Question 1)

a) **In this section I try to write the pseudocode to explain the logic of algorithm by following the style in lesson slides.**

- **CRCW**

```
function MatrixMultiplicationCRCW(Matrix_1[N][N],Matrix_2[N][N])
Model: CRCW PRAM with p=n processors
Input: Matrix_1[N][N], Matrix_2[N][N] (2 matrixes with size N by N)
Output: Result[N][N] (a matrix called with size N by N)

for i=0 to N-1 do
    for j=0 to N-1 do
        for k=0 to N-1 do in parallel
            Result[i][j] =Result[i][j]+Matrix_1[i][k]*Matrix_2[k][j]
        end
    end
end
return Result
```

- **EREW**

```
function MatrixMultiplicationEREW(Matrix_1[N][N],Matrix_2[N][N])
Model: EREW PRAM with p=n processors
Input: Matrix_1[N][N], Matrix_2[N][N] (2 matrixes with size N by N)
Output: Result[N][N] (a matrix called with size N by N)

for i=0 to N-1 do
    for j=0 to N-1 do
        for k=0 to N-1 do in parallel
            Result[i][(j+k)%N]=Result[i][j]+Matrix_1[i][k]*Matrix_2[k][(j+k)%N]
        end
    end
end
return Result
```

- **2-dimensional mesh $M_{n,n}$**

function 2_D_Meshes_Matrix_Multiplications (Matrix_1[N][N], Matrix_2[N][N])
Model: 2-Dimentional Mesh Mnn with p=n*n processors
Input: Matrix_1[N][N], Matrix_2[N][N] (2 matrices with size N by N)
Output: Result[N][N] (a matrix called with size N by N)

```
for a=1 to N-1 do in parallel
    for b=0 to N-1 do in parallel
        for a2=0 to a do
            if b-1==-1 do
                for b2=0 to N-1 do
                    Matrix_1[a][b+1].temp=Matrix_1[a][b]
                end
                Matrix_1[a][b]=Matrix_1[a][b].temp
            end
            Matrix_1[a][b-1]=Matrix_1[a][b]
        end
    end
end


for a=0 to N-1 do in parallel
    for b=1 to N-1 do in parallel
        for b2=0 to a do
            if a-1==-1 do
                for a2=0 to N-1 do
                    Matrix_1[a+1][b].temp=Matrix_1[a][b]
                end
                Matrix_1[a][b]=Matrix_1[a][b].temp
            end
            Matrix_1[a-1][b]=Matrix_1[a][b]
        end
    end
end


for k=0 to N-1
    for i=0 to N-1 do in parallel
        for j=0 to N-1 do in parallel
            Result[i][j]=Result+Matrix_1[i][j]*Matrix_1[i][j]
        end
    end
    for a=0 to N-1 do in parallel
        for b=0 to N-1 do in parallel
            if b-1==-1 do
                for b2=0 to N-1 do
                    Matrix_1[a][b+1].temp=Matrix_1[a][b]
                end
                Matrix_1[a][b]=Matrix_1[a][b].temp
            end
            Matrix_1[a][b-1]=Matrix_1[a][b]
        end
    end
```

```
    for a=0 to N-1 do in parallel
        for b=0 to N-1 do in parallel
            if a-1==-1 do
                for a2=0 to N-1 do
                    Matrix_1[a+1][b].temp=Matrix_1[a][b]
                end
                Matrix_1[a][b]=Matrix_1[a][b].temp
            end
            Matrix_1[a-1][b]=Matrix_1[a][b]
        end
    end
end
return Result
```

b) Below you can see the execution times of each agorithm compered to actual basic sequential algorithm.

Algorithm Type	Input Size = 1	Input Size = 2	Input Size = 3	Input Size = 4	Input Size = 5	Input Size = 6	Input Size = 7	Input Size = 8	Input Size = 9	Input Size = 10	Input Size = 11	Input Size = 12	Input Size = 13	Input Size = 14	Input Size = 15
Sequential Algorithm	1	64	729	4096	15625	46656	117649	262144	531441	1000000	1771561	2985984	4826809	7529536	11390625
CRCW Algorithm	1	16	81	256	625	1296	2401	4096	6561	10000	14641	20736	28561	38416	50625
EREW Algorithm	1	16	81	256	625	1296	2401	4096	6561	10000	14641	20736	28561	38416	50625
2-D Meshes Algorithm	4	172	2052	11824	45700	137484	348292	778432	1581444	2980300	5285764	8916912	14423812	22512364	34071300

The table contains only input sizes between 1 and 15. The more general version is in pdf file called table.pdf in this current directory. I only put here this version since in the general version numbers are not visible.

The all figure I have collect from the datas below and explain them one by one

1) **Matrix multiplication that the sequential algorithm used**

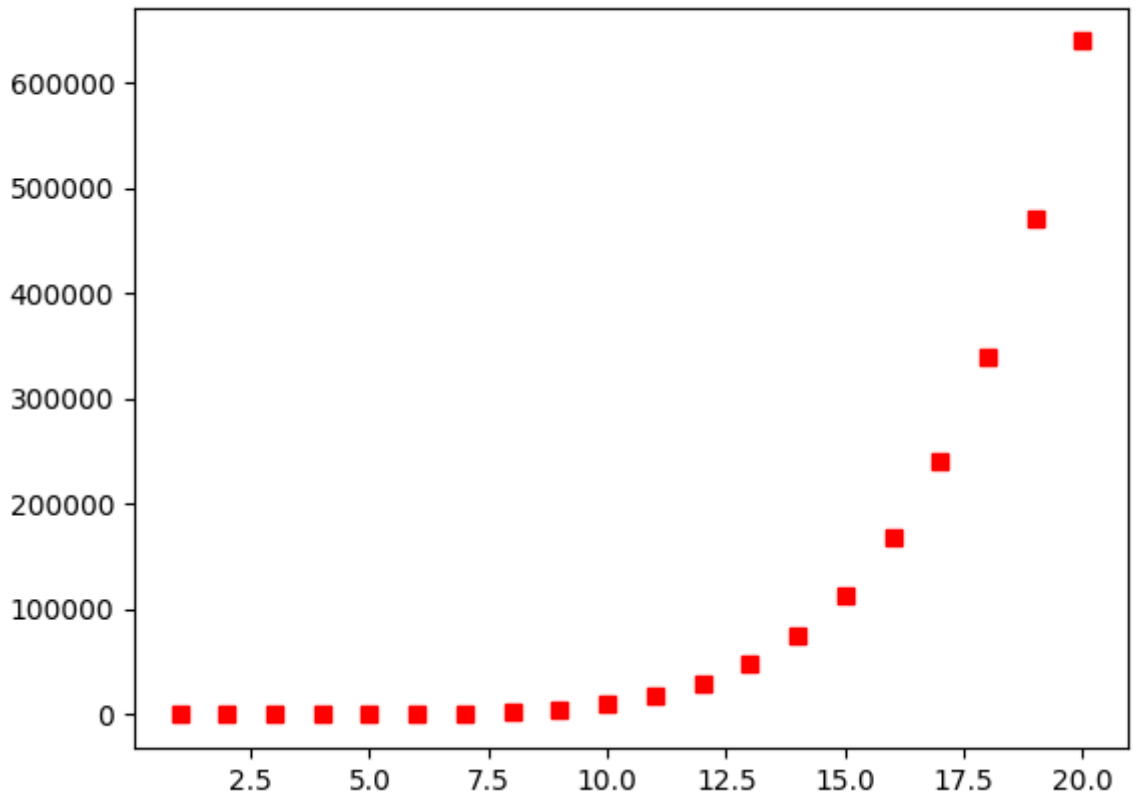


Figure 1

x axix is input sizes y axix is execution times divided by 100.

