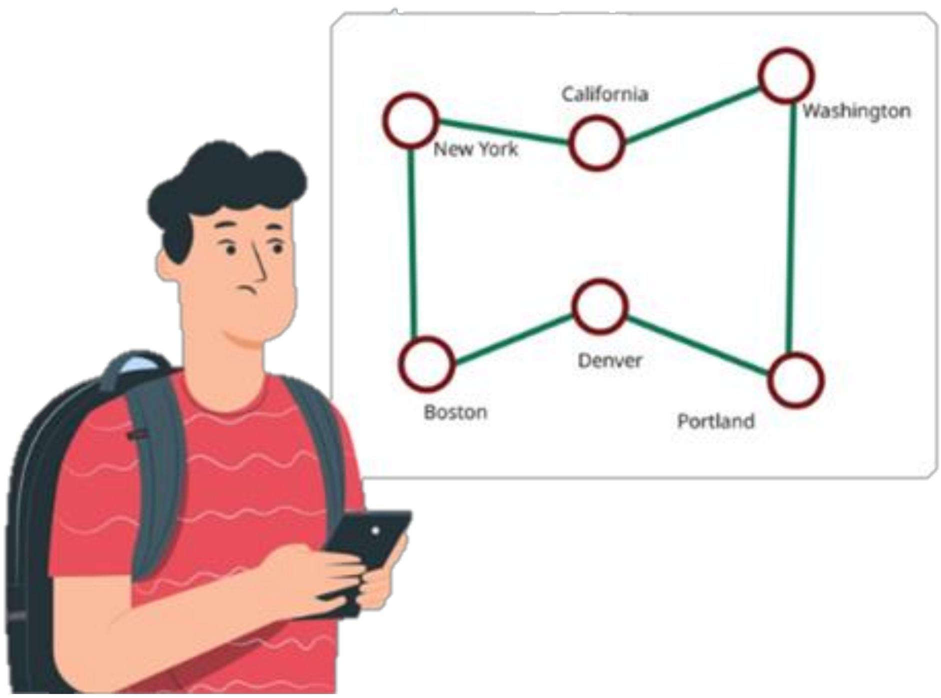
King Abdul-Aziz university
Faculty of Computing and Information Technology
Computer Science Department
CPCS223: Analysis and Design of Algorithms
Instructor: Dr. Reemah Alhebshi

Empirical Analysis of Solving the Traveling Salesman Problem



Student Name	Student ID	Section
Logain Sendi	[redacted]	B1A
[redacted]	[redacted]	B1A
[redacted]	[redacted]	B1A
[redacted]	[redacted]	B1
Hand in Date: 11th February 2023		

Task Assignment

Task	Student
Introduction	Logain Sendi
Experiment's Purpose	Logain Sendi
Efficiency Metric	Logain Sendi
Characteristics of The input Sample	[redacted]
Implementation Algorithms (pseudocode/tools)	[redacted]
Program run and data collection	[redacted]
Order of growth	[redacted]
Conclusion	[redacted]
Implement Second Algorithm	[redacted]

Table of Contents

1. Introduction

Algorithm analysis is an essential process in the field of computer science as well as a critical process in computing the complexity of an algorithm. Algorithm analysis determines the amount of time and space resources needed to execute an algorithm. Its characteristics are as follows: correctness, time and space efficiency, simplicity, and generality. These characteristics are what indicate an algorithm's complexity. (GeeksforGeeks, 2022)

Analyzing an algorithm is essential for multiple reasons, some of which are: anticipating how the algorithm will behave without running it; comparing various algorithms so we can decide which is ideal for our needs. However, we must remember that the analysis is simply an approximation.

In this report, we will analyze two different algorithms, the Traveling Salesman Problem and Traveling Salesman Dynamic Programming, and then compare them using the empirical analysis approach to decide which algorithm has better efficiency. The process will be broken down into several steps, with the first being to choose the efficiency metric and measurement unit, the second being to select the input range and size, the third being to implement and run each algorithm, and the fourth and final step being to evaluate the results of each algorithm and choose the one with the highest efficiency.

