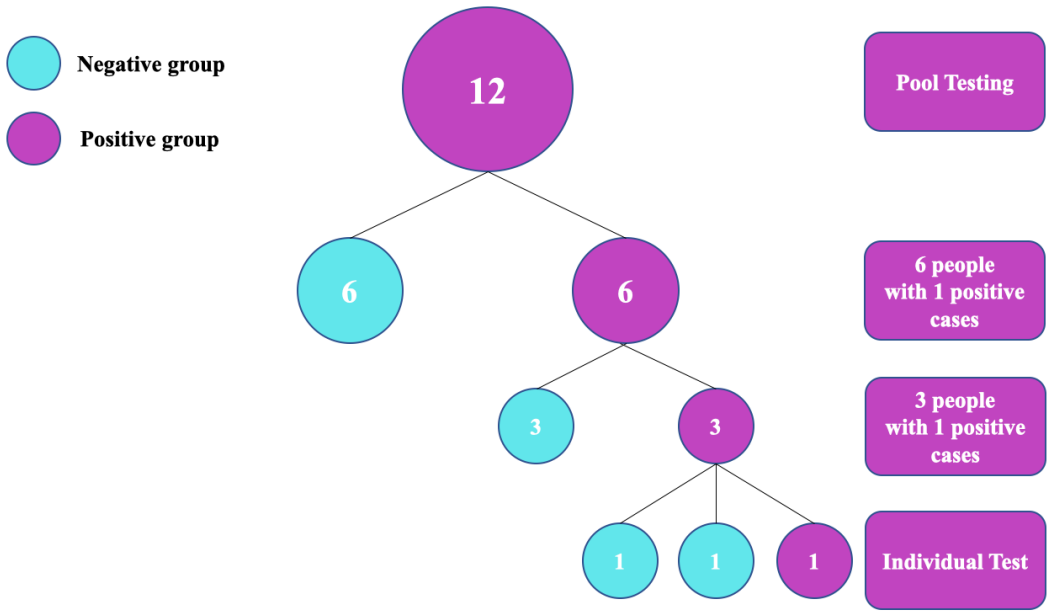

Figure 1: Pool Testing Example

From the example above, 12 people's saliva are mixed together. There is 1 positive case among 12 people. If we adopt the individual testing method, totally it requires 12 times of testing. However, pool

testing only requires 7 times of testing.

1.2 Target of the Model

For pool testing, several things need to be determined. At first, as shown in Figure 1, we need to determine the group number going to be divided, g, in each stages. Besides, the number of stages is also necessary, which k will also influence the number of testing.

This paper establishes two Pool testing models based on Binary Algorithm and Merge Algorithm. Both of them relies on different g and k. We will compare the efficiency of each model in different scenarios. The target of this paper is to establish and find a time-consuming and cost-efficient pool testing model.

1.3 Data Source

To evaluate the efficiency of two models in different scenarios. The paper chooses the positivity from the United States and India as data for the calculation.

The data of the USA is publicized by the Centers for Disease Control and Prevention (CDC), and the data of India is publicized by COVID19INDA.org.

1.4 Notation and Glossary

Table 1: Notation List

Variable	**Symbol**
Groups going to be devided in each stage	g
Total stages for the Pool Testing	k
Total Number of Testing	T
Number of the positive cases	p
Number of the testing cases	n
Expectation of the number of the positive groups	E
False Negative Testing Control Rate	CR

2 Pool Testing Model Based On Binary Algorithm

2.1 Introduction

Binary search, a method that divided the whole group into two subgroups during computational searching, is widely used in searching a specific sample among a huge entity. As shown in Figure 2, we have 8 people in total with 1 positive case. Dividing it into two subgroups, we will receive one positive and one negative subgroup, which shrinks the positive cases into a smaller scale for searching. Repeating this process, we will find out that positive case through 6 testing times.

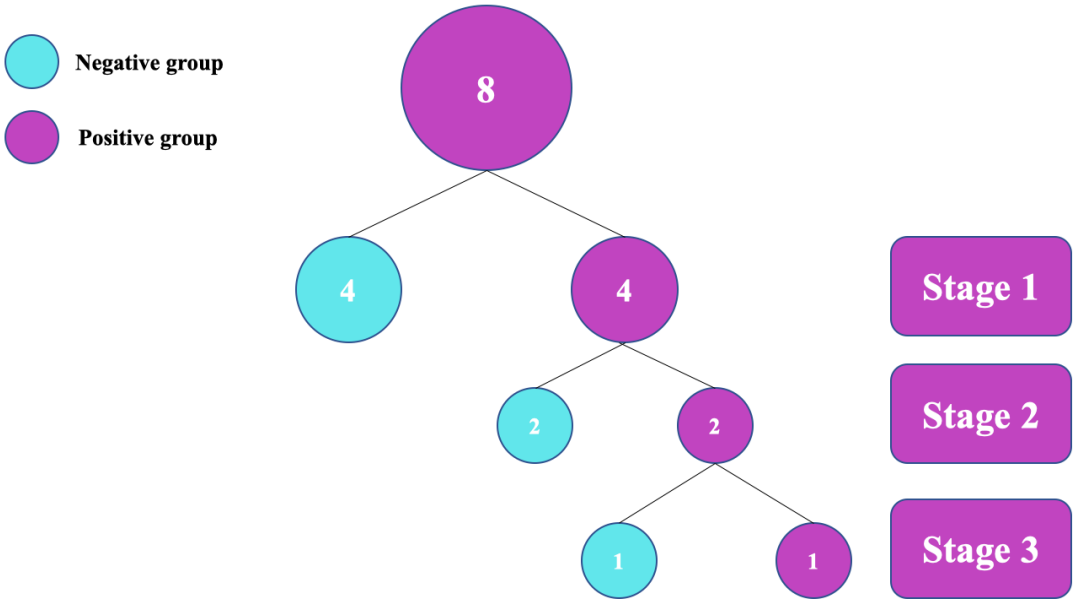

Figure 2: Binary Testing Example

2.2 Total Number of Testing Model Establishing

I will consider the worst situation in the following analysis, which all positive samples will be separated evenly into different groups. This will maximize the time of testing, because a negative group does not need to be tested again, but a positive group does.