The sequential algorithm is $O(n^3)$ and the figure looks like as expected. The all the plots are distributed exponentially.

2) Matrix multiplication that the CRCW PRAM with n processors used

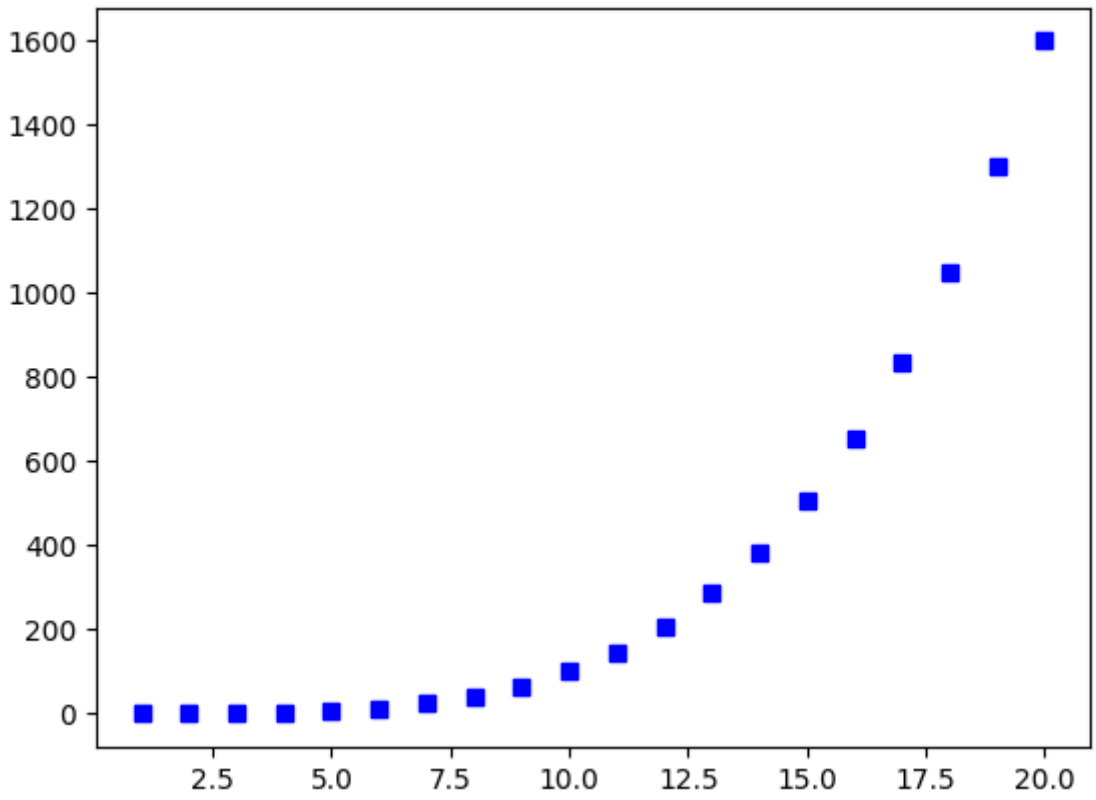


Figure 2

x axix is input sizes y axix is execution times divided by 100

In this approach we decrease the complexity from $O(n^3)$ to $O(n^2)$ since the last for loop is callculated in just one step so we think that we only iterate just 2 for loop so the paralel algorithm's complexity is $O(n^2)$ but we cannot see the diffreneces by just looking at the figure in part a and this figure. But If we look at the y axixs we can see the huge diffreneces. The y axixs in part a is going to from 0 to 600000 however here it goes to from 0 to 1600 which is very very small.

3) Matrix multiplication that the EREW PRAM with n processors used

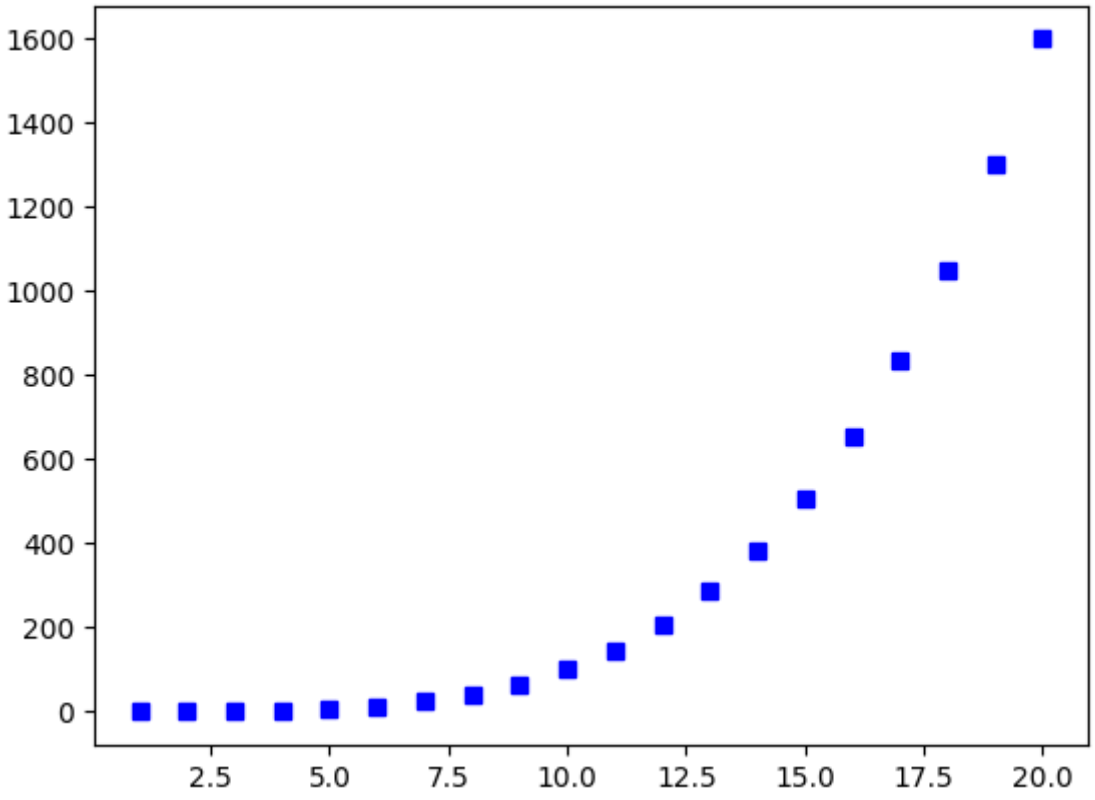


Figure 3

x axix is input sizes y axix is execution times divided by 100.

