COM361 Assignment 2
Qing (Cecilia) Lu
Student ID: 300363602

1 Question 1 - Gerrymandering

1.1 Give an algorithm to determine whether a given set of precincts is susceptible to gerrymandering; the running time of your algorithm should be polynomial in n and m.

I took the dynamic programming approach to address the gerrymandering problem. In this report, the party that holds the majority is named $party\ L$. The gerrymandering offers me n precincts, and each of which contains m voters. Hence, the total number of voters is n*m, i.e. nm. We need to divide n precincts into two districts – each has $\frac{n}{2}$ of the precincts and $\frac{nm}{2}$ of the voters. Therefore, to win a district, a party must have at least $\frac{nm}{4}$+1 voters in that district. Let t be the total number of Party-L voters in all precincts. In order to make the party L hold the majority in each district, t has to be at least ($\frac{nm}{4}$+1)x2=$\frac{nm}{2}$+2. If party L holds a majority in both districts, the number of voters who vote party L in $district_1$ is $d_{1.L}$ = $num \geq \frac{nm}{4}$+1; and the number of voters who vote party L in $district_2$ is $d_{2.L}$ = $t - num \geq \frac{nm}{4}$+1. Putting the two inequalities together, we can get that half of all precincts is susceptible to gerrymandering if there is $\frac{n}{2}$ precincts with num party-L voters in one district such that:

$$\left.\begin{aligned} num &\geq \frac{nm}{4}+1 \\ t-num &\geq \frac{nm}{4}+1 \end{aligned}\right\} \Rightarrow \frac{nm}{4}+1 \leq num \leq t-\frac{nm}{4}-1 \tag{1}$$

I solved the gerrymandering problem in two steps. The first step is trying to find all possible districts that consists of $\frac{n}{2}$ precincts (let's name the districts subsets Ss). The second step is trying to find one subset(S) among Ss: the number of party L voters num in S satisfies inequality 1. (Note: we suppose the party who holds the majority is $party\ L$, which contains at least $\frac{nm}{2}$ + 2 voters.)

In the first step , let's think about the nth precinct (P_n): the number of party-L voters in P_n is num_n. Now I have two choices: either include P_n in our district or not. If we include P_n in our district (when the precinct division is susceptible to gerrymandering): our district contains $\frac{n}{2}$-1 precincts (picked from P_1, P_2, ..., P_{n-1}) with num-$num_{n.L}$ party-L voters (num is the number of party-L voters in our district, including P_n). In another way, if we don't include P_n in our district (when the precinct division is susceptible to gerrymandering): our district contains $\frac{n}{2}$ precincts (picked from P_1, P_2, ..., P_{n-1}) with num party-L voters. Therefore, we can summarize this recursive relationship in Equation 2, where G(i, j, num) represents the question: "is it possible to form a set of j precincts (picked from the first i precincts) with num total party-L voters ?".

$$G(i,j,num)\begin{cases} true: \ if\ G(i-1,\ j-1,\ num-num_i)\ is\ true \\ true: \ if\ G(i-1,\ j,\ num)\ is\ true \\ true: \ if\ j=0, num=0 \\ true: \ if\ i=1,\ j=1,\ num=num_k \\ false: \ otherwise \end{cases} \tag{2}$$

Equation 2 shows the recurrence. The first line displays the case that we include P_i into our district. The second line is the case when we do not include P_i into our district. The third line demonstrates that we can always have a subset that has 0 precinct and 0 party-L voters. The fourth line shows the case that we form a set of 1 precinct (j = 1) from the $1st$ precinct (i = 1), (in other word, this only precinct is the first precinct P_1), and this set of precinct has num party L voters, where num is the number of party-L voters in P_1. If none of the four cases is satisfied, it is impossible to gerrymander, which is shown by $false$ in the last line.