2. Empirical Analysis of Algorithms

2.1. Experiment's Purpose

This experiment's purpose is to examine the behavior of the two algorithms, Traveling Salesman Problem and Traveling Salesman Dynamic Programming, using the empirical approach to analyzing algorithms to compare and find the more efficient algorithm.

2.2. Efficiency Metric

There are two units of measuring running time in the empirical approach, the first is using a time unit, and the second is counting the number of times the algorithm's basic operation is executed.

We will compare both approaches to know which is preferred in this study. Measuring running time using a time unit is simple, and it is done by estimating the actual time it takes for the algorithm to be executed from start to finish using a standard time unit. However, this method is not practical since many outside factors play a role in this approach, such as the dependence on the computer's speed, the program's quality, the compiler used, and the difficulty of clocking the actual running time.

Measuring the running time by counting the number of times a basic operation is executed is done first by identifying the basic operation in an algorithm. It is usually the operation that contributes the most to the running time. You then will use this equation: $T(n) \approx C_{op} C(n)$, where T(n) is the running time, Cop is the execution time for basic operations, and C(n) is the number of times a basic operation is executed, to estimate the running time. It is important to remember that the result is only an approximation and that this formula will give a reasonable estimate of the running time unless n is extremely large or small.

Since this study uses the same device and conditions to compare both algorithms, the outside factors won't play a significant role in this calculation, so the best approach for this study is using a time unit.

3. Design and Procedure

3.1. Characteristics of the Input Sample

Choosing the sample of inputs is one of the most crucial considerations in the empirical analysis of an algorithm. For this reason, we demonstrate our decision-making procedure for the input sample in this section.

A problem is NP (nondeterministic polynomial) if its solution can be guessed and verified in polynomial time. The TSP is an NP-hard problem, meaning that finding an exact solution becomes infeasible as the number of cities increases. As a result, we set a reasonable range of input sizes that will not prevent the program from running. This range is neither too small nor noticeably huge, ranging between 3 and 13 cities. The potential for some algorithms to behave atypically over some instances is a worry in any empirical analytic investigation. We used both even and odd numbers to prevent inaccurate findings in our study.

We chose the distances to be unique random values from 10 to 100 to remove any preferences for specific input instances. Now, we need to run the program multiple times on different distances for the same input size, so we can set a range to compute the efficiency for each algorithm later.

In general, when we chose the input sample, we kept in mind that it should be characterized by

- Simplicity: Input data should be easy to understand and process.
- Optimization: The input data should allow optimization to find the shortest possible route.
- Relevance: The input data should only include the relevant information needed to solve the TSP problem.
- Scalability: The input data should be scalable to accommodate enormous instances of the TSP problem.

3.2. Implementation Algorithms

3.2.1. Brute-Force Algorithm

```
Algorithm TSP(pos, path, cost)
// Brute force algorithm to solve the traveling salesman problem recursively
//Input: integer number pos that represent current position, array path that represent the //current path, integer number cost that represent the current
//Global variables: integer number n = number of areas (4<=n<=10), integer number minC = // the minimum cost, array vis[0…n-1] to show visited and unvisited areas, 2 dimensional //array dis[0..n-1][0…n-1] to represent our graph (areas and distances between them), array //pathCost to store cost of each path, 2 dimensional array allPaths to store all paths
//output: The shortest tour through the given set of n areas
        //base case
        if size of path = n
                cost+=dis[pos][0]
                add cost to the pathCost
                add path to the allPaths
                minC=min(minC,cost)

        else
                //Visit rest of the unvisited areas
                 for i = 1 to n-1
                        //go to next node
                        if !vis[i]
                              vis[i]=true;
                                create new path
                                newPath=path
                                add area i to the newPath
                                    TSP(i, newPath, cost+dis[s][i]);
                                     vis[i]=false;
```

(Levitin, 2023)