For this algorithm everyone can expect there is a higher complexity than the CRCW PRAM model. They have a point but most probably for this problem we have the same complexity as part b which is $O(n^2)$ since just like CRCW the last for loop is callculated in just one step so we think that we only iterate just 2 for loop so the paralel algorithm's complexity is $O(n^2)$. The figure and table above clearly support us.

4) Matrix multiplication that the 2-dimentional mesh M(n,n) used

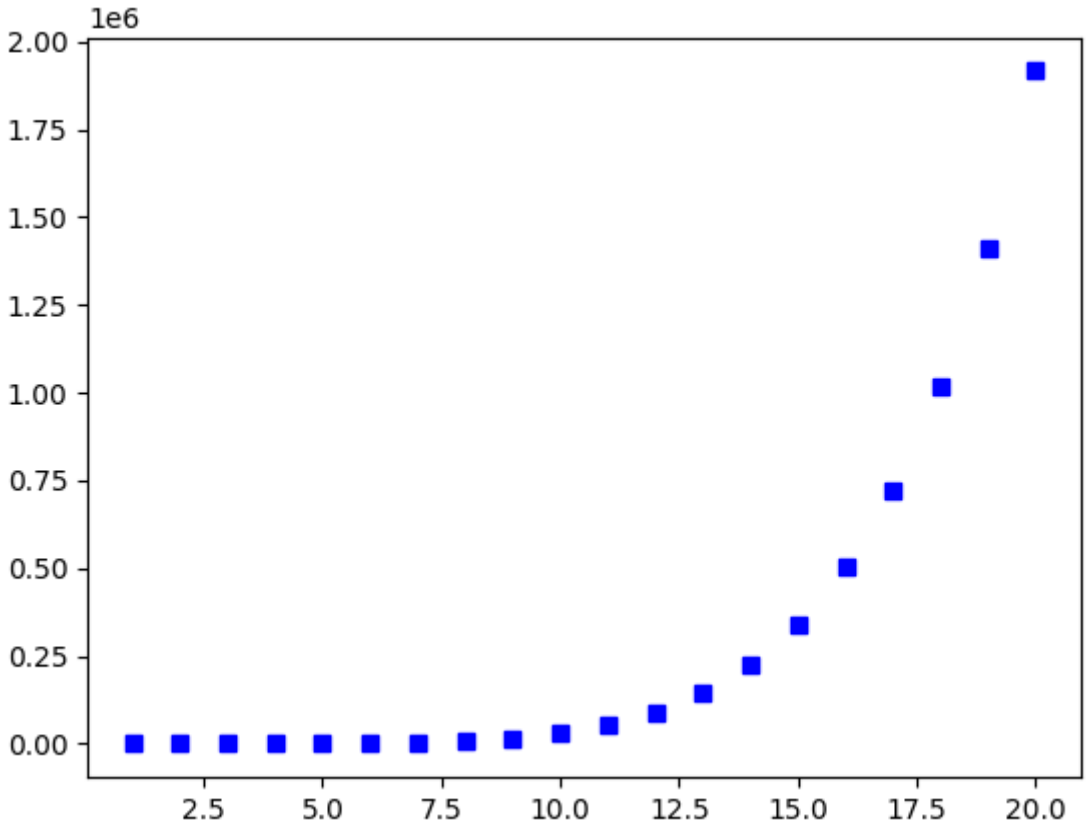


Figure 4

In this approach we fail to have less time complity than the actual basic sequential algorithm since the meshs are not good at multiplying the matrixs since we need to execute a circular roll operation and if we have on the edge of a mesh than there are n operation that we need to execute to roll from the most right or topest position to the most left or the most bottom direction and these roll operation takes a lot of times.

However I don't trust my algorithm to be best algorithm in parallel so meshes can be suitable to multiply matrices but in this problem I have tried to delevop as much as I do and it grows higher than the basic sequential algorithm.

Even though the mesh algorithm's complexity is O(n^2), the figure does not support me as you can see from figure 7. The sequential algorithms grows lover then this algorithm. Maybe I am doing something wrong or adding more count somewhere.

5) The comparation of the sequential algorithm and CRCW PRAM

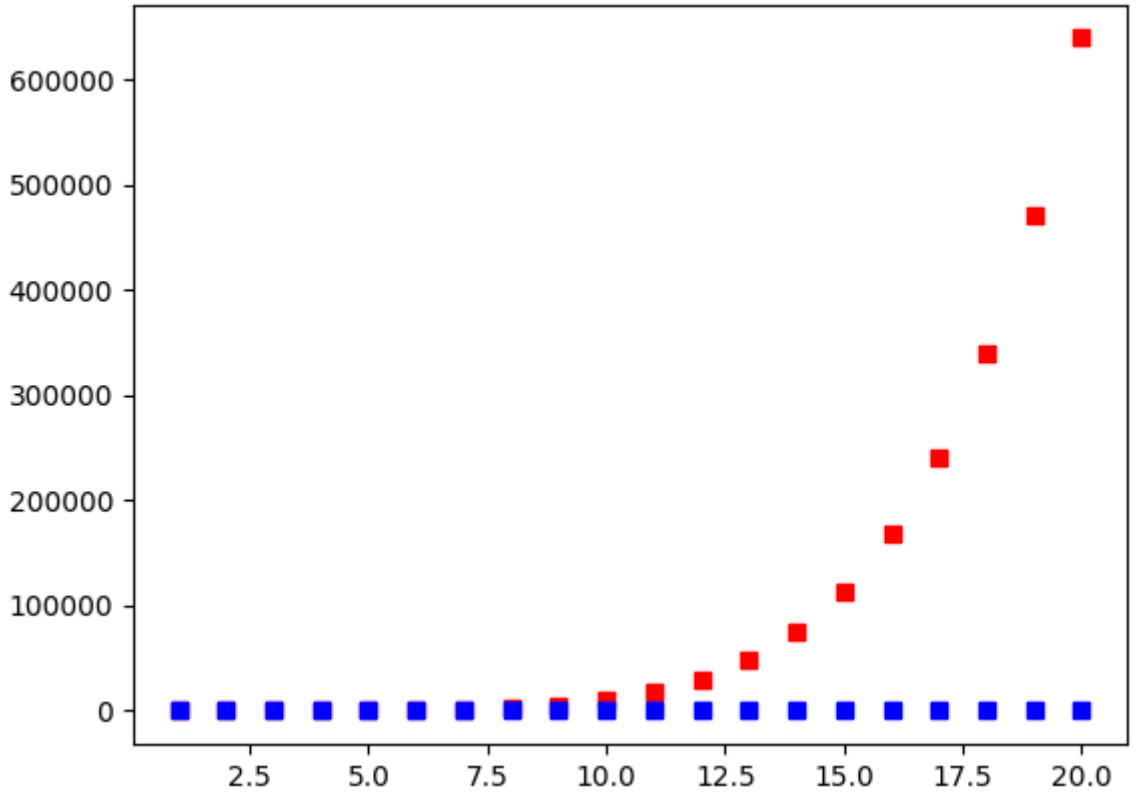


Figure 5

x axix is input sizes y axix is execution times divided by 100.