Algorithm 1: Gerrymandering: divide a set of precincts into two districts, each consisting of n/2 precincts; see if the same party holds the majority in both districts.

```
Input: n: number of precincts; m: number of voters in each precinct;
       P: a set of precincts; $num_L$: total number of party-L voters;
Output: a boolean value: true if it is susceptible to gerrymandering; otherwise, false.
1  Address the step one:
2  if $num_L < \frac{nm}{2}+2$ OR $num_{1.L} > t - \frac{nm}{4} - 1$ then
3      return false
4  end
5  for $i \in$ (1 to n) do
6      set G[i][0][0] ← true
7      other untouched elements ← default value $false$
8  end
9  set G[1][1][$num_{1.L}$] ← true
10 for $i \in$ (2 to n) do
11     for $j \in$ (1 to $\frac{n}{2}$) do
12         for $num \in$ (1 to $num_L$ - $\frac{nm}{4}$ -1) do
13             if G[i-1][j-1][num-$num_{i.L}$]=true then
14                 G[$i$][$j$][$num$] ← true
15             end
16             if G[i-1][j][num]=true then
17                 G[$i$][$j$][$num$] ← true
18             end
19         end
20     end
21 end
22 Address the step two:
23 for $num \in$ ($\frac{nm}{4}$+1 to $num_L$ - $\frac{nm}{4}$ -1) do
24     if G[n][$\frac{n}{2}$][num]= true then
25         return true
26     end
27 end
28 return false
```

Based on the recursive relationship shown in Equation 2, the algorithm builds a three dimensional table G[n][$\frac{n}{2}$][num_L - $\frac{nm}{4}$ -1]. I used i, j, and num to create a triple loop to fill the table. The range of i is from 1 to n, because we can include any number of possible precincts from all of the n precincts. The range of j is from 1 to $\frac{n}{2}$, as both districts have $\frac{n}{2}$ precincts eventually. The range of num is from 1 to $t - \frac{nm}{4} - 1$, as this is the maximum number of party-L voters in each district if the set of precincts is susceptible to gerrymandering (shown in Equation 1).

Each entry of the table stores a boolean value, which demonstrates whether there is any combination of precincts in the way that there are num total party-L voters in j precincts picked from the first i precincts in the precinct list. If yes, the entry stores $true$; otherwise, it stores $false$.

After the table has been built, we start to address the second step. As discussed above, if the system is susceptible to gerrymandering (i.e., the party L holds the majority in both districts), the number of party-L voters in our district must be between ($\frac{nm}{4}$+1) and (num_L - $\frac{nm}{4}$ -1). Therefore, we need to go over the entries that are in the last row (i = n), with j = $\frac{n}{2}$, and $num \in$ ($\frac{nm}{4}$+1 .... num_L - $\frac{nm}{4}$ -1). If any of these entries stores $true$, the given precincts are susceptible to gerrymandering. The pseudocode is provided in Algorithm 3.

1.2 Using the proof techniques (for Dynamic Programming Algorithms) discussed in the lectures, provide a proof that your algorithm is correct.

1.2.1 Input:

1. **n**: number of precincts;
2. **m**: number of voters in each precinct;
3. **P**: a set of precincts;
4. $\boldsymbol{num_L}$: total number of party-L voters;

1.2.2 Output:

An n *$\frac{n}{2}$ * (num_L - $\frac{nm}{4}$ -1) array G[i][j][num] ($i \in$ [1,n], $j \in$ [1,$\frac{n}{2}$], $num \in$ [1, num_L - $\frac{nm}{4}$ -1]). Each entry in the three dimensional array stores the answer to the corresponding G[i][j][num] question – is it possible to form a set of j precincts from the first i precincts that has exactly num total party-L voters? If the entry stores $true$, the answer is yes; otherwise, the possibility is ”no”.

1.2.3 Theorem/Assumption:

The number of precincts is an even integer (n can be divided by 2), which enables us to divide the precincts into 2 districts with the same amount of precincts.

G[i][j][num] is true if and only if it is possible to include j precincts into our district from the first i precincts, and meanwhile, num is equal to the sum of winning party's voters in our district (district that consists of the j precincts). If there exists any G[n][$\frac{n}{2}$][num] = $true$ ($num \in$ [$\frac{nm}{4}$+1, ..., num_L - $\frac{nm}{4}$ -1]), the precincts are susceptible to gerrymandering.

1.2.4 Requirement:

Build the three dimensional table G, and check if there is any G[i][j][num] = $true$ when i = n, j = $\frac{n}{2}$, $num \in$ [$\frac{nm}{4}$+1, ..., num_L - $\frac{nm}{4}$ -1].