3.2.2. Dynamic Programming algorithm

```
Algorithm TSPdp(msk , pos, p)
// Solve the traveling salesman problem by dynamic programming
// Input: msk = mask that represent which areas are visited and which are not, pos = current
//position
//Global variables: integer number n  = number of nodes (4<=n<=10), integer number //allvisited
= (2^n)-1 that represent the mask when all nodes are visited, 2 dimensional array //dis[0..n-
1][0…n-1] to represent our graph (areas and distances between them) , 3 //dimensional array
dp[0…(2^n)-1][0…n-1][0..n+1] to implement dynamic programming   //(initialized with -1), 2
dimensional array allPaths to store all paths
// Output: Array that stored at the first index the cost of the optimal path, and in the rest of //the
indexes stored the optimal path (the areas in the optimal order) ( The shortest tour /through the
given set of n areas)
        //base cases
        if msk=allvisited  // all the areas are visited
                create array opt
                //the first indix in opt will stor the cost of the path
                add dis[pos][0] to the opt
                add 1 to the opt  // start traveling from area 1
                return opt

        if dp[msk][pos][0]!=-1 // this path has already been calculated
                create array opt
                opt = the array that stored in dp[msk][pos]
                  return opt

        // store the answer (optimal path and the cost will stores at the first index)
        create array ans
        initialize the cost in ans with inf
        //Visit rest of the unvisited areas
         for i=0 to n-1
                if msk&(1<<i)=0)  //check if the area i is unvisited
                        // collect the path
                        create array newAns
                        newAns = TSPdp( msk|(1<<i),i)
                        add dis[pos][i] to the cost that stored at index 0 in the newAns
                        add the area I to the newAns
                        if pos=0
                                add the newAns to the allPaths
                          // get the optimal path
                        ans= min(ans,newAns)

        return dp[msk][pos] = ans;// return the answer  (Walker, 2022)
```

3.3. Tools

1- Netbeans java to write the java code
2- Microsoft Excel program to record the data obtained, compute the averages, and draw graphs based on the recorded data.

4. Empirical Analysis of The Algorithms

4.1. Run Program and Collect Data

After running the two algorithms 10 times for each input size (starting from 3 up to 13 cities) with a random distance between them in the range of 10km-100km. The data obtained is showed in the tables below. Following the tables are the graphs that represent the data recorded.

Brute Force Algorithm													
Cities	Trail										Best	Average	Worst
	trail 1	trail 2	trail 3	trail 4	trail 5	trail 6	trail 7	trail 8	trail 9	trail 10			
3	67200	107100	38400	127900	39600	64500	73200	70800	61800	78700	38400	72920	127900
4	199700	174599	100900	96400	158500	106000	197400	104400	187500	98001	96400	142340	199700
5	528200	737400	735499	558400	393800	1073300	818600	1475900	564200	469900	393800	735519.9	1475900
6	2036200	1277100	1572500	1480700	1641200	1487400	1486100	1470500	1491400	1056600	1056600	1499970	2036200
7	5765500	7063200	5745900	5655100	6274100	4292700	5752500	4344700	4620600	5767500	4292700	5528180	7063200
8	11122800	11022700	11322500	9247100	10602500	9853600	9991900	10782000	9851800	9158500	9158500	10295540	11322500
9	34159500	25453800	38468900	35650300	35517500	32435800	32496800	27364200	40197800	27774300	25453800	32951890	40197800
10	160248300	141274800	135740300	135263500	130413900	148305200	131079500	135833800	141131400	136776700	130413900	139606740	160248300
11	979861900	966770400	973815000	894101700	898452600	1040929700	876944299	873098500	1022489100	1030166000	873098500	955662919.9	1040929700
12	9254888000	9609440700	9540253600	9271744300	9542862200	11195418200	11252484200	9465117300	9241003900	10653177000	9241003900	9902638940	11252484200
13	1.23672E+11	1.18048E+11	1.18107E+11	1.21046E+11	1.17963E+11	1.2047E+11	1.20563E+11	1.21375E+11	1.17668E+11	1.20296E+11	1.17668E+11	1.19921E+11	1.23672E+11

