Algorithms - Semester Project

charles.schmitz2@marist.edu

December 2020

---

https://github.com/charlesschmitz2/CMPT435—Algorithms/tree/Semester-Project—Pooled-Testing-Simulation/Algorithms—SemesterProject

1 Analysis

—-Project Goal—-
Program a simulation of the testing protocol described in Alan's article. Run your simulation on population of 1000 people (at first) in groups of 8 assuming a 2 percent infection rate and 100 percent accurate tests. Output the results in a manner similar to those shown below:

Using the protocol described in Alan's article, there are three possibilities to consider:
(1) there are no infected samples
(2) there is exactly one infected sample
(3) there are two or more infected samples

ALL WITH GROUPS OF 8 NUMBERS NEEDED :
For case (1) the likelihood of randomly choosing 8 uninfected samples is (0.98)8 = **0.85** or 85 percent, here we have a **1** test per group of 8.
For case (2) the likelihood of randomly choosing ... **7** tests per 1 infection in group $-->$ **0.1496** or abut 15 percent of the time
For case (3) the likelihood of randomly choosing 2 infected samples is 0.**0004** or 0.04percent $-->$ **11** tests are needed

EXAMPLE RUN FROM ASSIGNMENT :
So, for 1000 people where 20 of them (2 percent) are infected and 980 are infection-free, we could make 125 groups of 8 samples each and work out what we expect based on the percentages:
Case (1): $125 \times 0.8500 = 106.25$ instances requiring 107 tests (since there are no partial tests) $-->$ 107x1
Case (2): $125 \times 0.1496 = 18.70$ instances requiring 131 tests $-->$ 19x7
Case (3): $125 \times 0.0004 = 0.05$ round up to 1 instance requiring 11 tests $-->$ 1x11

That's 249 tests to screen a population of 1000 people for a disease with an infection rate of 2 percent.

FUNCTIONS IF WORKING WITH DIFFERENT GROUP SIZES :
here it is the for group sizes of 8, this can be substituted for other values as well such as 4,2, whatever. The x represents the infection rate, here we are working with 2 percent
Case (1) $--> y = [(1 - x/100)^8)] * 100$
Case (2) $--> y = [(x/100)^2] * 100$
Case (3) $--> y = 100 - [(100 * (1 - x/100)^8) + (x/100)^2]$

MY APPROACH :

1. Use list to store our groups of 8, just taking groups of 8 starting at 0 up to the amount Inputted/8 or number of groups of 8
2. Produce output of likelihood of each case. NOT set as constants
3. Use a little I/O console input
4. Use a random() function and 0/1 to when testing the groups to see if positive or negative test
5. Assume 100 percent accuracy
6. Testing :

   If we are testing a group of 8 and there is an infection, then we will test four and four, if there is one or more infections from those tests then each one is tested individually.

   First, I will take input on simulation size and that sort of thing

   Second, I will create an list of the inputted simulation size, then calculate amount of group - (simulationSize/groupSize[here using 8])

   Third, I will loop through the list of people to be tested grabbing 8 at a time for the amount of time calculated in the previous step

   counter = 0;
   int i = 0
   while(i < calculated Number of Groups)
   -for(int j = 0; j < 8; j++)
   –if(list[counter].test(random function that gives either a 0 or a 1) == 1 [positive])
   list[counter].setSickness to positive I will be using a list of nodes that have an attribute of sickness which is either pos. or neg.
   –else
   do nothing since all people will be added and assumed to be negative...innocent until proven guilty as they say.

   Fourth, I should store and print out and compare these results to the statistically expected values that will be calculated and outputted alongside my test results these should match or be close to the expected

The diagram below is information taken from the long article provided to us and depicts how you can go about determining the number of instances, this number is multiplied for each case by the number of groups when split up into respective group sizes.

