CMPE 300 ANALYSIS OF ALGORITHMS
PROJECT 3 - ANSWERS

PART 1

1.1) Fill the steps of one successful and one unsuccessful execution for each p value

1.1.1) Success - p=0.7

	0	1	2	3	4	5	6	7
0	3	-1	-1	42	7	14	-1	-1
1	0	43	2	5	-1	-1	8	13
2	-1	4	41	34	15	6	-1	-1
3	44	1	18	-1	-1	35	12	9
4	19	24	33	40	11	16	29	36
5	-1	-1	20	17	26	39	10	-1
6	23	32	25	-1	21	30	37	28
7	-1	-1	22	31	38	27	-1	-1

1.1.2) Unsuccessful - p=0.7

	0	1	2	3	4	5	6	7
0	-1	-1	-1	-1	-1	15	-1	-1
1	-1	-1	-1	-1	-1	-1	-1	16
2	-1	-1	-1	-1	14	9	-1	-1
3	-1	-1	-1	8	-1	-1	13	10
4	-1	-1	-1	-1	6	11	-1	-1
5	-1	4	7	2	-1	-1	-1	12
6	-1	1	-1	5	-1	-1	-1	-1
7	-1	-1	3	0	-1	-1	-1	-1

1.1.3) Success - p=0.8

	0	1	2	3	4	5	6	7
0	-1	18	7	28	-1	44	47	30
1	6	**27**	**-1**	**17**	**32**	**29**	**-1**	**45**
2	19	**8**	**21**	**24**	**43**	**46**	**31**	**48**
3	22	**5**	**26**	**9**	**16**	**33**	**42**	**-1**
4	13	**20**	**23**	**4**	**25**	**10**	**49**	**-1**
5	0	**3**	**12**	**15**	**38**	**41**	**34**	**-1**
6	-1	**14**	**1**	**-1**	**11**	**36**	**39**	**50**
7	2	**-1**	**-1**	**37**	**40**	**51**	**-1**	**35**

1.1.4) Unsuccessful - p=0.8

	0	1	2	3	4	5	6	7
0	14	31	24	-1	-1	-1	18	21
1	25	**10**	**15**	**30**	**23**	**20**	**-1**	**-1**
2	28	**13**	**26**	**9**	**-1**	**17**	**22**	**19**
3	11	**-1**	**29**	**16**	**-1**	**8**	**-1**	**-1**
4	-1	**27**	**12**	**-1**	**6**	**3**	**-1**	**-1**
5	-1	**-1**	**-1**	**2**	**-1**	**-1**	**7**	**4**
6	0	**-1**	**-1**	**-1**	**-1**	**5**	**-1**	**-1**
7	-1	**-1**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**

1.1.5) Success - p=0.85

	0	1	2	3	4	5	6	7
0	24	37	46	41	32	39	44	-1
1	47	**18**	**25**	**38**	**45**	**42**	**33**	**54**
2	-1	**23**	**36**	**-1**	**40**	**31**	**12**	**43**
3	17	**48**	**19**	**26**	**13**	**34**	**53**	**0**
4	20	**-1**	**22**	**35**	**52**	**1**	**30**	**11**
5	49	**16**	**-1**	**14**	**27**	**8**	**5**	**2**
6	-1	**21**	**-1**	**51**	**6**	**3**	**10**	**29**
7	-1	**50**	**15**	**-1**	**9**	**28**	**7**	**4**

1.1.6) Unsuccessful - p=0.85

	0	1	2	3	4	5	6	7
0	-1	23	8	-1	18	21	-1	-1
1	15	**-1**	**17**	**22**	**-1**	**-1**	**-1**	**20**
2	24	**7**	**14**	**9**	**-1**	**19**	**-1**	**-1**
3	-1	**16**	**1**	**6**	**13**	**-1**	**-1**	**-1**
4	0	**5**	**-1**	**-1**	**10**	**-1**	**-1**	**-1**
5	-1	**2**	**-1**	**4**	**-1**	**12**	**-1**	**-1**
6	-1	**-1**	**-1**	**11**	**-1**	**-1**	**-1**	**-1**
7	-1	**-1**	**3**	**-1**	**-1**	**-1**	**-1**	**-1**

1.2) Fill the table and comment on it

1.2.1)

p	Number of Success	Number of Trials	Probability	Total Time of Execution
0.7	16649	100000	0.16649	8.81
0.8	2443	100000	0.02443	17.67
0.85	558	100000	0.00558	26.59

1.2.1) Comments

Also answer these questions while commenting

1- How do changes on p affect total success probability and total execution time?