Table 1:brute-force algorithm

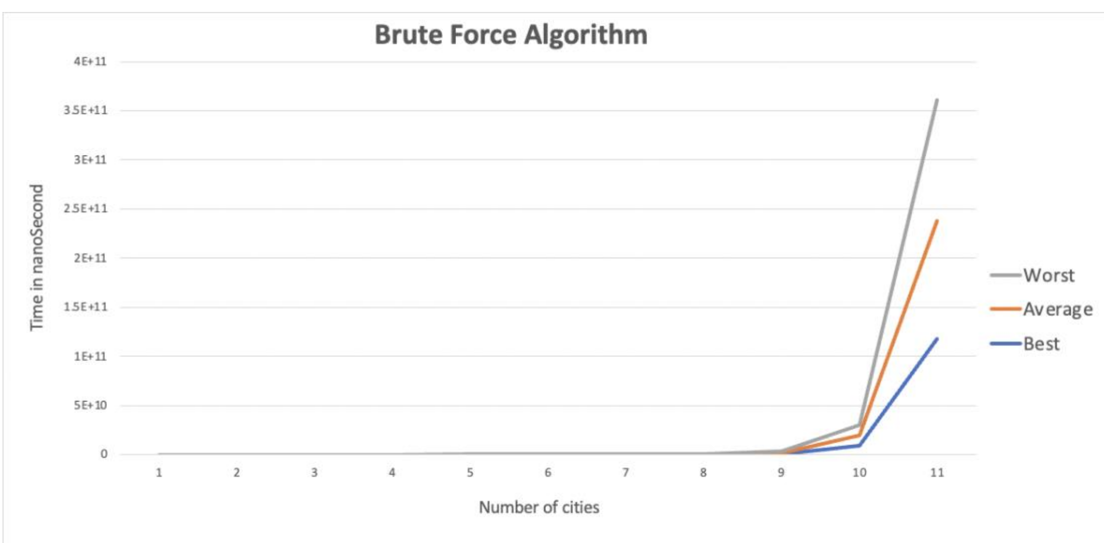


Figure 1:Brute-Force Algorithm graph

Dynamic Programming Algorithm													
Cities	Trail										Best	Average	Worst
	trail 1	trail 2	trail 3	trail 4	trail 5	trail 6	trail 7	trail 8	trail 9	trail 10			
3	41200	114400	73000	41300	69400	41100	73400	42800	89900	41999	41100	62849.9	114400
4	172700	131000	261200	127100	172800	124800	207100	192100	189100	135600	124800	171350	261200
5	440600	631800	444500	510100	499700	1182500	379700	492600	682200	950400	379700	621410	1182500
6	1016500	1299800	1225500	872301	897900	924700	694100	882200	826300	737700	694100	937700.1	1299800
7	2337000	3660100	2342200	1607500	2013100	1659800	2320300	2244200	1803200	2356600	1607500	2234400	3660100
8	5899300	4664600	4824000	4236700	5150700	4426600	4642000	5483500	5090900	4892400	4236700	4931070	5899300
9	8369600	9947800	4826800	4687400	6387600	6292500	6351700	6673000	6914600	6256200	4687400	6670720	9947800
10	12933000	9263400	8729900	6720700	8553500	6416700	7114500	7035200	7678100	8364500	6416700	8280950	12933000
11	11601200	16940400	9667000	15027200	11378700	12780600	14581000	15451500	15275800	14120000	9667000	13682340	16940400
12	21380800	24345000	18769200	18562700	17786500	17497800	19914200	27654600	21191400	18581500	17497800	20568370	27654600
13	32270300	53757900	36483900	44493100	37195000	34343300	44917900	32588100	30562800	30087800	30087800	37670010	53757900