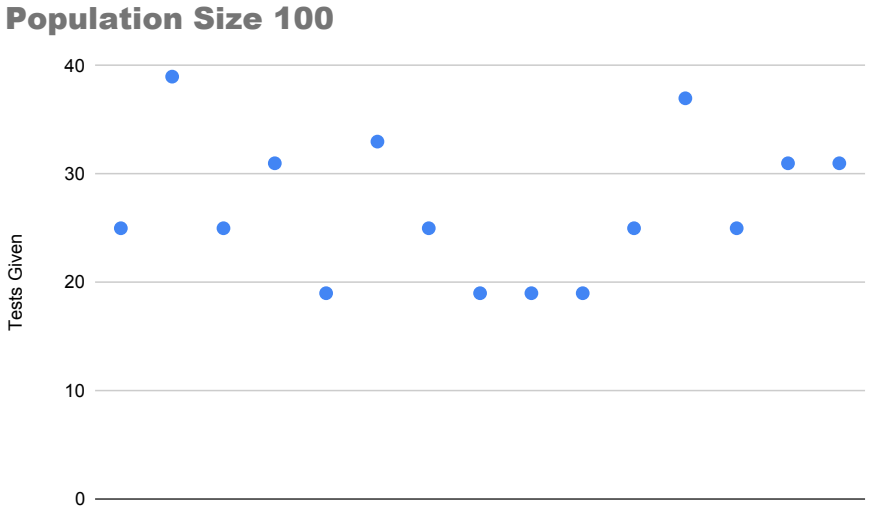
$$y = \left[\left(1 - \frac{x}{100}\right)^{groupsize}\right) \cdot 100\right] \Rightarrow \text{best case (1)}$$

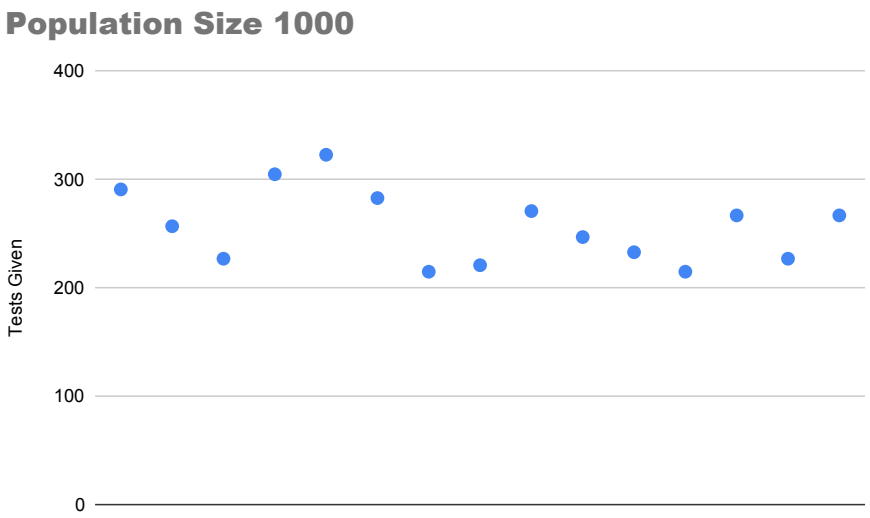
$$y = 100 - \left[\left(100 \cdot \left(1 - \frac{x}{100}\right)^{groupsize}\right) + \left(\frac{x}{100}\right)^{12}\right] \Rightarrow \text{middle case(2)}$$