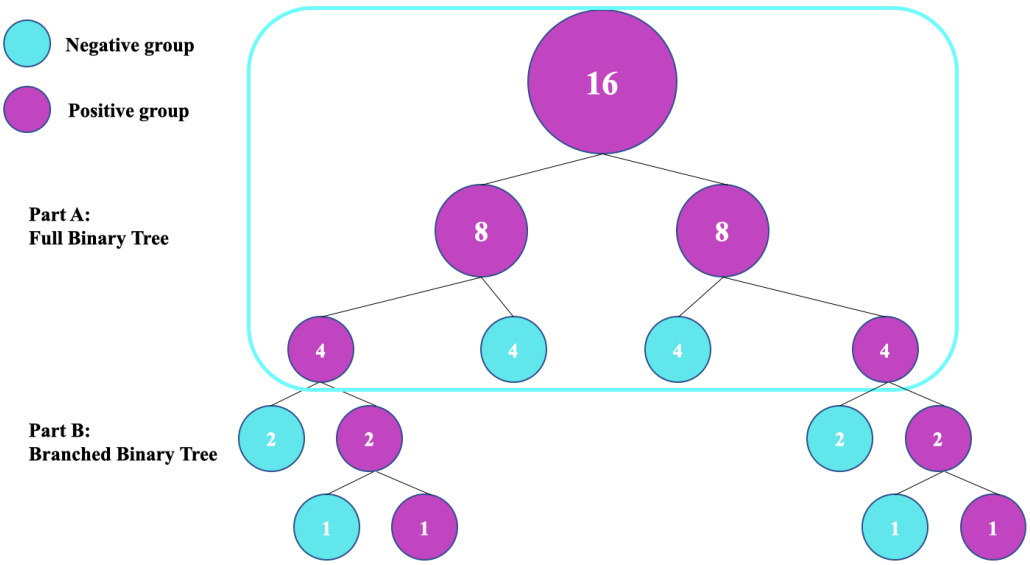

Figure 3: Binary Testing Improved Model

As shown in the figure 3, supposing that there are going to be 2 positive samples among the entity of 16 people. First, they will be divided into two subgroups, each with 8 people. These 8 people's saliva are been collected together to a test kit, if one person has the saliva with the positive results, then the whole group will be reported as "Positive group", colored as purple. Therefore, as we assume that the positive samples will be evenly separated into the subgroup, each of the group with 8 people will be reported as positive group.

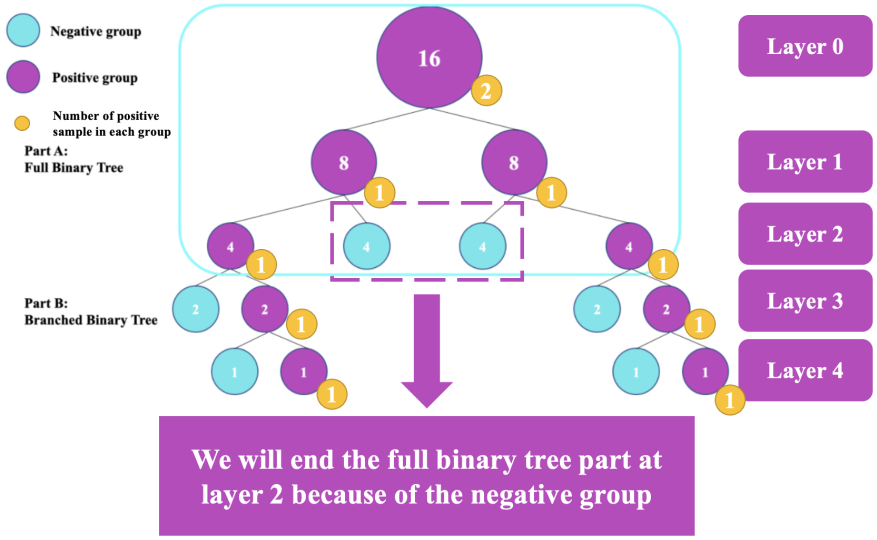

Figure 4: Full binary tree ended illustration

Then, we divide it into a smaller group with 4 people. Since there is only 1 positive sample in the group of 8 people, it must produce a group of 4 which all the people are with negative results. A negative group does not need to be tested again, so it will not produce any subgroups. Thus, we named the stage when first produced the negative group as the "full binary tree", which each branch is fulfilled with two nodes, as shown in the Figure 4.

To calculate the total time of testing, we need to find the depth of the full binary tree. As mentioned above, the depth of full binary tree is decided by the number of the positive samples. In the x layer of the tree, it will have 2^x number of groups. For example, in the layer 0, we have a big group with 16 people; in the layer 1, we have 2 groups with 8 people each, which is 2^1; in the layer 2, we have 4 groups with 4 people, which is 2^2. However, for example in figure 3, the number of the positive sample (2) is smaller than the number of groups in layer 2 (4), which means it must produce 2 negative groups. Therefore, our full binary tree will end because negative groups will not produce any more subgroups.

Hence, we can calculate the number of the full binary tree layer by using $\log_2 p$ and use the floor to ensure that the result is an integer. However, taking the example from above, when number of the positive case equals to 2^n, we still need an extra layer of testing, because $\log_2 2$ tells us that it only requires 1 layer, but actually all the nodes in layer 1 are positive groups, so it can form another layer again. If p does not equal to 2^n, because of the floor, it will loss the last layer as well. For example, when p equals to 3, it is supposed to have 2 layers, but floor of the log will only show the result of 1.

Therefore, we need to add an extra layer to the floor, which is $\lfloor \log_2 p \rfloor + 1$, p stands for the number of positive cases.

Still, we need to measure the depth of the branched binary tree. The depth for the entire binary tree is $\log_2 n$. However, we need to use the ceiling here. For example, shown in figure 3, if we have 17 people instead of 16 people, which is not 2^n, in the layer 4, there must exist a group with 2 people, instead every group is only consisted of 1 people. Therefore, we need to have an extra layer of testing for the group with 2 people, which increases the number of layer to 5, and $\lceil \log_2 n \rceil$ will solve this problem entirely.

Thus, using the depth of entire tree minuses the depth of the full binary tree, we will receive the depth of the branched binary tree, which is $(\lceil \log_2 n \rceil - \lfloor \log_2 p \rfloor - 1)$.

After getting the depth of each part of the tree, we need to count number of groups, or number of testing. For the full binary tree part, we need to count from 2^0 to the $2^{\lfloor \log_2 p \rfloor + 1}$, which is the number of

group in the first stage and last stage of the full binary tree. So, the number of testing in the full binary tree is $\sum_{i=0}^{\lfloor \log_2 p \rfloor + 1} 2^i$.

For the branched binary tree, as shown in figure 3, one separation of a larger group will lead to a positive group and a negative group, so using the depth of the branched part times $2p$ ("2" means every time of separation in branched part will produce 2 subgroups, one positive and one negative), which is

$$2p \cdot (\lceil \log_2 n \rceil - \lfloor \log_2 p \rfloor - 1) \tag{1}$$

Therefore, adding those two parts together we will get the T:

$$T(n, p) = 2p \cdot (\lceil \log_2 n \rceil - \lfloor \log_2 p \rfloor - 1) + \sum_{i=0}^{\lfloor \log_2 p \rfloor + 1} 2^i \tag{2}$$

To verify expression (2), we can use the value in Figure 3 to verify the results:

$$T(16, 2) = 2 \cdot 2 \cdot (\lceil \log_2 16 \rceil - \lfloor \log_2 2 \rfloor - 1) + \sum_{i=0}^{\lfloor \log_2 2 \rfloor + 1} 2^i \tag{3}$$

$$T(16, 2) = 4 \cdot (4 - 1 - 1) + 1 + 2 + 4 = 15 \tag{4}$$

The result from expression 4 is accordant to the time of testing in Figure 3, which verifies the reliability of the model.

2.3 Data Calculation

The data from CDC shows that the positive test rate in the USA is %3.6 (CovidUSA, 2021). Therefore, considering that $10,000$ people are going to attend the pool testing, there might be 360 positive sample. Through coding (Appendix-1), the results are shown below:

$$T(10000, 360) = 720(\lceil \log_2 10000 \rceil - \lfloor \log_2 360 \rfloor - 1) + \sum_{i=0}^{\lfloor \log_2 360 \rfloor + 1} 2^i = 4623 \tag{5}$$

Thus, it only takes 4,623 times to find out all the positive cases from the 10,000 people.

3 Pool Testing Model Based on Merge Algorithm

3.1 Introduction

In the binary algorithm, the number of the group divided in each stage is determined, which every division will produce 2 subgroups. Defining g as the number of group been divided in each stage. We can alter g to improve the efficiency of the testing. So, in this section, we will find out how g is going to influence the testing times.