As we can see from the figure we can say the CRCW can perfectly suceed to decrease the time complexity. Until input 10 the two algorith grows the same but from that point it grows much more than CRCW PRAM so I would rather use CRCW PRAM algorithm than the basic sequential algorithm.

Over all we clearly say that we need to use CRCW PRAM instead sequential algorithm if we want to decrease the time complexity.

6) The comparation of the sequential algorithm and EREW PRAM

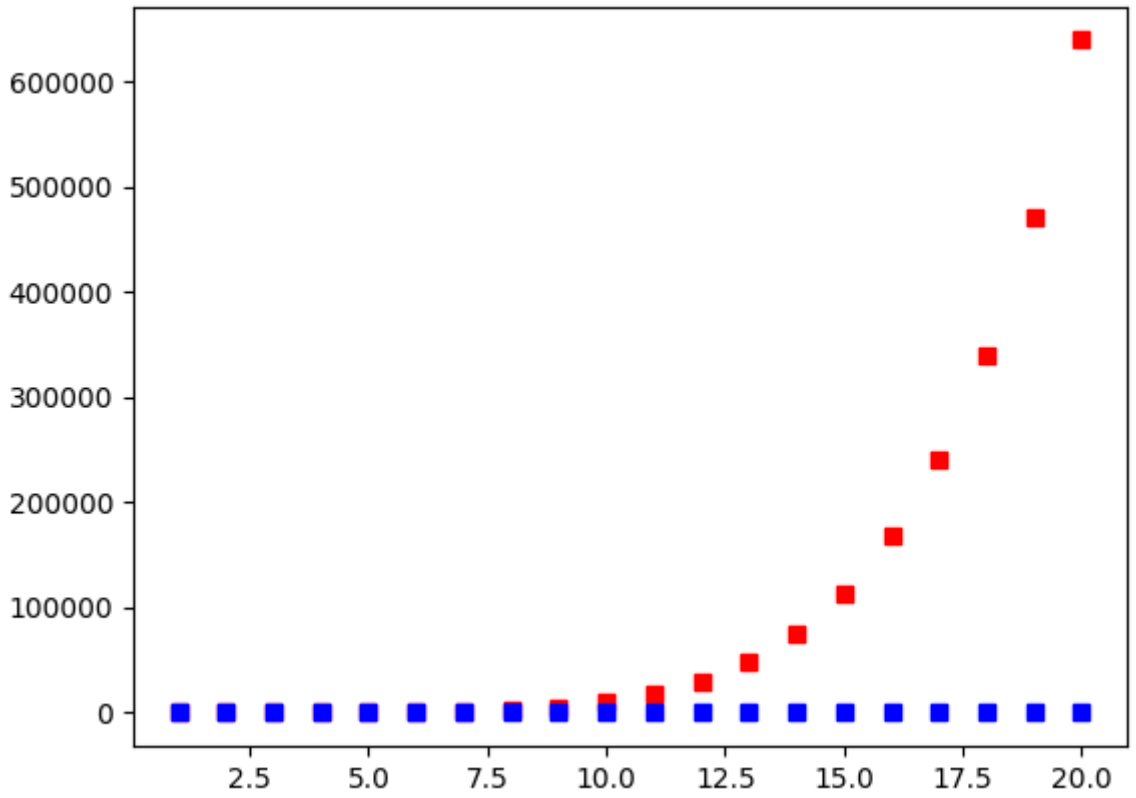


Figure 6

x axix is input sizes y axix is execution times divided by 100.

As we can from the figure just like the comparation figure above we see the same since EREW PRAM and CRCW PRAM have the same complexity.

Over all we clearly say that we need to use EREW PRAM instead sequential algorithm if we want to decrease the time complexity.

7) The comparation of the sequential algorithm and 2-dimensional mesh M(n,n)

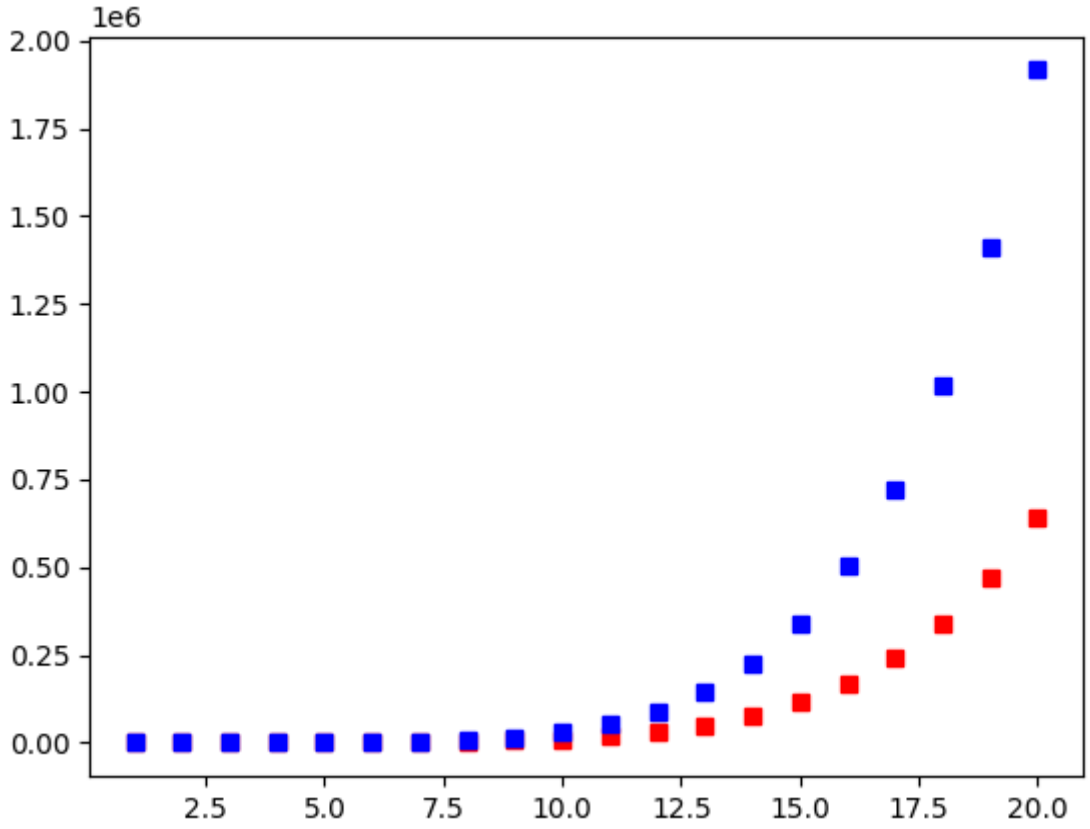


Figure 7

As we explain part 4 above the meshes does not fit this problem we have the same execution time until input 10 but from that point the algorithm that meshes used grows much more than the normal sequential algorithm.

Over all we cannot use mesh paralel algorithm instead basic sequential algorithm.