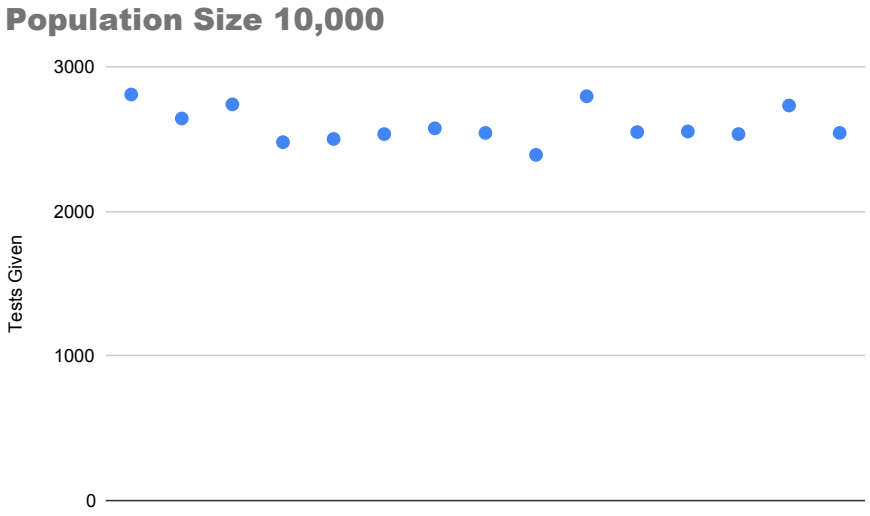
$$y = \left(\frac{x}{100}\right)^{2} \cdot 100 \Rightarrow \text{worst case (3)}$$

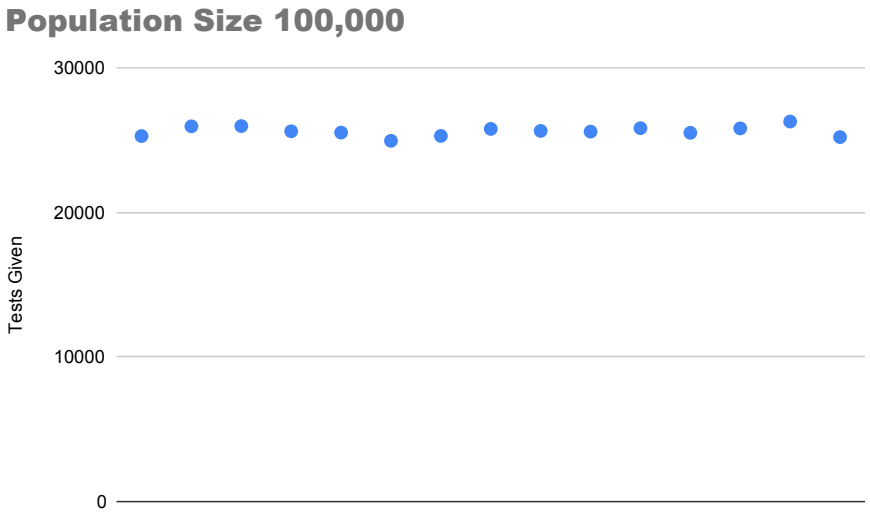
$$X = \text{infection rate}$$

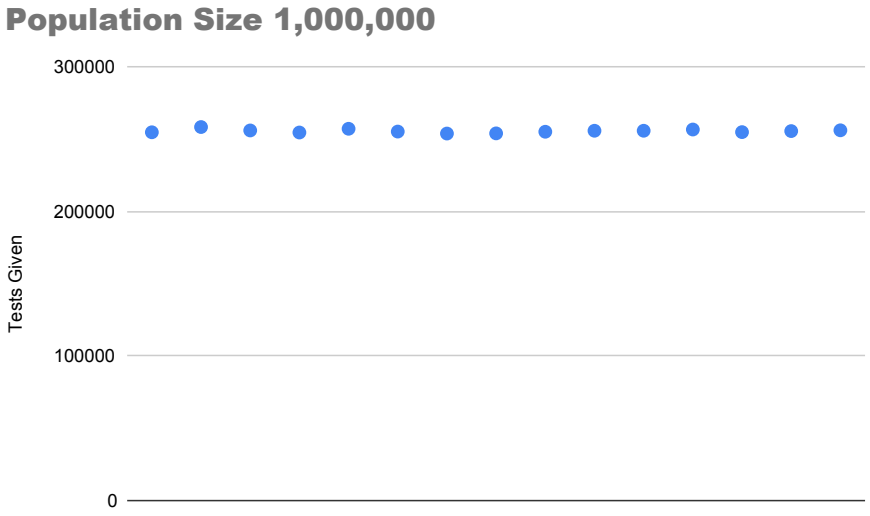
Our program simulates the exact random chances with all the factors described above. By running our simulation and averaging out the results we can see that these do in fact match the statistical values. Below to illustrate my point I have included graphs of the simulation data running at different population sizes with our default settings. As more and more people are tested we are able to see its consistency, and how it matches the number of tests that should be prepared to be given.