3.2 Two-Stage Total Number of Testing Model Establishing

At first, we can focus on the situation that the stage number is 2. As shown in Figure 5, the first stage is going to divide n people into g groups, and each group will contain $\frac{n}{g}$ people. Then, in the second stage, all the people in the positive group need to be tested individually. Therefore, our all goal is to minimize the number of testing in these two stages.

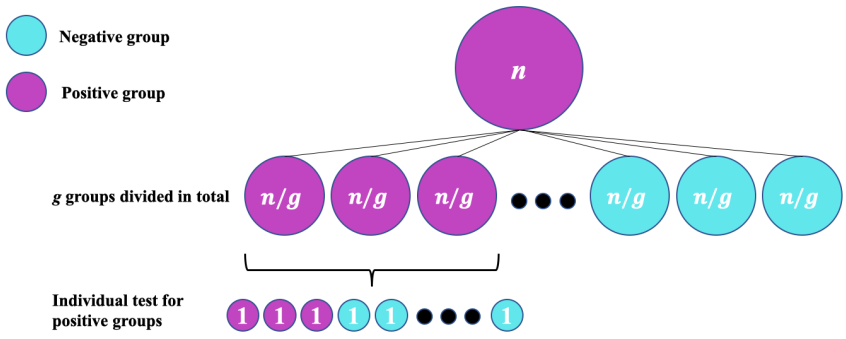

Figure 5: Binary Testing Improved Model

Through the first stage, we need g number of testing, and people affected by the COVID will distribute into subgroups during stage 1. We want to determine how many groups in the stage 1 will have positive results, so we need to calculate the expectation of the positive group's number.

Thus, we need to determine how many positive groups will appear. Specifically, we need to find the expectation of positive group's number (E) in stage 1, and then multiply this expectation with the number of the people in each positive group, which is $\frac{n}{g}$, the product of them is the number of the individual tests need to be conducted. Hence, adding the number of group testing in stage 1 with the

number of individual testing, the total number of testing T can be expressed through the following expression:

$$T(n, p, g) = g + E(n, p, g) \cdot \frac{n}{g} \tag{6}$$

For E, it is correlated to the number of people affected by COVID (p), total number of people get into the testing (n), and number of group that is going to be divided in stage 1 (g).

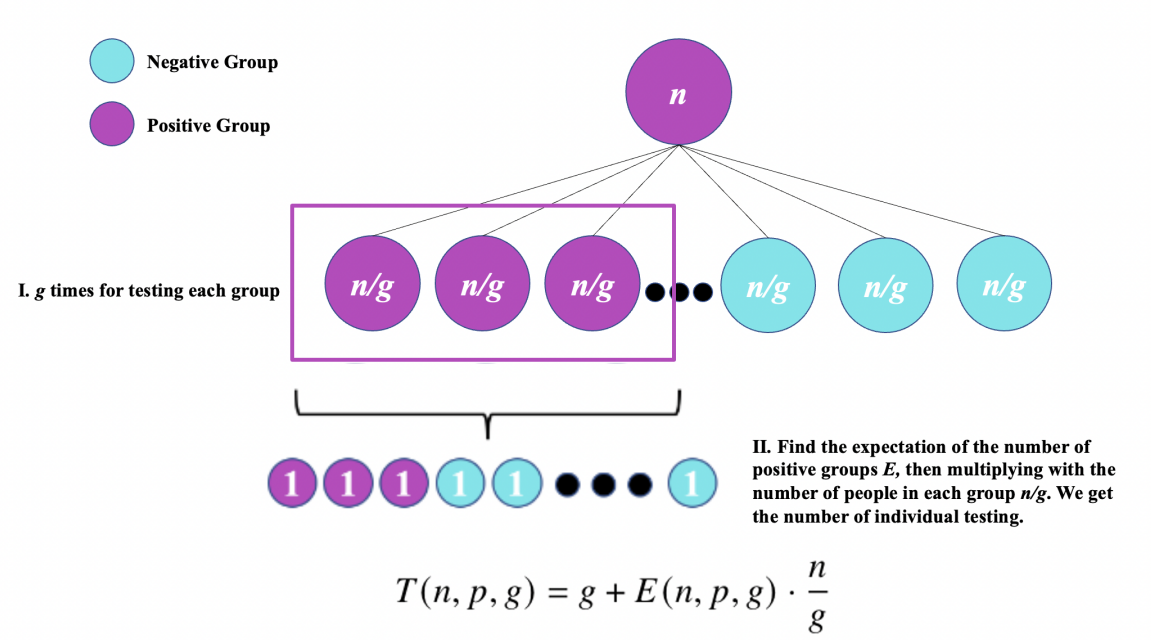

Figure 6: Illustration for $T(n, p, g)$

We define probability function $P(i)$ as the probability for i number of positive groups to be appeared in stage 1. For example, $P(10)$ stands for probability that there are 10 positive groups among g groups in the first stage. So we can express the expectation as the following expression:

$$E(n, p, g) = \sum_{i=1}^{n} P(i) \cdot i \tag{7}$$

For P, we can use the binomial distribution to find it. The probability that randomly selecting a people with negative result from n is $\frac{n-p}{n}$. For a negative group to appear, every one in the group ($\frac{n}{g}$ people in a group) must have negative result, which is $(\frac{n-p}{n})^{\frac{n}{g}}$. Moreover, we can also induce the probability for a positive group to appear as $[1 - [\frac{n-p}{n}]^{\frac{n}{g}}]$. In the binomial, we want to find the probability for i positive groups to appear, which is

$$P(i) = \binom{g}{i} \cdot [[\frac{n-p}{n}]^{\frac{n}{g}}]^{g-i} \cdot [1 - [\frac{n-p}{n}]^{\frac{n}{g}}]^{i} \tag{8}$$

Hence, E is the accumulative expectation for i from 1 to n, which can be expressed through the following expression:

$$E(n,p,g) = \sum_{i=1}^{n} \binom{g}{i} \cdot [[\frac{n-p}{n}]^{\frac{n}{g}}]^{g-i} \cdot [1 - [\frac{n-p}{n}]^{\frac{n}{g}}]^{i} \cdot i \tag{9}$$

Therefore, T for two-stage is

$$T(n,p,g) = g + \frac{n \cdot \sum_{i=1}^{n} \binom{g}{i} \cdot [[\frac{n-p}{n}]^{\frac{n}{g}}]^{g-i} \cdot [1 - [\frac{n-p}{n}]^{\frac{n}{g}}]^{i} \cdot i}{g} \tag{10}$$

However, E is so complex that it's hard to utilize it to find the solution through coding. So, using the data that $n = 10000$ and $p = 360$, by simulating g from 1 to 1000, we can regress it with a proper regression function. Figure 7 shows the result $E(10000, 360, g)$ (g ranges from 1 to 1,000):