Question 2)

a) For different values let's write the all posible outputs

N=1, List = [50] the only output is ∅

N=2, List = [50,100] outputs = ∅,[1,2]

N=3, List = [50,100,150] outputs = ∅,[1,2],[1,3],[2,3],[1,2,3]

N=4, List = [50,100,150,200] outputs = ∅,[1,2],[1,3],[1,4],[2,3],[2,4],[3,4],[1,2,3],

[1,2,4],[2,3,4],[1,3,4],[1,2,3,4]

N=5, List = [50,100,150,200,250] outputs = ∅,

[1,2],[1,3],[1,4],[1,5]

[2,3],[2,4],[2,5],[3,4],[3,5],[4,5],

[1,2,3],[1,2,4],[1,2,5],[1,3,4],[1,3,5],[1,4,5],

[2,3,4],[2,3,5],[2,4,5],

[3,4,5]

[1,2,3,4],[1,2,3,5],[1,2,4,5],[1,3,4,5],[2,3,4,5],

[1,2,3,4,5]

b) If we can generalize the algorithm for a list with n item inside then we execute

$$\binom{n}{0} + \binom{n}{2} + \binom{n}{3} + \cdots . + \binom{n}{n} - \binom{n}{1} = 2^n - n$$

The worst case in general looks like n(n-1)/2 since we first take 1 as pivot and compare with 2,3,4,5 and does not find any duplicate and take 2 as pivot and compare 3,4,5 and does not find any duplicate again, then take 3 as pivot compare 4 and 5 again and lastly take 4 as pivot and compare with 5 and gives empty lists and it gives us n^2 complexity

However, if we look at course slides called Lower Bound Theory slide no 46, we need to take the log of $2^n - n$ to get the depth of the decision tree. Because from part a we found the max number of outputs and placed them in leaves and using the logic of decision trees we need to take log of 2^n and log of n.

If the all outputs are in leaves then we can find the worst case lower bound of this problem by calculating the depth of tree.

$$log(2^n) - log(n)\ gives\ us\ the\ depth\ of\ tree$$

$$As\ you\ can\ see \log(2^n)\ is\ n\ and \log(n)\ is \log n$$

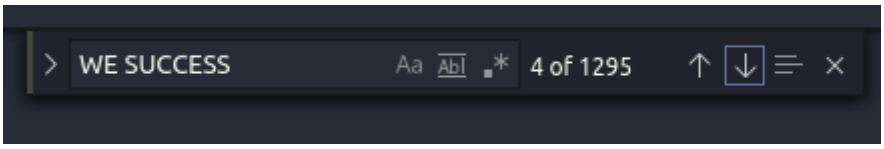
$$n - \log(n)\ \epsilon\ \Omega(n)$$

Question 3)

```
0.1258 , 0.1265 , 0.1278 , 0.1289 , 0.1281 , 0.1335 , 0.1354 , 0.1328 , 0.1262 , 0
.1208 , 0.1285 , 0.128 , 0.1272 , 0.1338 , 0.1292 , 0.1325 , 0.133 , 0.1326 , 0.13
01 , 0.1305 , 0.13 , 0.1316 , 0.1275 , 0.1273 , 0.1327 , 0.1314 , 0.1218 , 0.1331
, 0.1302 , 0.1307 , 0.1246 , 0.1332 , 0.131 , 0.1236 , 0.1251 , 0.1336 , 0.1354 ,
0.1268 , 0.1312 , 0.1264 , 0.1203 , 0.1313 , 0.1291 , 0.1362 , 0.1304 , 0.1251 , 0
.1285 , 0.1307 , 0.129 , 0.1221 , 0.133 , 0.1301 , 0.1285 , 0.1337 , 0.1267 , 0.12
95 , 0.1218 , 0.1303 , 0.1295 , 0.1243 , 0.1314 , 0.1274 , 0.13 , 0.1344 , 0.1258
, 0.1268 , 0.1279 , 0.1277 , 0.1295 , 0.129 , 0.1274 , 0.1247 , 0.1304 , 0.1287 ,
0.1263 , 0.1309 , 0.1215 , 0.1243 , 0.1265 , 0.1312 , 0.1296 , 0.1274 , 0.135 , 0.
1267 , 0.1312 , 0.1302 , 0.1301 , 0.1315 , 0.1271 , 0.1328 , 0.13 , 0.1282 , 0.128
4 , 0.1312 , 0.1327 , 0.1307 , 0.1342 , 0.1332 , 0.1308 , 0.1308 ,
```

We develop an algorithm by following the operations in pdf file and find the success probability clearly close to 0.1293 as expected.

And then I execute 10000 times the first algorithm in a for loop with range 10 which means that I collect 10 data and all these 10 data in totally give me 1295 success as I look at the file that I show the all executions but this file occupies 25MB so I added the smaller version to the overall zip file.



For the second part I developed an algorithm similar to previous algorithm and make some comparisons and plot them on graphs as you can see below.

If you see blue color on any figure, then it means that I execute second algorithm and if you see red color then it means that I execute first algorithm.

For these figures I execute 100 times the two algorithm and make this comparison.

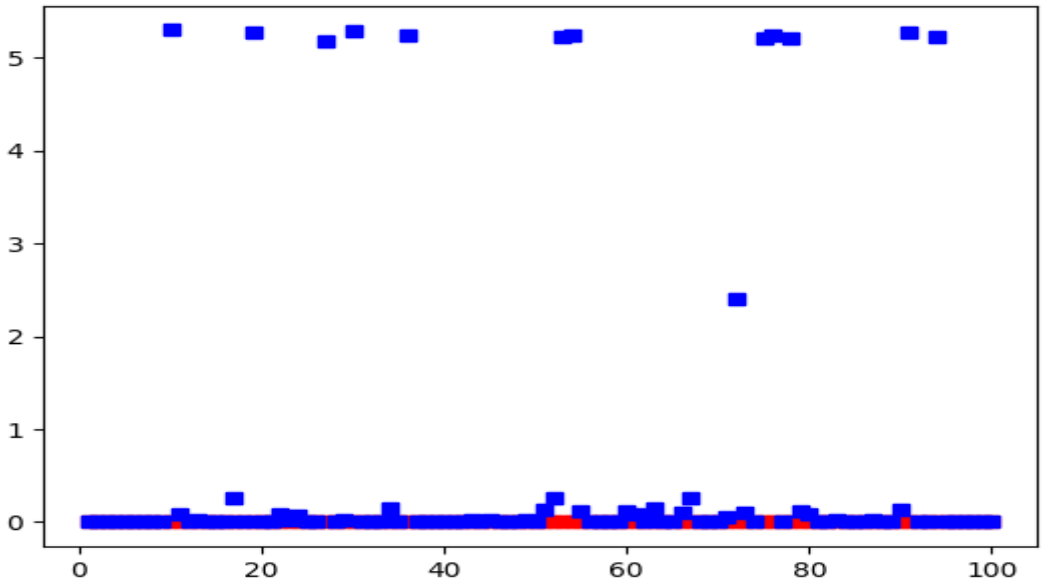


Figure 8

The red points show the execution times of the first algorithms and blue points show execution times of the seconds algorithms. As we can see mainly blue points overlap the red point. Nevertheless, blue points sometimes 10 times bigger than red points.
So, the second algorithm last more.