Population Size 1000
400
300
200
100
0
Tests Given


Population Size 10,000
3000
2000
1000
0
Tests Given





If you wish to mess around and see the different types of results you can get outside of this assignment, I have implemented the ability to easily change group size for the pools, as well as an easy change within my code to change the infection rate to analyze how these affect this pooled testing method.

2 Main Class

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;


public class SemesterProjectMain {

    /* ----Project Goal----
    Program a simulation of the testing protocol described in Alans article. Run your
    simulation on population of 1000 people (at Lirst) in groups of 8 assuming a 2%
    infection rate and 100% accurate tests. Output the results in a manner similar to those
    shown below.
    Using the protocol described in Alans article, there are three possibilities to consider:
        (1) there are no infected samples
        (2) there is exactly one infected sample
        (3) there are two or more infected samples
    */

    //ALL WITH GROUPS OF 8 NUMBERS NEEDED :
    //For case (1) the liklihood of randomly choosing 8 unifected samples is (0.98)^8 = **0.85** or 85%, here we have a **1** test per group of 8
    //For case (2) the liklihood of randomly choosing ... **7** tests per 1 infectiom in group --> **0.1496** or abut 15% of the time
    //For case (3) the liklihood of randomly choosing 2 infected samples is 0.**0004** or 0.04% --> **11** tests are needed

    /* EXAMPLE RUN FROM ASSIGNMENT :
    So, for 1000 people where 20 of them (2%) are infected and 980 are infection-free, we could make
    125 groups of 8 samples each and work out what we expect based on the percentages:
    Case (1): 125    0.8500 = 106.25 instances requiring 107 tests (since there are no partial tests) --> 107*1
    Case (2): 125    0.1496 = 18.70 instances requiring 131 tests --> 19*7
    Case (3): 125    0.0004 = 0.05 round up to 1 instance requiring 11 tests --> 1*11



    That's 249 tests to screen a population of 1000 people for a disease with an infection rate of 2%.
    */

    //FUNCTIONS IF WORKING WITH DIFFERENT GROUP SIZES : here it is the for group sizes of 8, this can be substituted for other values as well such as 4,2, whatever. The x represents the
```