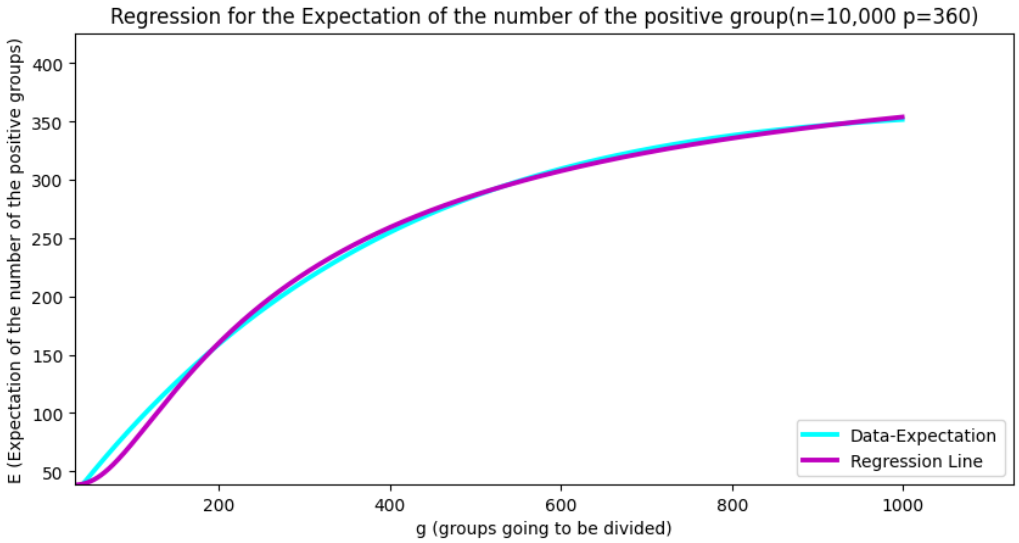

Figure 7: Regression for E

Blue continuous line represents real expectation and the purple line is the regression line $f(g)$:

$$f(g) = p \cdot (e^{-\frac{b}{g}-c}) \tag{11}$$

Here are the reasons for using the $f(g)$ as the regression line:

1. Both the data and *f(g)* have the limit

When the group number divided g largely exceeds the p, the number of the positive group will also be p, because they are more prone to divide evenly into the groups. For example, if there are 100,000 groups divided, but there are only 5 positive samples, they will evenly separate into different groups, because the probability for 2 of them to occur in the same group is far too small. Therefore, as g increases, the E will approach to p. Same, $f(g)$ also has the limit when g approaches infinity, which

$$\lim_{g\to\infty} p \cdot (e^{-\frac{b}{g}-c}) = p \cdot \lambda \tag{12}$$

Where λ is minor constant producing by the coefficient c. However, as the regression (13) showed, c is a very small constant (-0.106219) which won't influence the results of the regression in a great extent. Also, $R^2 = 0.92$[1] for Figure 7, which is in a acceptable range. Therefore, it's reasonable to do this approximation.

2. Regression line fits the trend of the data

The data increases at first, but the speed of increase slows down gradually. $f(g)$ fits this data trend before the g approaches the limit.

Therefore, through the regression, we simplify E into a fundamental function that is easy to utilize and calculate in the following process, which

$$f(g) = 360 \cdot (e^{-\frac{237.528}{g}+0.106219} + 0.105781) \tag{13}$$

And the new T is expressed as

$$T(10000, 360, g) = g + \frac{n \cdot 360 \cdot (e^{-\frac{237.528}{g}+0.106219} + 0.105781)}{g} \tag{14}$$

[1] R^2 value describes the goodness of fit of a regression function, exact calculation procedures please check Appendix 7.3

So, the optimization for the two stages testing model is expressed as

$$\min\ (g + \frac{n \cdot 360 \cdot (e^{-\frac{237.528}{g}+0.106219} + 0.105781)}{g})$$

$$s.t. \quad 1 \leq g \leq 10000$$

As shown in the Figure 8, the expression is optimized when $g = 1858.387$, but it can only be integer in reality. Therefore, for the integer solution, when $g = 1858$, the expression reaches the minimum with 3960 numbers of testing (upper rounding result).

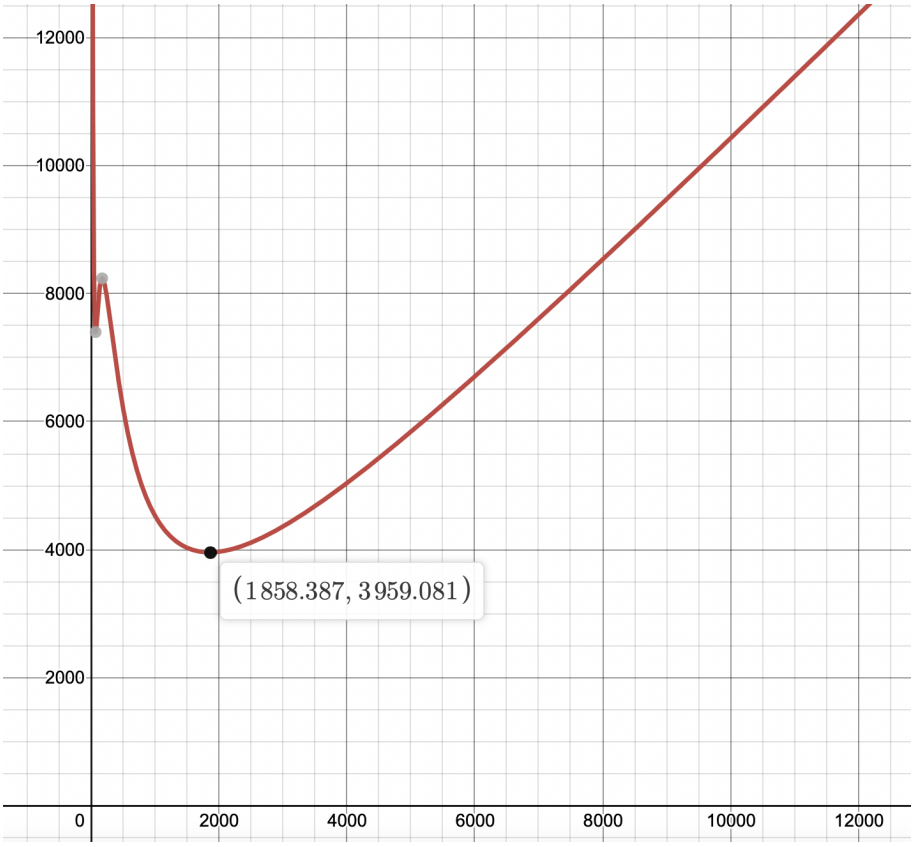

Figure 8: Optimization Figure

3.3 Merge Algorithm Model Establishing

3.3.1 The introduction to Merge Algorithm

The Merge Algorithm is a method that widely uses in computer sorting by separating a massive and complex array into different subsections and then combining them together in the end.