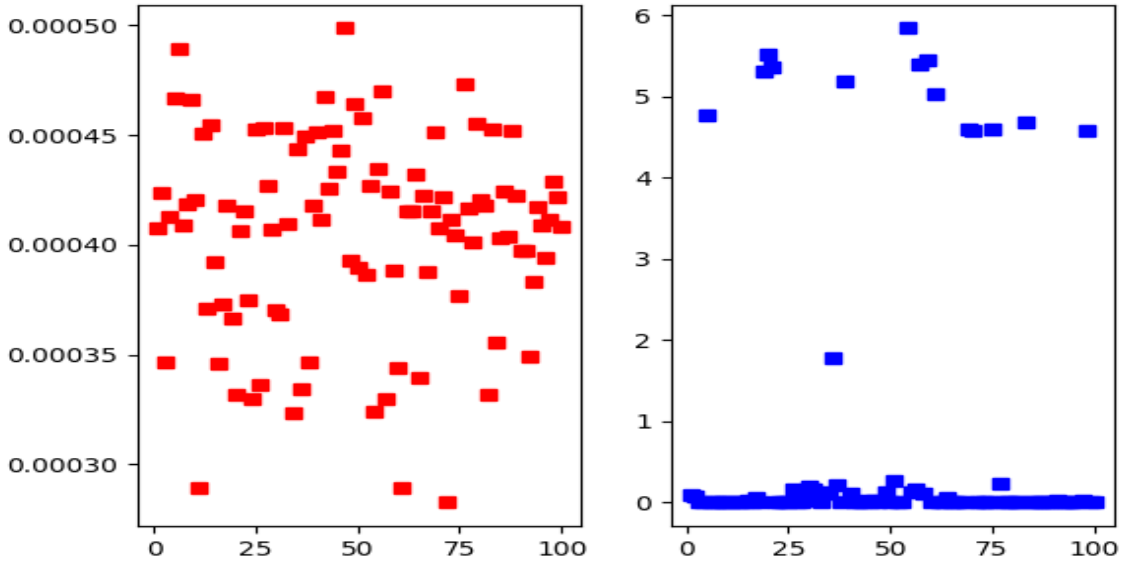


Figure 9

If we need to observe the differences between execution times of each algorithm, the left figure is first algorithm's execution times distribution and the right figure is second algorithm's execution times distribution. The red points are placed between 0 and 0.00050 and the blue points are mainly around 0 point at some point it goes to around 2, 5 and 6.

For these figures I execute 1000 times the two algorithm and make this comparison.

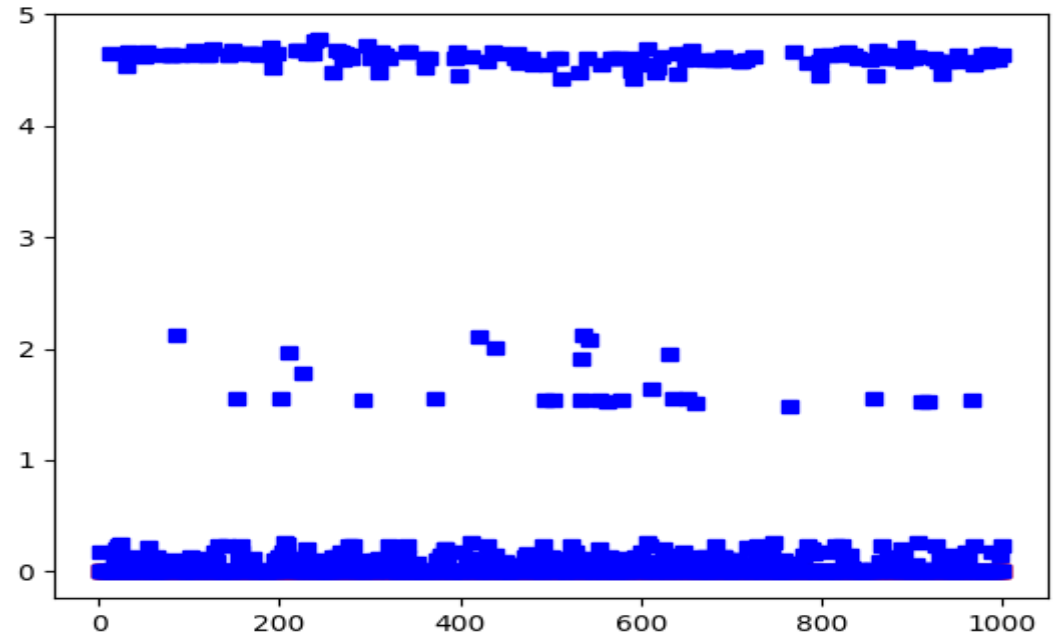


Figure 10

In this approach again blue points overlap the red point as **I execute 1000 times** but this time there are more points that are greater than red points.

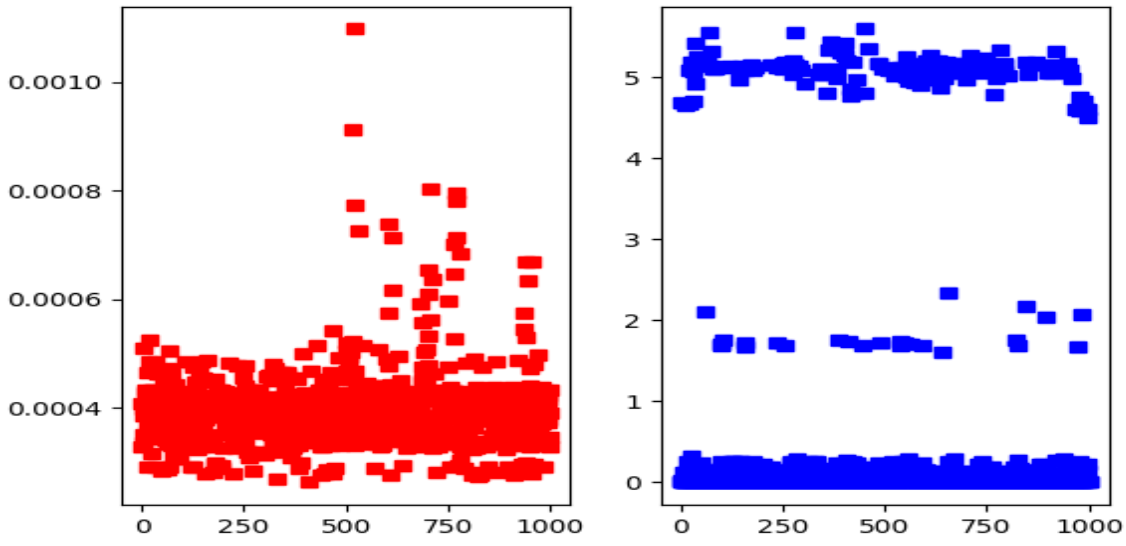


Figure 11

If we need to observe the differences between execution times of each algorithm, it looks like the previous figure 9. However, this time the red points are placed between 0 and 0.0010 so we can say that the execution time of the first algorithm decreases and the execution times of second algorithm this time go to around 2, 5 and 6 more so the execution times of second algorithm increases.