```
    infection rate, here we are working with 2%
35  //Case (1) --> y = [(1-x/100)^8)]*100
36  //Case (2) --> y = [(x/100)^2]*100
37  //Case (3) --> y = 100 - [(100*{(1-x/100)^8}) + (x/100)^2]
38
39  /* MY APPROACH :
40      1. Use list to store our groups of 8, just taking groups of
     8 starting at 0 up to the amountInputed/8 or # of groups of 8
41      2. Produce output of liklihood of each case. NOT set as
    constants
42      3. Use a little I/O console input
43      4. Use a random() function and 0/1 to when testing the
    groups to see if positive or negative test
44      5. Assume 100% accuracy
45      6. Testing :
46          If we are testing a group of 8 and there is an
    infection, then we will test four and four, if there is one or
    more infections from those tests
47          then each one is tested individually.
48      First, I will take input on simulation size and that sort
    of thing
49      Second, I will create an list of the inputted simulation
    size, then calculate amount of group - (simulationSize/
    groupsize[here using 8])
50      Third, I will loop through the list of people to be tested
    grabbing 8 at a time for the amount of time calculated in the
    previous step
51              counter = 0;
52              int i = 0
53              while(i < calculated Number of Groups){}
54                  for(int j = 0; j < 8; j++){
55                      if(list[counter].test(random function that
    gives either a 0 or a 1) == 1 [positive])
56                          list[counter].setSickness to positive
    // I will be using a list of nodes that have an attribute of
    sickness which is either pos. or neg.
57                      else
58                          do nothing since all people will be
    added and assumed to be negative...innocent until proven guilty
     as they say.
59                  }
60              }
61      Fourth, I should store and print out and compare these
    results to the statistically expected values that will be
    calculated and outputted alongside my test results
62              these should match or be close to the expected
63  */
64
65
66  public static int menuSelection = 0;
67  public static int infectionRate = 2; //this represents a 2%
    infection rate
68  public static int infectedCounter = 0;
69  public static int groupSize = 8; //this is our default size
    that we are running for this project, could adjust if desired
70  public static int numTests = 0;
71  public static int numPeople = 1000;
```

```
72
73      public static List<Person> people = new ArrayList<>(1000);
74      public static ListPeople peopleList = new ListPeople(people);
75
76  /*----Main----*/
77      public static void main(String[] args) {
78
79          System.out.println("\n\nHello & Welcome to my Pooled
        Testing Simulation : ");
80          System.out.println("\n\tInformational output will be
        provided along the journey, you may continue to keep running
        the simulation with varying");
81          System.out.println("\ttest numbers so once you have
        finished press '8' to quit. You may alter the number of people
        the simulation is performed on,");
82          System.out.println("\tas well as the group size that these
        individuals are split into when testing. For this project the
        default is set to **group sizes of 8**,");
83          System.out.println("\tbut can be altered if desired by
        pressing '6' within our menu selection. **Default simulation
        size is 1,000** as well. I hope you find my method of testing
        satisfactory.");
84          System.out.println("\tAlso please note that our **infection
         rate of this disease is 2%**, you can run this program with
        different infection rates but this must be changed within the
        code itself.");
85          System.out.println("\n\tPlease note that these test values
        are near or at the expected values, I do not list the expected
        within this program as that is detailed within");
86          System.out.println("\tmy LaTeX documentation found on
        Github. By running the program multiple times and averaging the
         actual results you can match the expected values.");
87          System.out.println("\nStay safe and enjoy!");
88
89          //Run the simulation until you quit
90          do{
91              System.out.println("\n-------------RUNNING SIMULATION
        --------------");
92              System.out.println("CURRENT SIMULATION SIZE : " +
        numPeople);
93              System.out.println("CURRENT GROUP SIZE : " + groupSize)
        ;
94
95          } while(runSimulation());
96
97
98
99      }// main
100
101 /*----Simulation Functions----*/
102         //This is the function where the simulation is run and
        analized
103         public static boolean runSimulation(){
104
105             menuSelection = menu();
106             infectedCounter = 0;
107             numTests = 0;
```