3.3.2 The methodology of the model

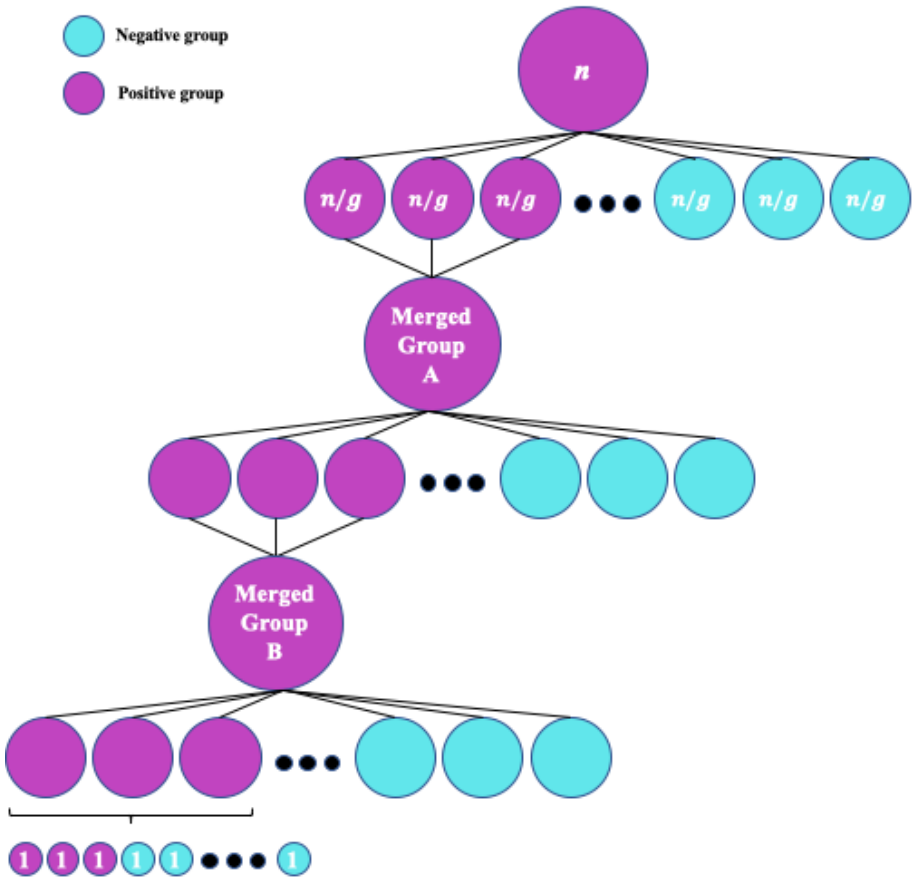

Figure 9: Merge Algorithm model example

As shown in Figure 9, same as the previous method, the largest group will be divided into different groups. Differently, for all positive groups, we will merge them together and then separate them into a smaller group.

The purpose of this operation is to shrink the positive samples into a condenser scale. Each time, if the number of the group divided exceeds the number of positive sample, it must produce negative

groups.

For example, if there are 5 positive people in the merged group, and we are going to divide this merged group into 7 smaller group, we can ensure that 2 of the smaller groups must be negative group.

Nevertheless, all the algorithms have the requirement to end. Here is the ending situation for this algorithm:

When optimized groups going to be divided is less than the number of the positive cases

When this happens, we can't ensure that a negative group will occur and it's possible for all the groups to have a positive sample. **The merge algorithm can only be efficient when we can ensure that it can produce negative groups.** Otherwise, the merge ends.

The following flowchart shows how will the merge algorithm work:

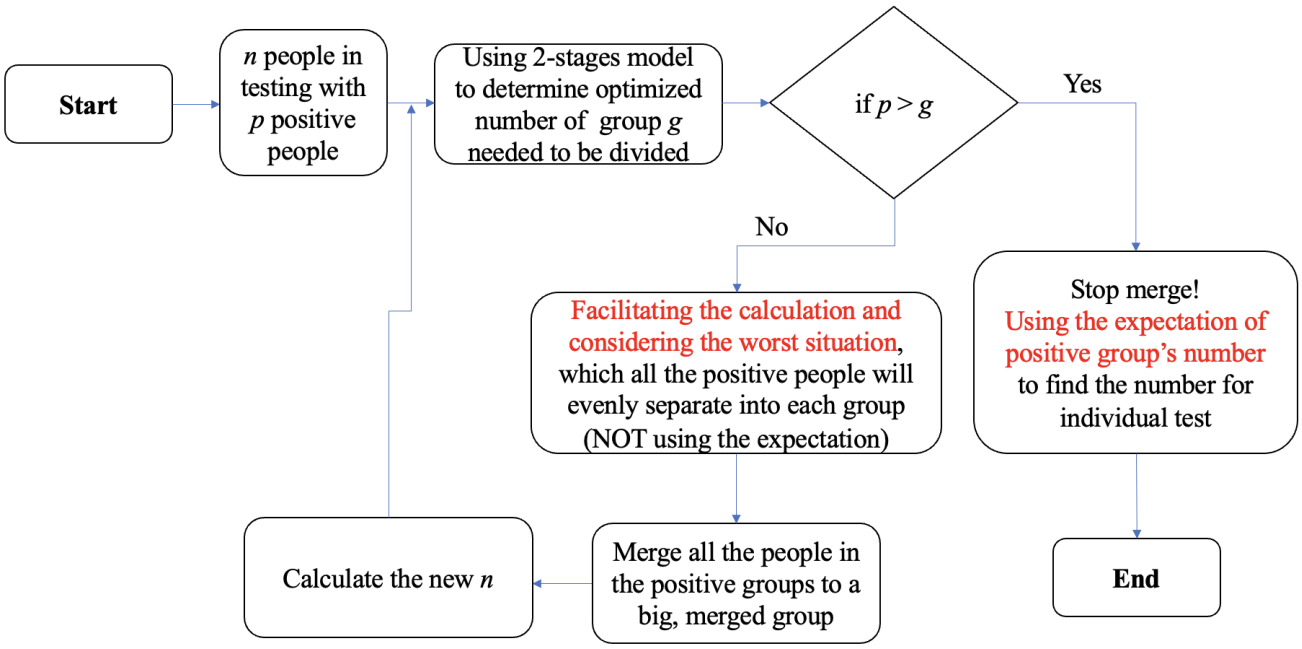

Figure 10: Merge Algorithm Results

3.4 Data Calculation

This section will use data from USA, which $n = 10000$ and $p = 360$.

3.4.1 Merge Algorithm Example Calculation

In section 3.2, we have already determined that for stage 1, the optimized plan to separate 10,000 people into 1858 groups (to simplify the calculation, we will use the average number of people in group as calculation, which is 5.38 people per group). **Considering the worst situation that all the positive samples are evenly separated into each group** (this also facilitate the calculation by considering the simplest situation), there are going to have 360 positive groups.

Merging all the samples in the 360 positive groups together, so in the merged group A, we will have 1937 people (5.38 people per group multiples 360 groups) with 360 people affected by COVID. Therefore, we successfully shrink the positive sample into a smaller scale. The new n_2 is 1937 and p stays the same.

So what we are going to solve is to find the optimized groups for $n = 1937$ and $p = 360$. By changing n, its relative expectation E also changes, and our new optimized expression is

$$\begin{aligned} &\min\ (g + \frac{n \cdot 360 \cdot (e^{-\frac{237.1285}{g}+0.11142} + 0.105443)}{g}) \\ &s.t. \quad 1 \leq g \leq 1937 \\ &\qquad n = 1937 \end{aligned}$$

The optimized groups need to be divided for this situation is $g_2 = 73$, and each group will contain 27 people (rounding result, should be 26.53 people per group).

However, $g_2 < p$, which we can't ensure that there must have a negative group, because it is possible for all the people affected by COVID to spread into all groups (360 people to spread into 73 groups). So we will end the merge, using the expectation of the number of positive groups as calculation. Using the expectation times the number of people in each group, we will get the number of individual testing.

The total testing number is 1858+73+1422=3353 (1422 is for the individual test after 73 groups are been divided, this number is based on the expectation of how many positive groups will occur in the second shuffling).

Merge algorithm provides us a chance to reduce the number of testing compared to the 2-stages testing model. We have the probability to reduce the testing time from 3960 to 3353.

Figure 11 shows the details illustrated above.