Success Probability:
- If p represents the percentage of squares to cover, increasing it may improve the success probability. A higher p means the algorithm aims to cover more squares, making it more likely to succeed. However, it may also make the problem harder to solve, and the success probability might decrease for very high `p` values.This is evident in the data, where success rates drop from 16.649% at **p=0.7** to 0.558% at **p=0.85**.

- Reducing p means the algorithm aims to cover fewer squares, potentially making it easier to find a solution. However, very low values of `p` might lead to a higher success probability but may not be as challenging.

Execution Time:
- As p increases, the algorithm's goal becomes more ambitious, potentially leading to longer execution times. The algorithm has to explore more possibilities to achieve the higher coverage, which could result in increased computational time.

- Reducing p may lead to faster execution times, as the algorithm has a less complex task to accomplish. However, very low p values might not challenge the algorithm enough, leading to faster but less interesting results.

-The total execution time approximately doubles as **p** increases from 0.7 to 0.85 (8.81 seconds to 26.59 seconds).

2- Define trade-offs of the algorithm.

- Complexity vs. Success Probability: There is a trade-off between the complexity of the problem (represented by `p`) and the success probability. A more complex task (higher p) may lead to a lower success probability, while a simpler task (lower p) may yield higher success rates.

- Execution Time vs. Precision: The trade-off between execution time and precision is apparent. Striking a balance between achieving a precise solution (higher p) and limiting the execution time is crucial. Adjusting `p` allows you to tune this trade-off.

PART 2

2.1) Fill the tables and comment on them

2.1.1) p = 0.7

k	Number of Success	Number of Trials	Probability	Total Time
0	100000	100000	1.0	23.82
2	100000	100000	1.0	22.67
3	99928	100000	0.99928	22.12

2.1.2) p = 0.8

k	Number of Success	Number of Trials	Probability	Total Time
0	100000	100000	1.0	27.16
2	100000	100000	1.0	25.85
3	99946	100000	0.99946	25.55

2.1.1) p = 0.85

k	Number of Success	Number of Trials	Probability	Total Time
0	100000	100000	1.0	28.55
2	100000	100000	1.0	27.23
3	99940	100000	0.9994	26.59

2.1.2) Comments

Also answer these questions while commenting

1- How does total time change with k?

- Total Time Trend with k: The total time tends to decrease as k increases. This is observed consistently across different values of p. As k increases from 0 to 3, the total time generally decreases. For example, at p=0.7, the total time decreases from 23.82 seconds at k=0 to 22.12 seconds at k=3. When the number of initial random steps (k) increases, there is a tendency for the execution time to decrease. This unexpected phenomenon stem from the algorithm's adaptability, where an increased number of random steps influences the reduction of deterministic steps needed to reach a specific move count, consequently accelerating the algorithm.

2- How do total time change with p for a specific k value? How does this change differ from the first part?

-Total Time Trend with p for a Specific k: For a specific k value, the total time tends to increase as p increases. This is consistent with the observations in the first part where higher values of p lead to more complex problems and longer execution times. For instance, at k=3, the total time increases from 22.12 seconds at p=0.7 to 26.59 seconds at p=0.85. This indicates that as the problem complexity increases (p), the algorithm requires more time to visit more squares. The trend is consistent with the first part where higher p values lead to longer total execution times since there is more squares to visit.

3- Run this algorithm for each p value for k a value larger than 10 multiple times. What are your thoughts?