```
108
109
110             if (menuSelection == 1){
111                 System.out.println("\n----Running Simulation with
    100 People---- \n");
112                 numPeople = 100;
113                 peopleList.addPeople(numPeople); //each time an
    addPeople is run the list is cleared and refilled so the
    program can run multiple cycles of differnet simulation sizes
114                 System.out.println("\n----The Infection has Begun
    to Spread---- \n");
115                 runInfection();
116                 test();
117                 System.out.print("\n \u2022--> Number of Tests
    Given : " + numTests + " <--\u2022\n");
118                 //peopleList.print();
119
120             } // if 100
121             else if (menuSelection == 2) {
122                 System.out.println("\n----Running Simulation with
    1,000 People---- \n");
123                 numPeople = 1000;
124                 peopleList.addPeople(numPeople);
125                 System.out.println("\n----The Infection has Begun
    to Spread---- \n");
126                 runInfection();
127                 test();
128                 System.out.print("\n\u2022--> Number of Tests Given
     : " + numTests + " <--\u2022\n");
129                 // peopleList.print();
130
131             } // if 1000
132             else if (menuSelection == 3) {
133                 System.out.println("\n----Running Simulation with
    10,000 People----\n");
134                 numPeople = 10000;
135                 peopleList.addPeople(numPeople);
136                 System.out.println("\n----The Infection has Begun
    to Spread---- \n");
137                 runInfection();
138                 test();
139                 System.out.print("\n\u2022--> Number of Tests Given
     : " + numTests + " <--\u2022\n");
140                 // peopleList.print();
141
142             } // if 10000
143             else if (menuSelection == 4) {
144                 System.out.println("\n----Running Simulation with
    100,000 People----\n");
145                 numPeople = 100000;
146                 peopleList.addPeople(numPeople);
147                 System.out.println("\n----The Infection has Begun
    to Spread---- \n");
148                 runInfection();
149                 test();
150                 System.out.print("\n\u2022--> Number of Tests Given
     : " + numTests + " <--\u2022\n");
```

```
151                 // peopleList.print();
152 
153             } // if 1000000
154             else if (menuSelection == 5) {
155                 System.out.println("\n----Running Simulation with
    1,000,000 People----\n");
156                 numPeople = 1000000;
157                 peopleList.addPeople(numPeople);
158                 System.out.println("\n----The Infection has Begun
    to Spread---- \n");
159                 runInfection();
160                 test();
161                 System.out.print("\n\u2022--> Number of Tests Given
     : " + numTests + " <--\u2022\n");
162                 // peopleList.print();
163 
164             } // if 10000000
165             else if(menuSelection == 6){
166                 menu2();
167                 return true;
168             }//if want to change the group size
169             else {
170                 System.out.print("Quitting Program, please come
    again!");
171                 return false;
172             } // else quitting
173 
174             return true;
175 
176         }//runSimulation
177 
178 /*----Menu Functions----*/
179     public static void runInfection() {
180         peopleList.diseaseGiver(infectionRate);//this is the
    function that infects people at the infection rate that is set
    (2% for us)
181                 for(int i = 0; i < peopleList.size(); i++){
182                     if (people.get(i).getIsSick()==1){
183                         if(peopleList.size()<=10000){System.out.
    println("\t\u2022 Person " + i + " has been infected");}
184                         infectedCounter++;
185                     }//if
186 
187                 }//for
188 
189                 if(peopleList.size()==1000000){System.out.println("
    1,000,000 is way too many to print out around 20,000 'has been
    infected' messages");}
190                 if(peopleList.size()==100000){System.out.println("
    100,000 is too many to print out around 2,000 'has been
    infected' messages");}
191 
192                 if (infectedCounter == 0){
193                     System.out.println("\tNo one was infected,
    unlikely but always possible...expect the unexpected");
194                 }
195                 else{
```

```
                    System.out.println("\n-->" + infectedCounter +
" people were infected\n\n");
                }
    }//runInfection

    public static void test() {
        //break each sample size up into groups of groupSize (8
default), then test
        //if we get EVEN ONE positive test in the sample we need to
 break that group of 8 into 2 groups of 4
        //test each group and if there is EVEN ONE positive test
within that group then we then
        //test each one of those 4 people and we tally up the total
 amount of tests needed to get through the entire group
        //NOTE: while we may know which person is sick this is a
simulation so we have to abide by these rules to get an
accurate representation
        //of how many tests are needed as there is no magic person
node that marks people as sick or not sick that you can just
summon up.

        double numberTestGroups = Math.ceil((float) peopleList.size
()/groupSize);

        //here I split up our list of people into groups of the
specified group size and print them out using parts
        //parts represents a list of list, each sublist being a
test group