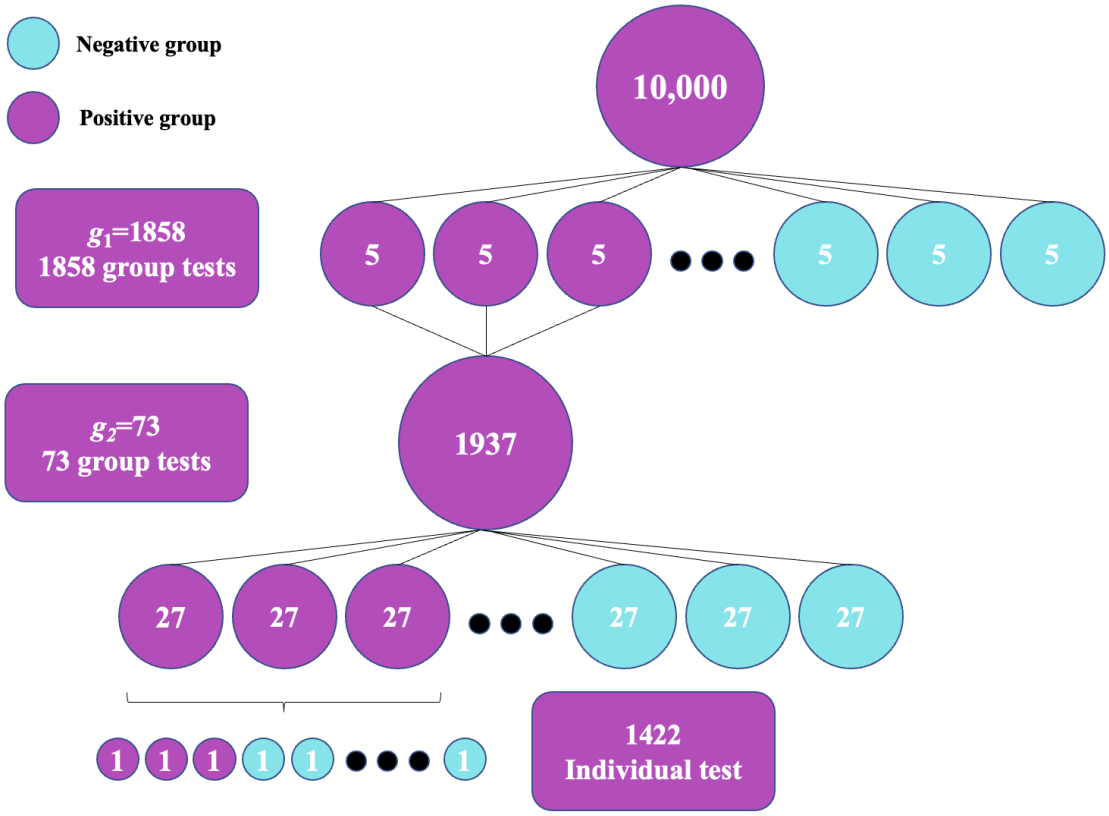

Figure 11: Merge Algorithm Results

4 Evaluation of the both model

4.1 Higher Positive Rate Example Calculation——The Data from India

This paper chooses the data from the India as comparison of the two algorithms. According to *COVID19INDIA*, the positive testing rate is about 6.2%, so there will be 620 positive cases from 10,000 people.

For the binary model, the result is:

$$T(10000, 620) = 1240(\lceil \log_2 10000 \rceil - \lfloor \log_2 620 \rfloor - 1) + \sum_{i=0}^{\lfloor \log_2 620 \rfloor + 1} 2^i = 7007 \tag{15}$$

The first stage of the merge, model is expressed as

$$\begin{aligned} &\min\ (g + \frac{n \cdot 620 \cdot (e^{-\frac{369.01342}{g} + 0.0159} + 0.09378)}{g}) \\ &s.t. \quad 1 \leq g \leq 10000 \\ &\qquad n = 10000 \end{aligned}$$

The first stage optimized group number is 2247 (with average 4.45 people per group). Therefore, considering the worst situation, which all the positive samples are evenly separated into 620 groups. Merging 620 positive groups together, we will get the merged group with 2759 people (4.45 multiples 620), so the optimization for the second stage is expressed as

$$\begin{aligned} &\min\ (g + \frac{n \cdot 620 \cdot (e^{-\frac{380.5756}{g} + 0.07758} + 0.09649)}{g}) \\ &s.t. \quad 1 \leq g \leq 2759 \\ &\qquad n = 2759 \end{aligned}$$

The optimized group number is 114, which is less than the number of the positive sample. Therefore, the merge ends, and all positive groups start the individual testing. The total testing times is 4383 (2247+114+2023, as shown in Figure 12:)

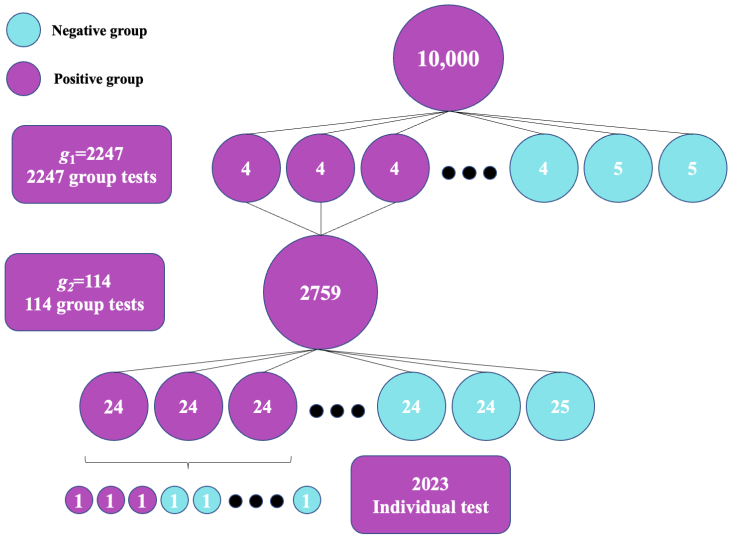

Figure 12: Merge Algorithm Results

5 Model Improvement based on *FNT* (False Negative Testing)

5.1 Definition of *FNT* in the Pool Testing

False Negative Testing stands for the patient who infects by the virus but with the negative result in the testing. With the negative results, the FNT patients are prone to spread the virus to the people who are healthy, leading to the negative externality. Therefore, the most straightforward method to control the COVID is by deterring the transmission of the virus from FNT patient.

The false positive testing is caused by the insensitivity of the machine. These in-sensitivities of the machine are largely due to the minor concentration of the COVID DNA, which is very hard for the machine to detect it from a large sample. This means that the pool testing sacrifices the accuracy of the testing and focuses on the speed of the testing. According to the clinic study conducted by the Columbia University and Weill Cornell Medicine, the false negative rate for the first testing is about 17.0%.