1.2.5 Proof:

By induction on G[i][j][num], $i \in (1,n)$ and $j \in (1,\frac{n}{2})$

Base case:

If there is no precincts in the array, simply print ”There is no precinct.” and return.

As the amount of precincts is an even number, it is meaningless to consider gerrymandering in the case of $singleton$.

When there are two precincts (n = 2, i = 2, j = 1), put one precinct in each district. In this case, simply compare party-L and party-N voters in each district. If one party holds the majority in both districts, the precincts is susceptible to gerrymandering; otherwise, it is not.

Induction hypothesis:

G[i][j][num] is true when it is possible to include j precincts into our district from the first i precincts, and meanwhile, num is equal to the sum of winning party's voters in our district (district that consists of the j precincts).

If G[$i-1$][$j-1$][num-$num_{i.L}$] is $true$, or G[$i-1$][j][num] is $true$, G[i][j][num] is true. Besides, G[i][0][0] ($i \in$ [1, ..., n]) (among the first i precincts, pick 0 precinct in our district, and there is no voters in this district)and G[1][1][$num_{1.L}$] (pick the first precinct from the array and assign it into our district) are $true$.

Induction step: prove the induction hypothesis also suits n+1

1. Correctness of G[i][0][0] = $true$: None of the first i precincts is assigned to our district. Therefore, the number of winning-party voters in our district is 0. $\leftarrow$ Proved

2. Correctness of G[1][1][$num_{1.L}$] = $true$: Assign the first precinct into our district, and the number of winning-party voters in our district equals the number of winning-party voters in the first precinct $num_{1.L}$. Suppose G[1][1][$num_{1.L}$] is $false$, the $num_{1.L}$ doesn't equal the number of winning-party voters in the first precinct, which is a contradiction. $\leftarrow$ Proved

3. Suppose G[i][j][num] ($i \in$ (1, ..., n-1), $j \in$ (1, ..., $\frac{n}{2}$-1), $num \geq$0) is true (induction hypothesis). Now we need to assign the (i+1)th precinct. There are two way of doing it: assign the (i+1)th to our district, or to another one.

   a. Situation one: if we include the $(i+1)th$ precinct into our district, the entry G[i+1] [$\frac{j}{2}$+1] [num_L+$num_{(i+1).L}$] will be $true$ in the table. Suppose G[i+1] [$\frac{j}{2}$+1] [num_L+$num_{(i+1).L}$] were $false$, this means it is impossible to form a set of precincts in our district with num_L+$num_{(i+1).L}$ winning-party voters. However, it is obvious that adding one more precinct makes the number of winning-parting voters in our district to be num_L+$num_{(i+1).L}$. Therefore, this entry should be $true$. This proves the supposed case is a contradiction. $\leftarrow$ Proved