        /* PLAN :
        IF Infection found
            split into two list
            IF one group shows infection AND the other does not
                test all members of the infect group and the other
group is clear
            ELSE both groups show infection
                test all members of both groups
        ELSE
            done with 1 test
        */

        System.out.println("--Splitting Group into " +
numberTestGroups + "--");
        List<List<Person>> parts = chopped(peopleList.getList(),
groupSize); //set groupSize above
        for (List<Person> list : parts) {
            if(peopleList.size() <= 1000){System.out.print(" [");}

            for (Person person : list) {
                if(peopleList.size() <= 1000){ System.out.print(" "
 + person + ",");} //too many console print statements if doing
 anything bigger than 1000

                //Compute if they are sick, then we send that to
another method where they are split down and tested and our
test counter will be
                //incremented.
```

```
234                 if(person.getIsSick() == 1){
235                    List<List<Person>> splitList = new ArrayList<>()
    ;
236                    splitList = split(list);
237
238                    for(int i = 0; i < splitList.size(); i++){
239                         performTest(); //test the two split lists
240                    }//for
241
242                    for (List<Person> listSubGroup : splitList){
243                        for(Person personSubGroup : listSubGroup){
244                             if(personSubGroup.getIsSick() == 1){
245                                 //test all of the people in that
    subgroup group for each subgroup that is found to have a sick
    person
246                                 for(int i = 0; i < listSubGroup.
    size(); i++){
247                                     performTest();
248                                 }
249                             }
250                        }//for
251                    }//for
252                } //if
253            } //for
254            if(peopleList.size() <= 1000){System.out.println("] ")
    ;}
255            performTest();
256        } //for
257
258
259    }//test
260
261     // chops a list into non-view sublists of length L, these
    represent our testing groups since we are passing it our List
    of person objects and the number that we want the list to be
    chopped into
262    public static <T> List<List<T>> chopped(List<T> list, final int
     L) {
263        List<List<T>> parts = new ArrayList<List<T>>();
264        final int N = list.size();
265        for (int i = 0; i < N; i += L) {
266            parts.add(new ArrayList<T>(list.subList(i, Math.min(N,
    i + L))));
267        }//for
268        return parts;
269    }//chopped
270
271    // Generic function to split a list into two sublists in Java
272    public static <T> List<List<T>> split(List<T> list)
273    {
274        List<List<T>> splitInTwo = new ArrayList<List<T>>();
275
276        // get size of the list
277        int size = list.size();
278
279        // construct new list from the returned view by list.
    subList() method
```

```
280         List<T> first = new ArrayList<>(list.subList(0, (size + 1)
    /2));
281         List<T> second = new ArrayList<>(list.subList((size + 1)/2,
     size));
282 
283         // return a List array to accommodate both lists
284         splitInTwo.add(first);
285         splitInTwo.add(second);
286 
287         return splitInTwo;
288     }
289 
290     //simply keeps track of how many tests have been given
291     public static void performTest(){
292         numTests++;
293     }//performTest
294 
295     public static int menu() {
296 
297         int selection;
298         Scanner input = new Scanner(System.in);
299 
300         /**************************************************/
301 
302         System.out.println("\nChoose the AMOUNT OF PEOPLE the
    Simulation will be Run With :");
303         System.out.println("1 - 100");
304         System.out.println("2 - 1,000");
305         System.out.println("3 - 10,000");
306         System.out.println("4 - 100,000");
307         System.out.println("5 - 1,000,000");
308         System.out.println("6 - Change Testing Group Size");
309         System.out.println("7 - Run with Default Settings");
310         System.out.println("8 - Quit");
311 
312         while (!input.hasNextInt()) {
313             String scanner = input.next();
314             System.out.print(" '" + scanner + "' is not a valid
    number.\n");
315         } // while
316 
317         selection = input.nextInt();
318         if(selection == 8){input.close();}
319         if(selection == 7){selection = 2; groupSize = 8;}
320 
321         return selection;
322     }// menu
323 
324     //if time allows can adjust to work with different sized groups
     other than 8 may come back and implement later
325     public static void menu2() {
326 
327         int selection;
328         Scanner input2 = new Scanner(System.in);
329 
330         /**************************************************/
331 
```