Therefore, by using the definition of the *FNT* and the false testing data, we can apply ***Bayes Theorem*** to solve the probability of the *FNT*. As shown in the Figure 13, the probability for one positive group to appear is the number of positive group p over the total number of groups going to be divided n, which is $\frac{p}{n}$. The probability of each scenario to be happened is shown in the tree diagram, Figure 13:

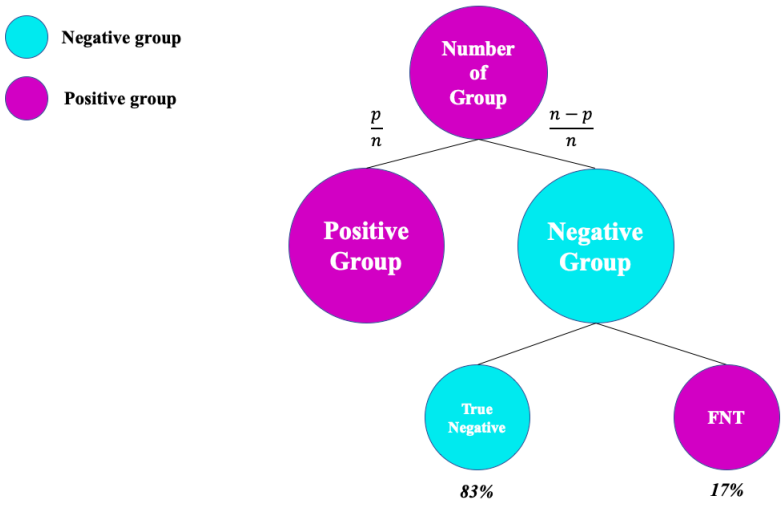

Figure 13: Probability for False Negative testing

The probability for the false negative testing to be happened is the false negative result but with the condition of the negative testing result, expressing it using the Bayes Theorem:

$$P(false|negative) = \frac{P(false \cap negative)}{P(negative)} \tag{16}$$

$$\because P(false \cap negative) = 0.17, P(negative) = \frac{n-p}{n} \tag{17}$$

$$\therefore P(false|negative) = \frac{0.17 \cdot n}{n-p} \tag{18}$$

Therefore, we get the probability for a positive mixed sample but with negative testing result.

5.2 Strategy to deal with *FNT*

We can deal with the FNT by repeatedly testing the mixed sample which with the negative result. Through the repetition of testing, we can lower down the probability for the occurrence of the FNT. Thus, we can establish the False Testing Rate probability function to calculate the relationship between the number of the repeated test and the false negative rate for a single mixes sample:

$$P_{FNR}(n,p,r) = (\frac{0.17 \cdot n}{n-p})^{r} \tag{19}$$

The expression in the bracket stands for the conditional probability we get in 3.5.1, where n stands for the total number of groups going to be divided, p stands for the number of positive groups that is going to be appeared, and r is the number of repeated testing for the each negative group.

However, different country has different standards towards the false negative testing rate, and none of them discloses the restrictions. Therefore, for this essay, we set the restriction of the probability for the false negative testing that must lower than $CR\%$, which

$$(\frac{0.17 \cdot n}{n-p})^{r} < CR\% \ (0\% < CR\% < 11\%) \tag{20}$$

Because we can not get an accurate CR through the disclosed resource, we will dynamically adjust the $CR\%$ from 0% to 10% to see how will the restriction of the FNT rate influences the number of the test. Figure 14 is the example for $CR = 1\%$: (All the calculations are based on Merge Algorithm)

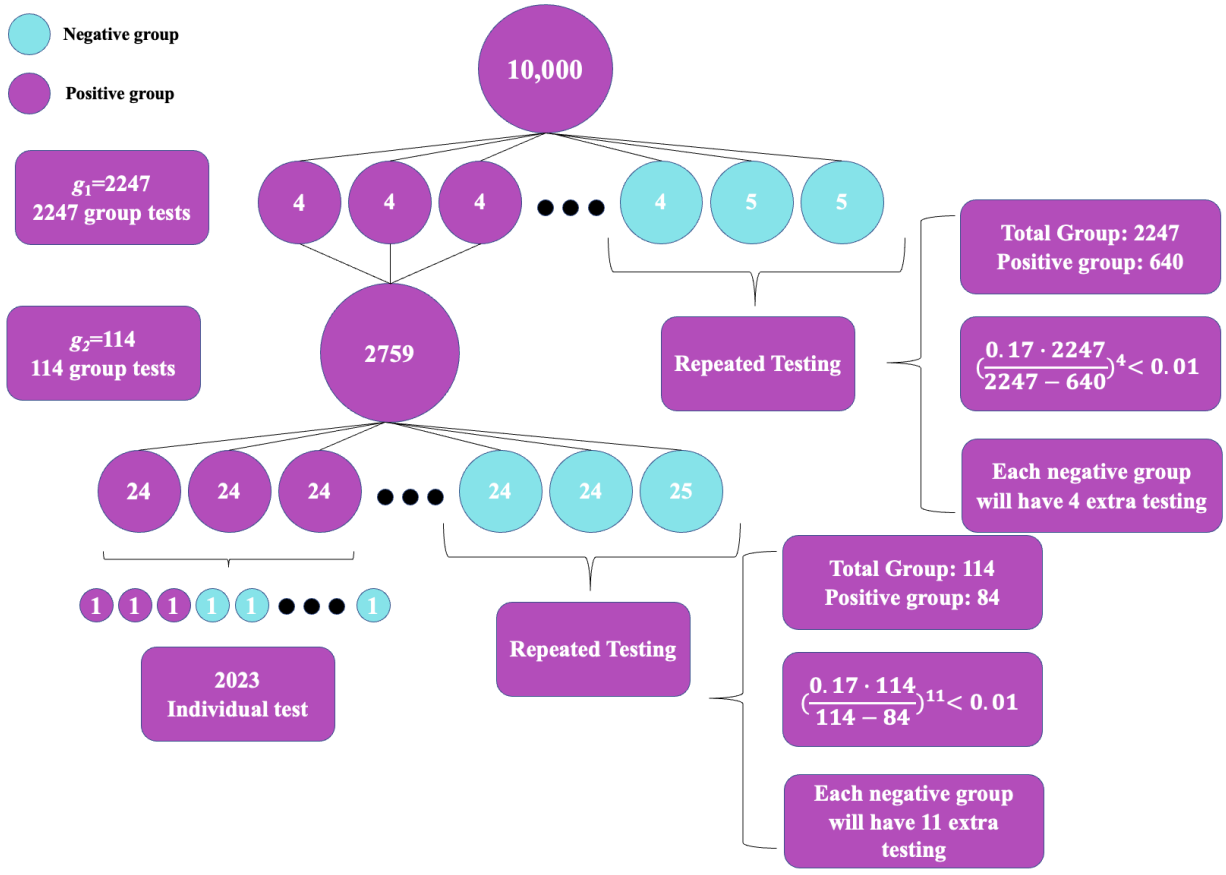

Figure 14: FNT Real Example Calculation: Based on India

Through the example above, we need to re-test the sample of mixed negative group in stage 1 and

stage 2. In stage 1, which 4 to 5 people are put into each group, we have 2247 mixed sample, which $n = 2247$, and there are 640 positive groups, which $p = 640$. At $CR = 1\%$, by using the expression (19), we find that $r = 4$ can satisfy the CR. Each negative group in the first stage requires 4 extra testings to lower down the probability of the FNT. There are 1296 negative groups in total, so 6428 extra testings are required.

For the second stage, we have 114 mixed samples, which $n = 114$, and 84 of them are positive groups, which $p = 84$. To satisfy the CR, each negative group requires 12 extra testings. There are 30 negative groups, so 330 extra testing is required.

Individual test does not require the repeated testing, because we reckon the main reason caused the FNT is the insufficient concentration of positive sample during the mixed testing.