Table 2: Dynamic programming algorithm

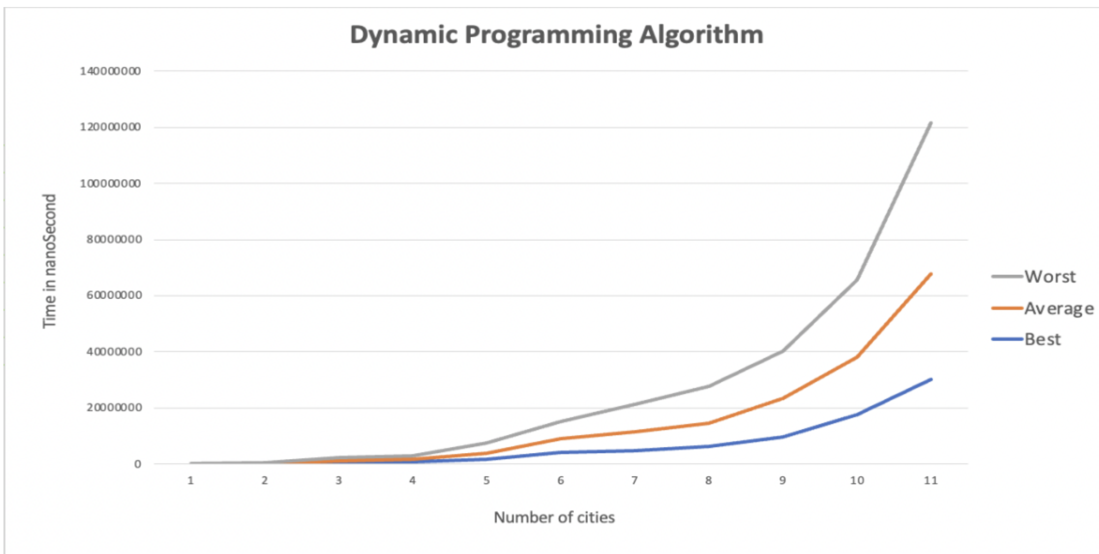
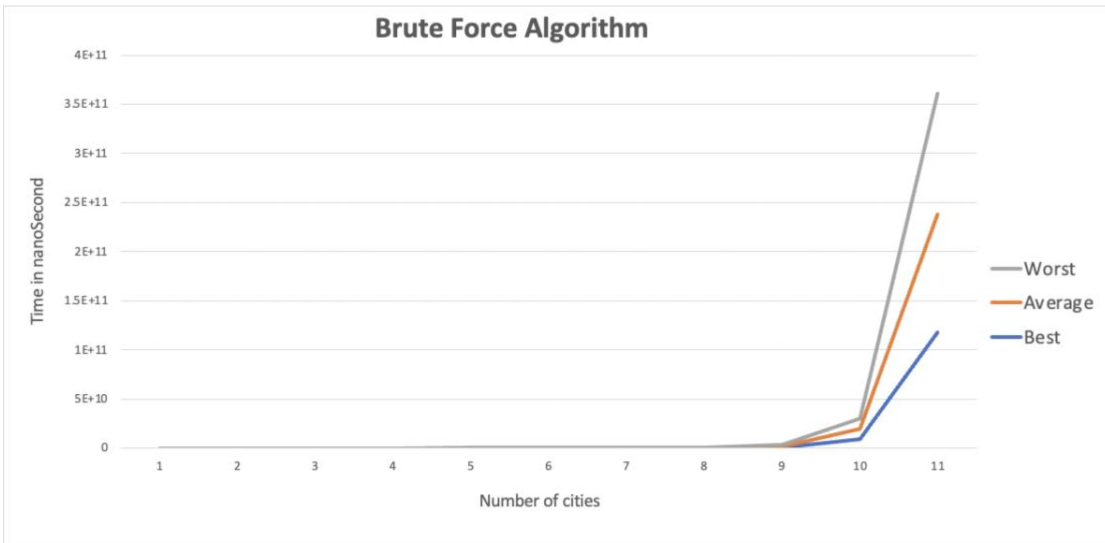