```
332         System.out.println("\nChoose the GROUP SIZE the Simulation
    will be Run With :");
333         System.out.println("1 - 4 Groups");
334         System.out.println("2 - 6 Groups");
335         System.out.println("3 - 8 Groups");
336         System.out.println("4 - 16 Groups");
337         System.out.println("5 - 32 Groups");
338         System.out.println("6 - Quit");
339 
340 
341         while (!input2.hasNextInt()) {
342             String scanner = input2.next();
343             System.out.print(" '" + scanner + "' is not a valid
    number.\n");
344         } // while
345 
346         selection = input2.nextInt();
347 
348         //Processing choice here as it is a submenu and want to
    keep main cleaner and for our main menu
349         if(selection == 1) {
350             groupSize = 4;
351         }
352         if(selection == 2) {
353             groupSize = 6;
354         }
355         if(selection == 3) {
356             groupSize = 8;
357         }
358         if(selection == 4) {
359             groupSize = 16;
360         }
361         if(selection == 5) {
362             groupSize = 32;
363         }
364         if(selection == 6) {
365             System.out.println("Group Size Remains the Same");
366             //input2.close();
367         }
368 
369         System.out.println("Rerunning Simulation Program with new
    group size");
370 
371 
372     }
373 
374 }//SemesterProjectMain
```

3 ListPeople Class

```
1  import java.util.ArrayList;
2  import java.util.List;
3  import java.util.Random;
4
5  public class ListPeople {
6
7      //int person = 0;
8      private List<Person> people = new ArrayList<>();
9
10     public ListPeople(List<Person> listPeople){
11         this.people = listPeople;
12     }//ListPeople
13
14     //Clears the list and adds adds however many people given to
       the list, so if 100 have a list of 100 people
15     public void addPeople(int sizeWanted){
16         this.people.clear();
17         for(int i = 0; i < sizeWanted; i++){
18             Person person = new Person(0);
19             people.add(person);
20         }//for
21     }//addPerson
22
23     //Here I have set the infection rate to 2 by generating a
       random number btw 0-100 and if that number is less than 2
24     //then the person has covid dun dun dunnnnn hope he is
       quarantining
25     public void diseaseGiver(int infectionRate){
26         for(int i = 0; i < people.size(); i++){
27             Random r = new Random();
28             int percentChanceGetSick = r.nextInt(100);
29             if (percentChanceGetSick < infectionRate){
30                 people.get(i).setIsSick(1);
31             }
32
33             //return testResult.nextInt((1-0)+0);
34
35         }
36     }//test
37
38     public List<Person> getList(){
39         return people;
40     }//getList
41
42     public void clearList(){
43         this.people.clear();
44     }//clear
45
46     public int size(){
47         return people.size();
48     }//size
49
50     public Person get(int index){
```

```
51          return people.get(index);
52      }
53
54      public void print(){
55          if(!people.isEmpty()){
56              for(int i = 0; i < people.size(); i++)
57                  System.out.println(people.get(i));
58          }//if
59
60      }
61
62  }//ListPeole
```

4 Person Class

```
public class Person {

    int isSick = 0; //0 represents not sick, 1 represents sick

    public Person(int isSick) {
        this.isSick = isSick;
    }//person constructor

        public int getIsSick(){
            return this.isSick;
        }//getter

        public void setIsSick(int maybeGotCovid){
            this.isSick = maybeGotCovid;
        }//setter

        public String toString(){
            String s = Integer.toString(this.isSick);
                return (s);
        }



}//person
```