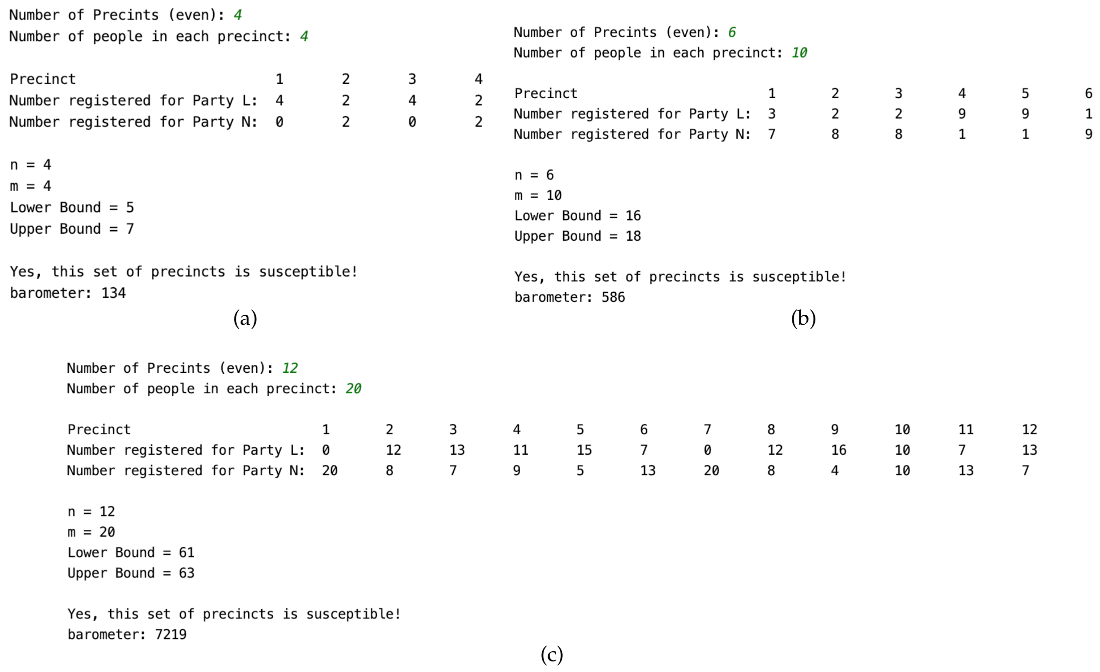
   b. Situation two: if we include the $(i+1)th$ precinct into the other district, the entry G[i+1] [$\frac{j}{2}$] [num_L] will be $true$ in the table. This means: among the first $i+1$ precincts, the precincts in our district remains the same after the $(i+1)th$ precinct's assignment. Suppose G[i+1] [$\frac{j}{2}$] [num_L] were $false$, it is impossible to form a set of precincts in our district with num_L winning-party voters. But as G[i][j][num] is $true$, and precincts remains the same, the first i precinct can form a district with num_L winning-party voters, which is a contradiction. $\leftarrow$ Proved

In sum, the proposed induction hypothesis is true in the case of (i+1)th precincts in both possible situations ($true$).

4. As discussed in part 1, the number of winning-party voters in our district must be between $\frac{nm}{4}+1$ and $t-\frac{nm}{4}-1$. Hence, we only need to look at the entries in this range when we get two districts with same amount of precincts. If there is any entry with $true$, the system is susceptible to gerrymandering. If we look at the entries out of the range, no matter whether it is $true$ or $false$, we didn't take all $\frac{n}{2}$ precincts in our district into consider. Therefore, the step two mentioned in part 1 is proved correct. $\leftarrow$ Proved

1.3 Write a simple test case generator and generate a reasonable set of test cases for your algorithm to be used to evaluate its asymptotic complexity. Use a barometer or similar technique to plot the execution times for different numbers of input sizes and state the expected complexity of your algorithm.

In my Java program, the test case generator function is named $generatePrecincts$ in the class $Gerrymandering$. This generator receives the number of precincts (n) and the amount of voters in each precincts (m) entered by the user, and generates a list of precincts with random integers (between 0 and m) for party-L voters (the rest of voters in the precinct belongs to party N). All of the samples are tested correct. Fig. 1 shows some case examples.



```
Number of Precints (even): 4
Number of people in each precinct: 4

Precinct                        1       2       3       4
Number registered for Party L:  4       2       4       2
Number registered for Party N:  0       2       0       2

n = 4
m = 4
Lower Bound = 5
Upper Bound = 7

Yes, this set of precincts is susceptible!
barometer: 134
```

(a)

```
Number of Precints (even): 6
Number of people in each precinct: 10

Precinct                        1       2       3       4       5       6
Number registered for Party L:  3       2       2       9       9       1
Number registered for Party N:  7       8       8       1       1       9

n = 6
m = 10
Lower Bound = 16
Upper Bound = 18

Yes, this set of precincts is susceptible!
barometer: 586
```

(b)

```
Number of Precints (even): 12
Number of people in each precinct: 20

Precinct                        1       2       3       4       5       6       7       8       9       10      11      12
Number registered for Party L:  0       12      13      11      15      7       0       12      16      10      7       13
Number registered for Party N:  20      8       7       9       5       13      20      8       4       10      13      7

n = 12
m = 20
Lower Bound = 61
Upper Bound = 63

Yes, this set of precincts is susceptible!
barometer: 7219
```

(c)

Figure 1: Test cases with the barometer of the proposed algorithm

The barometer counts two parts: (1) initialising each entry in the table G, and (2)searching entries that equals the number of winning-party voters in a set of precincts.

