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

1.1.2) Unsuccessful - p=0.7

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

1.1.3) Success - p=0.8

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

1.1.4) Unsuccessful - p=0.8

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

1.1.5) Success - p=0.85

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

1.1.6) Unsuccessful - p=0.85

	0	1	2	3	4	5	6	7
0	-1	-1	14	35	-1	33	16	-1
1	-1	**36**	**19**	**12**	**15**	**-1**	**-1**	**-1**
2	-1	**13**	**-1**	**21**	**34**	**17**	**32**	**-1**
3	37	**20**	**11**	**18**	**-1**	**22**	**-1**	**0**
4	28	**-1**	**38**	**-1**	**-1**	**31**	**6**	**23**
5	39	**10**	**29**	**26**	**7**	**4**	**1**	**-1**
6	-1	**27**	**8**	**3**	**30**	**-1**	**24**	**5**
7	9	**40**	**-1**	**-1**	**25**	**2**	**-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	16484	100000	0.16484	5.16512
0.8	2425	100000	0.02425	5.19868
0.85	593	100000	0.00593	5.27263

1.2.1) Comments

Also answer these questions while commenting

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

➔ As p value goes up, the success probability goes down; because the threshold of move count gets set to higher values and it becomes difficult for each try to achieve that many moves without going into a dead-end. On the other hand, total execution time gets higher when p value increases. The main reason is that for lower levels of p, the algorithm has the chance to stop execution early. Considering number of successes for each p value, it can be seen that 0.7 has the lead, so this allows 0.7 to have more chance to stop execution early, which contributes to the reduction in time of execution. However, differences are very small, due to nature of this algorithm. It simply performs several random moves and terminates without backtracking, so the effect of increased steps is very small (overhead of a few extra random calls, checks, etc.).

2- Define trade-offs of the algorithm.

➔ As the number of trials increases, algorithm yields more precise results for each value of p, but execution time increases.
➔ As the value of p increases total number of successes and, in turn, probability of getting a success decreases, and vice versa.
➔ As the value of p increases, total execution time also increases.

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	2.68968
2	100000	100000	1.0	2.70679
3	99942	100000	0.99942	2.73466

2.1.2) p = 0.8

k	Number of Success	Number of Trials	Probability	Total Time
0	100000	100000	1.0	3.26246
2	100000	100000	1.0	3.56355
3	99931	100000	0.99931	5.66781

2.1.1) p = 0.85

k	Number of Success	Number of Trials	Probability	Total Time
0	100000	100000	1.0	7.72483
2	100000	100000	1.0	245.41450
3	99939	100000	0.99939	685.53360

2.1.2) Comments

Also answer these questions while commenting

1- How does total time change with k?

Trials clearly show that total time increases as k increases. Even though increasing k causes recursing on a smaller tree, it also reduces the number of solutions. Since the tree grows exponentially, even smaller trees have too many paths to explore. Because of that, reducing the number of solutions has the higher impact and causes algorithm to work longer.

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

Total time increases as p increases. However, the increase is much more significant than the first part. Since this algorithm uses backtracking to find several steps, it scans through whole solution tree. It spends more time as the target move count increases because less traces match this condition (one can simplify this as harder solutions take more time to find).

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

Every p value approximately gave 0.095 probability of success for k=15, with increasing execution time as p increases. This can indicate that after k number of moves (k>10) only some specific solutions remain in tree and those solutions are answer to all p values until a certain p value threshold (that value might be 1 too but this would require a very long time to test). It is also important to note that we timed-out if a run takes more than 30 seconds for 0.85 p value, because it was getting stuck at some point and taking extremely long time. Here are the resulting logs (we run first two 10000 times too to make sure results are accurate):

```
--- p = 0.7 ---
LasVegas Algorithm With p = 0.7, k = 15
Number of successful tours : 945
Number of trials : 1000
Probability of a successful tour : 0.0945
Time taken : 0.0370631217956543

--- p = 0.8 ---
LasVegas Algorithm With p = 0.8, k = 15
Number of successful tours : 952
Number of trials : 1000
Probability of a successful tour : 0.0952
Time taken : 2.7290141582489014

--- p = 0.85 ---
LasVegas Algorithm With p = 0.85, k = 15
Number of successful tours : 947
Number of trials : 1000
Probability of a successful tour : 0.0947
```

```
Time taken : 136.77832508087158
```

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?

<u>1) Run Part1 algorithm with p=1</u>

```
Time taken for p = 1 : 5.212768793106079
LasVegas Algorithm With p = 1
Number of successful tours : 0
Number of trials : 100000
Probability of a successful tour : 0.0
```

Las Vegas algorithm was not able to find the complete solution within 100000 trials. Executing it more times might yield a success however this is not viable, since 100000 execution is high enough number of trials, it is accurate to estimate that this algorithm has 0 probability to find a solution when p=1. However, since deterministic algorithm's execution time grows very fast, it is more viable to use Las Vegas algorithm for a quick solution (note that we don't terminate when a successful result found in given execution times, if we did that, we would see execution time is much less than deterministic algorithm).

<u>2) Run Part2 algorithm with p=1 and k=0.</u>

This produces huge trees; we observed depth to become 62 however we couldn't record it to become 64. Theoretically if we wait long enough the algorithm will find a solution. However, this time is very long and not viable to use as solution in any situation.

<u>3) Run Part2 algorithm with p=1 and k=35.</u>

We chose 35 to create a half-determined half-random algorithm that utilizes both techniques strong points (and rounded 64/2 to 35, 30 didn't perform as well as 35). This produced:

```
LasVegas Algorithm With p = 1, k = 35
```

```
Number of successful tours : 31
Number of trials : 100000
Probability of a successful tour : 0.00031
Time taken : 583.6663038730621
```

Moreover, first solution was found within 3 seconds. This is a blazingly algorithm and produces a total solution to knight's tour problem.

So, the algorithm in the first part is useful when we want to visit more squares in a limited short amount of time. However, as the number of target squares increases the probability of success decreases, so after 0.95 p value it can't find a solution (experimental result). Algorithm in the second part is useful if we have enough computation resources and want to find a definite answer in a certain amount of time. However, finding an absolute solution to this problem using deterministic approach is not time effective. The most effective solution to this problem is using a mixture of both methods. Stepping half of the squares using Las Vegas algorithm and finding a solution for the remaining squares using deterministic methods yields both higher probability of success and faster execution times. This is because Las Vegas algorithm with a big k value reduces the solution tree significantly, while deterministic approach can find a solution from that tree without using too much time.