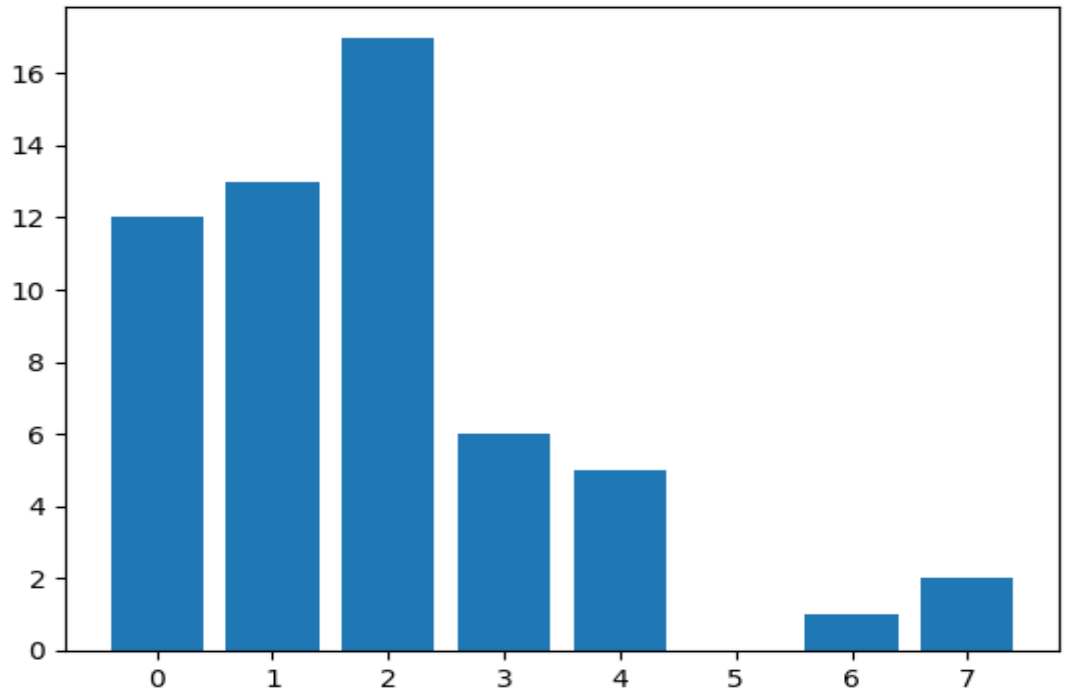


Figure 12

In this figure above **I execute 100 times second algorithm** and x axis represents the k value (k<8) and y axis represents the success times that if we eventually place the all queens. As we can see from this figure, we have success if k is 0,1,2.

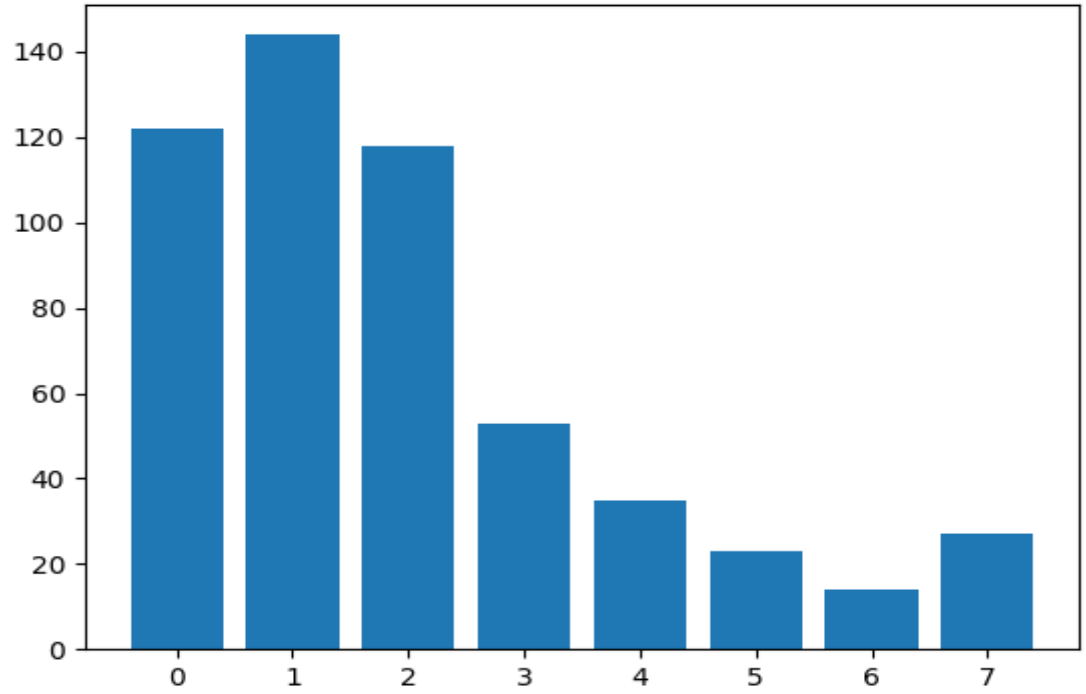


Figure 13

In this figure above **I execute 1000 times second algorithm** and it looks like as previous one.

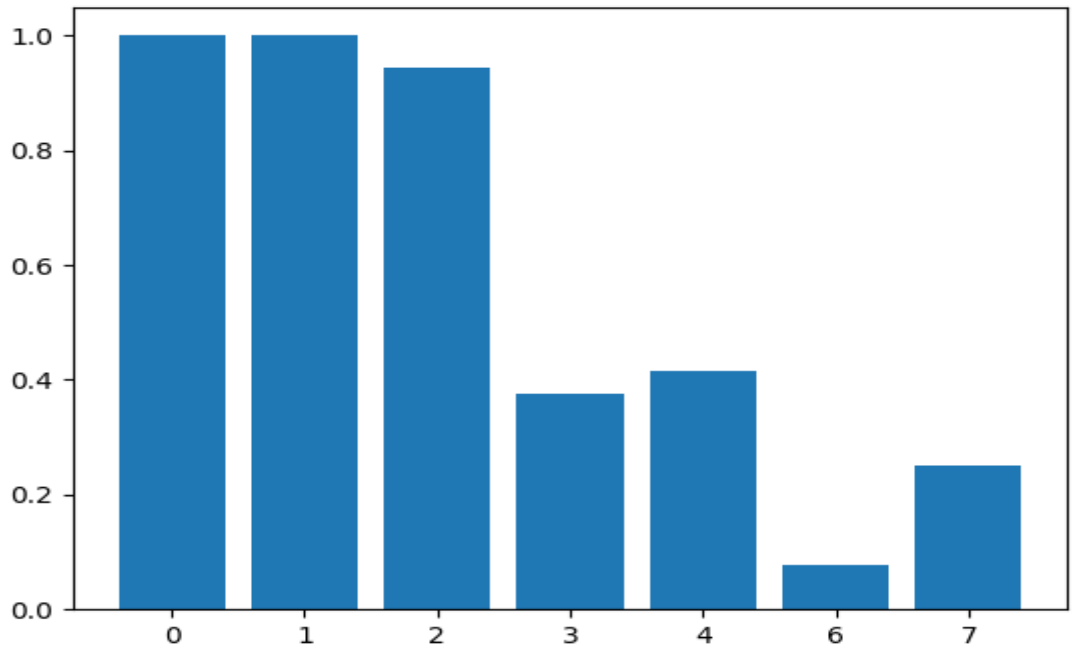


Figure 14

In this figure **I execute 100 times second algorithm** and again x axis represents the k value (k<8) and y axis represents the success probability of placing all the queens. As expected, k=0 has success probability 1.0 and k=1 has the same values. Then we can understand that if we choose k as 0 or 1, we can place all the queens since deterministic algorithm can places all the queens but as we increase the k value the success probability decreases. The overall first algorithm gives us 0.1293 success probability in average in this algorithm in average the success probability 0.54 much higher since from some point we use deterministic algorithm but nevertheless the success probability is not 1 since we use random indexes to place queens.