If the number of initial random steps (k) increases, there is a tendency for the execution time to decrease, contrary to the expected. Additionally, there is a consistent pattern where an increase in the probability parameter (p) leads to an increase in execution time. The reason for this phenomenon is that as k increases, the number of random steps also increases, potentially reducing the deterministic steps required to reach a certain move count, thus accelerating the algorithm. On the other hand, an increase in p result in a need for more unsuccessful attempts to visit more squares, leading to an increase in execution time.

```
--- p = 0.7 ---
LasVegas Algorithm With p = 0.7, k = 11
Number of successful tours: 97738 Number of trials: 100000
Probability of a successful tour: 0.97738
Execution time: 17.84432601928711

--- p = 0.8 ---
LasVegas Algorithm With p = 0.8, k = 11
Number of successful tours: 97635 Number of trials: 100000
Probability of a successful tour: 0.97635
Execution time: 20.45591711997986

--- p = 0.85 ---
LasVegas Algorithm With p = 0.85, k = 11
```

```
Number of successful tours: 97753 Number of trials: 100000
Probability of a successful tour: 0.97753
Execution time: 21.586161375045776
——————————
--- p = 0.7 ---
LasVegas Algorithm With p = 0.7, k = 12
Number of successful tours: 97176 Number of trials: 100000
Probability of a successful tour: 0.97176
Execution time: 17.05636477470398

--- p = 0.8 ---
LasVegas Algorithm With p = 0.8, k = 12
Number of successful tours: 97157 Number of trials: 100000
Probability of a successful tour: 0.97157
Execution time: 19.873543977737427

--- p = 0.85 ---
LasVegas Algorithm With p = 0.85, k = 12
Number of successful tours: 97118 Number of trials: 100000
Probability of a successful tour: 0.97118
Execution time: 68.09282207489014
—————————
--- p = 0.7 ---
LasVegas Algorithm With p = 0.7, k = 13
Number of successful tours: 96459 Number of trials: 100000
Probability of a successful tour: 0.96459
Execution time: 16.72095012664795

--- p = 0.8 ---
LasVegas Algorithm With p = 0.8, k = 13
Number of successful tours: 96441 Number of trials: 100000
Probability of a successful tour: 0.96441
Execution time: 19.431943893432617

--- p = 0.85 ---
LasVegas Algorithm With p = 0.85, k = 13
Number of successful tours: 96467 Number of trials: 100000
Probability of a successful tour: 0.96467
Execution time: 20.78782296180725
```

PART 3

In this part, you will compare Part1 and Part2 algorithms according to their ability to solve the actual Knight’s Problem where p=1.

- Run Part1 algorithm with p=1.
- Run Part2 algorithm with p=1 and k=0.
- Run Part2 algorithm with p=1 and a k value you think will work well.

Clearly state your findings and comment on them. When would you choose Part1 algorithm and when would you choose the other?

Firstly, after we run Part1 algorithm with p=1 on 100000 trials, we get 0 successful trial and the execution takes around 15 seconds.

Secondly, after we run Part2 algorithm with p=1 and k=0 on 100000 trials, we get 100000 successful trials (all of them) and the execution takes around 40 seconds.

Thirdly, running Part2 algorithm with k values larger than 0 to some bigger values (around 40) results in too long executions times which the result sometimes can not be observable, so we decided to use k=0 again for a better work and got the result similar to the above observation.

Lastly, according to previous experiments, we can claim that Part1 algorithm is faster due to the fact that it is a probabilistic solution to the problem. However, the algorithm tends to produce very few exact solutions, moreover we can say it is very far from finding a solution since it decides the next step randomly in every time. On the other hand, Part2 algorithm finds an exact solution for every random start on the table since it is a deterministic algorithm. The algorithm may take long time than Part1 algorithm but it can produce more solution than Part1 algorithm because of its heuristic approach.

To sum up, if we need an algorithm too fast but cannot guarantee a proper solution, we can use Part1. Also, if we need an algorithm which is slower but might find appropriate solution, we can use Part2.