In total, when the $CR = 1\%$, 6758 extra testing is required. Repeating the process above, we can get the result for both the USA (10,000 people with 360 positive cases) and the India (10,000 people with 640 positive cases), shown in the Table 2 [2]:

Table 2: Extra testing based on different CR

CR	**INDIA**	**USA**	CR	**INDIA**	**USA**
1%	6758	5544	6%	3424	4188
2%	5091	4188	7%	3424	2802
3%	5091	4158	8%	3394	2802
4%	5061	4128	9%	3394	2772
5%	5031	4128	10%	3394	2772

Different countries might have different situation. For the country that has enough testing kit, they can adopt the more strict testing mode. However, for those country with inadequate testing kit, they can adopt less strict testing mode.

[2]Exact calculating code please check the Appendix 7.4

6 Conclusion

This paper establishes two pool testing models and uses the data from two different places to compare the efficiency of the model.

For the low positive rate (360 positive samples among 10,000 people), the Binary algorithm requires 4623 number of testing, and the Merge Algorithm requires 3353 number of testing, increasing the efficiency by 27.5%.

For the high positive rate (620 positive samples among 10,000 people), the Binary algorithm requires 7007 number of testing, and the Merge Algorithm requires 4383 number of testing, increasing the efficiency by 37.4%.

Therefore, it can be concluded that the merge algorithm is more efficient in pool testing than the binary algorithm. Basing on the result above, this paper also extends to the improvement towards the more efficient Merge Algorithm, establishing the model to calculate the False Negative Testing and optimized repeated testing strategy. Basing on the data from the USA and the India, this paper dynamically alters the CR of the False Negative Testing from 1% to 10% and finds the corresponding testing times.

Furthermore, this model might not only be applied to the COVID-19 Pool Testing but also various infectious diseases. However, other diseases might have different ways to mix the sample. Therefore, to establish a general Pool Testing optimization model is the further way to extend my model.

References

[1] Center for Devices and Radiological Health. (n.d.). Pooled Sample Testing and Screening Testing for COVID-19. U.S. Food and Drug Administration. https://www.fda.gov/medical-devices/coronavirus-covid-19-and-medical-devices/pooled-sample-testing-and-screening-testing-covid-19.

[2] Green, D. A., Zucker, J., Westblade, L. F., Whittier, S., Rennert, H., Velu, P., Craney, A., Cushing, M., Liu, D., Sobieszczyk, M. E., Boehme, A. K., amp; Sepulveda, J. L. (2020). Clinical Performance of SARS-CoV-2 Molecular Tests. Journal of Clinical Microbiology, 58(8). https://doi.org/10.1128/jcm.00995-20

[3] United States COVID-19 Statistics: 33,106,232 Cases / 589,740 Deaths / 363,825,123 Tests / Avg cases/day 68,121 declined 25.1 from 14 days ago Avg deaths/day 2,034 declined 17.22 from 14 days ago (Updated May 23, 2021 @ 12:24pm).covidusa.net. (n.d.). https://covidusa.net/.

[4] What is 'pool testing' and does it really expedite Coronavirus detection? Business Insider. (2020, April 13). https://www.businessinsider.in/india/news/what-is-pool-testing-and-does-it-really-expedite-coronavirus-detection/articleshow/75121771.cms.

7 Appendix

7.1 Appendix - Binary Model

```
import java.util.*;
public class BinaryUSAExample {
       static int p,n;
       static double log(int basement, int n){return Math.log(n)/Math.log(basement);}
       public static void main(String[] args) {
          //set the number of the positive cases and total sample
              n = 10000;
              p = 360;
              int ans = 0;
              for(int i=0;i<=(int)Math.floor(log(2,p))+1;i++) {
                     ans += Math.pow(2, i);
              }
              int BranchedStage = (int)
                  ((Math.ceil(log(2,n)))-Math.floor(log(2,p))-1);
              ans += 2*p*BranchedStage;
              System.out.println(ans);
       }
}
```

7.2 Appendix - Expectation Regression

```
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
#Define regression function
def func(x, a, b,c):
   return (p*(np.exp((b/x)-c)-a))
```

```
x = [0 for i in range(998)]
num = [0 for i in range(998)]
for i in range(0,998):
    x[i] = i+2
#Set the total testing case and the number of the patient
t = 10000
p = 360
#Calculate the expression of expectation
for g in range(2,1000):
    TotalP = 0
    for j in range(1,p):
        C =math.factorial(p)/(math.factorial(j)*math.factorial(p-j))
        P = j*C*((1-((1-g/t)**(t/p)))**j)*(((1-g/t)**(t/p))**(p-j))
        TotalP += P
    num[g-2] = TotalP
#Regression
x = np.array(x)
y = np.array(num)
popt, pcov = curve_fit(func, x, y)
print(popt)
a = popt[0]
b = popt[1]
c = popt[2]
yvals = func(x,a,b,c)
print('popt:', popt)
print('Coefficient of a:', a)
print('Coefficient of b:', b)
print('Coefficient of c:', c)
print('Copcov:', pcov)
print('Coyvals:', yvals)
plot1 = plt.plot(x, y, 'aqua',label='Data-Expectation',linewidth=2.8)
plot2 = plt.plot(x, yvals, 'm',label='Regression Line',linewidth=2.8)
```

```
plt.xlabel('g (groups going to be divided)')
plt.ylabel('E (Expectation of the number of the positive groups)')
plt.legend(loc=4)
plt.title('Regression for the Expectation of the number of the positive
    group(n=10,000 p=360)')
plt.show()
```

7.3 Appendix - Optimization Solving and R^2Calculation

```
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
def func(x, a, b,c):
    return x+Total*(p*(np.exp((b/x)-c)-a))/x
#Number of the total testing and positive samples
Total = 10000
p = 640
#Coefficient from Appendix 2
a = -0.0964944
b = -380.575682534
c = -0.077584552
#Optimization Solving
ma = 99999999
maval = 99999999
for i in range(1,10000):
    if(func(i,a,b,c))<maval:
        ma = i
        maval=func(i,a,b,c)
print(ma)
print(maval)
```

```
import math
import numpy as np
x = [0 for i in range(998)]
num = [0 for i in range(998)]
def func(x, a, b,c):
    return (p*(np.exp((b/x)-c)-a))
t = 10000
p = 360
for i in range(0,998):
    x[i] = func(i+1,-0.105781,-237.528,-0.106219)
for g in range(2,1000):
    TotalP = 0
    for j in range(1,p):
        C =math.factorial(p)/(math.factorial(j)*math.factorial(p-j))
        P = j*C*((1-((1-g/t)**(t/p)))**j)*(((1-g/t)**(t/p))**(p-j))
        TotalP += P
    num[g-2] = TotalP
MidX = 0
SumX = 0
SST = 0
SSR = 0
for i in range(0,998):
    SumX = SumX + x[i]
MidX = SumX/999
for i in range(0,998):
    SST = SST + (x[i]-MidX)**2
    SSR = SSR + (num[i]-MidX)**2
R = SSR/SST
print(R)
```

7.4 Appendix - FNT Testing Times Calculation and Data Results

```
#Function for FNT
def Func(n,p,r):
    Val = (0.17*n/(n-p))**r
    return Val
def Solve(n,p,t):
    r = 1
    while(Func(n,p,r)>t):
        r = r + 1
    return r
#Number of the group and number of positive group
n = 117
p = 87
for ControlRate in range(1,12):
    t = ControlRate/100
    ans = Solve(n,p,t)
    print(ans)
```