Table 1: Table for barometer results from testing. n is the precinct number, m is the number of voters in each precinct.

n	6	12	18	24	30
m	10	20	30	40	50
barometer	586	7219	38688	155832	277046
n	36	42	48	54	60
m	60	70	80	90	100
barometer	662338	1041697	1977518	3157510	4627978

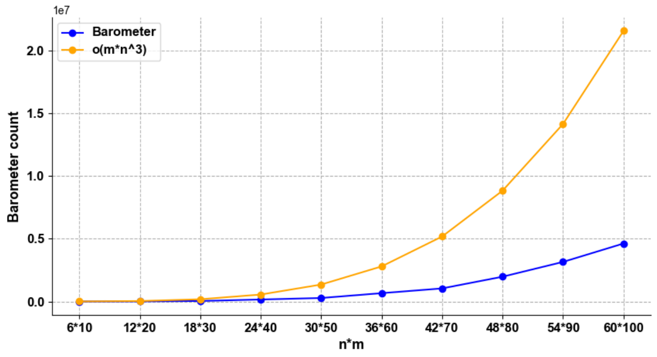


Figure 2: This line graph shows the relationship between execution times and different n and m. The blue line shows the barometer's estimated complexity; and the orange line demonstrates the theorem complexity $O(n^3m)$.

Table 1 shows the relationship between n, m, and the barometer. Fig. 2 shows the relationship between execution times and different n and m. These results imply that the complexity of my algorithm is expected to be less than $O(n$*$\frac{n}{2}$*$(t$-$\frac{nm}{4}$-1)), which can be simplified as $O(n^3m)$.

1.4 Using the complexity analysis arguments similar to those used in the lecture show and prove the complexity of your algorithm and discuss if it does and or does not match the result you obtained using a barometer.

The proposed algorithm identifies whether a given set of precincts is susceptible to gerrymandering. Although the problem has two parties, I only need to check the party that holds the majority in total – the party has at least $\frac{nm}{2}$ + 2 voters. Thus, I apply my algorithm only once. Then I can identify whether the given precincts are susceptible.

I build an n*n*($num_{winparty}$-$\frac{nm}{4}$-1) table. The space used is $\theta(n^3m)$.

Firstly, the program checks whether winning-party voters holds the majority among all voters (the $2nd$ line in pseudocode on page 2). As this can be done for only one time, the complexity of it is $O(1)$.

Secondly, we need to go through the 3-dimensional table in order to initialise all of the entries in the table. There are 3 nested loops for doing so: the outer loop is done for n times; the first inner loop is done for $\frac{n}{2}$ times for each i ($i \in$[1, ..., n]); the second inner loop is done for $t - \frac{nm}{4} - 1$

times for each j (j ∈[1, ..., $\frac{n}{2}$]). Therefore, the complexity of this part is: $O(n$*$\frac{n}{2}$*(t-$\frac{nm}{4}$-1)).

Thirdly, we need to search the entries that match the number of winning-party voters when selecting j from the first i precincts. In this case, we also go through the 3-dimensional table, but not all of the entries will be changed from $false$ to $true$.

1. The outer loop is in the 10th line in the pseudocode – $O(n)$.
2. The first inner loop is in the 11th line in the pseudocode – $O(n * \frac{n}{2})$.
3. The most inner loop is from line 12 to 19 in the pseudocode – $O(n$*$\frac{n}{2}$*(t-$\frac{nm}{4}$-1)).

Therefore, the complexity of the proposed algorithm is n^3m+n^3m+$nm \in \theta(n^3m)$.

$$\begin{aligned}
&n * \frac{n}{2} * (num_{target} - \frac{nm}{4} - 1) \\
&= \frac{n^2}{2} + \frac{n^3m}{8} - \frac{n^2}{2} \\
&\in O(n^3m)
\end{aligned} \tag{3}$$

As shown in Fig. 2, the analysed complexity does not perfectly match the result I obtained using a barometer. This is because the analysis process gives an approximate complexity of the worst case. However, regarding the second line of equation 3 shown above, the realistic complexity is expected to be lower than the $\theta(n^3m)$, as the coefficient $\frac{1}{8}$ is removed. This claim is supported by Fig. 2.