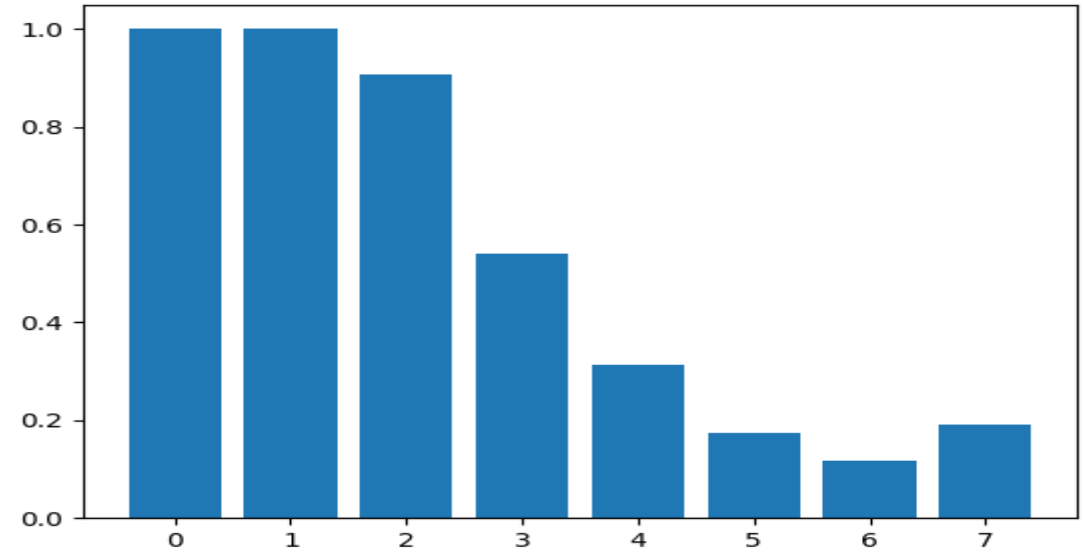


Figure 15

In this figure **I execute the second algorithm 1000 times** and it looks like as I execute 100 times.

Lastly, I plot the execution times of second algorithm for each k value.

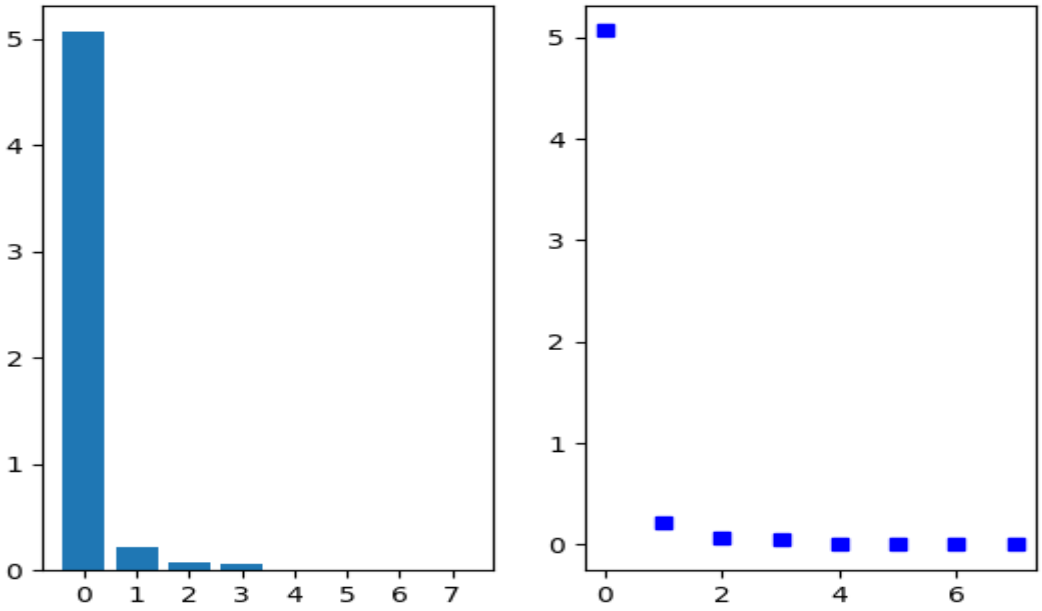


Figure 16

This figure was plotted when I execute the second algorithm 100 times.

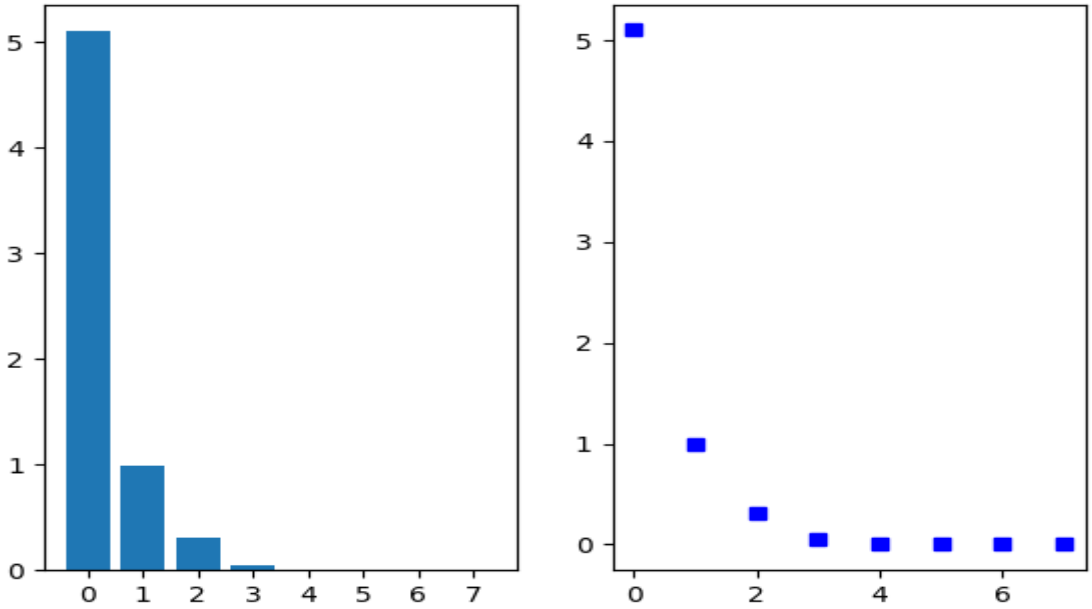


Figure 17

This figure was plotted when I execute the second algorithm 1000 times.

As you can see from both figure in average the execution times are distributed exponentially so if k values are small then the execution times are high in second algorithm.