Figure 2: Dynamic programming algorithm graph

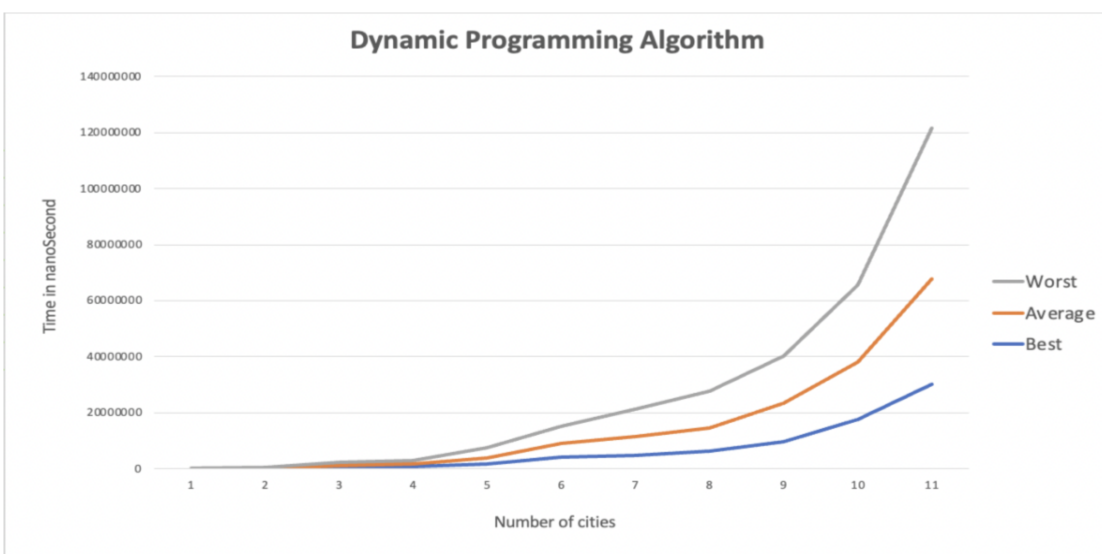
4.2. Analysis of Obtained Data

4.2.1. Order of Growth

After running the algorithms and collecting our data we calculate the square error for each algorithm in all its cases (best, average ,and worst)to find their efficiency and we analyze that cases by using our result with the data, graphs, and tables that we obtained in 4.1 in this report .



Let's start with Brute-Force Algorithm. After calculating the square error of the best, average, and worst-case of TSP (see results in Appendix A) and the figure shown above we can notice that best, average, and worst cases are acting similarly. They are growing n factorial (n!) as the size of inputs(the number of cities) increases. We can say that the order of growth of TSP using Brute-Force Algorithm is O(n!) .



On the other hand, when calculating the square error of using the dynamic programming algorithm best, average, and worst-case of TSP (see results in Appendix A) and the figure shown above we found that best, average, and worst cases are acting also similarly. But they are growing exponentially as the size of inputs increases so we can say that the order of growth of TSP using a dynamic programming algorithm is O(2^n).

By comparing the order of growth of both algorithms to find the best algorithm to solve TSP. O(n!) is greater than O(2^n) so we conclude that dynamic programming is the best algorithm to solve this problem.

5. Conclusion

At the end of the empirical analysis, after implementing the brute-force algorithm and Dynamic Programming algorithm in our problem Traveling Salesman Problem (TSP), we calculated the execution time for each algorithm, recorded all the results, and compared them. We concluded that the TSP is best solved using the dynamic programming technique. On the other side, when we encounter a TSP, the brute force technique must be our final resort.

6. Appendix

	Brute Force Algorithm												
Cities	Trail												
	trail 1	trail 2	trail 3	trail 4	trail 5	trail 6	trail 7	trail 8	trail 9	trail 10	Best	Average	Worst
3	67200	107100	38400	127900	39600	64500	73200	70800	61800	78700	38400	72920	127900
4	199700	174599	100900	96400	158500	106000	197400	104400	187500	98001	96400	142340	199700
5	528200	737400	735499	558400	393800	1073300	818600	1475900	564200	469900	393800	735519.9	1475900
6	2036200	1277100	1572500	1480700	1641200	1487400	1486100	1470500	1491400	1056600	1056600	1499970	2036200
7	5765500	7063200	5745900	5655100	6274100	4292700	5752500	4344700	4620600	5767500	4292700	5528180	7063200
8	11122800	11022700	11322500	9247100	10602500	9853600	9991900	10782000	9851800	9158500	9158500	10295540	11322500
9	34159500	25453800	38468900	35650300	35517500	32435800	32496800	27364200	40197800	27774300	25453800	32951890	40197800
10	160248300	141274800	135740300	135263500	130413900	148305200	131079500	135833800	141131400	136776700	130413900	139606740	160248300
11	979861900	966770400	973815000	894101700	898452600	1040929700	876944299	873098500	1022489100	1030166000	873098500	955662919.9	1040929700
12	9254888000	9609440700	9540253600	9271744300	9542862200	11195418200	11252484200	9465117300	9241003900	10653177000	9241003900	9902638940	11252484200
13	1.23672E+11	1.18048E+11	1.18107E+11	1.21046E+11	1.17963E+11	1.2047E+11	1.20563E+11	1.21375E+11	1.17668E+11	1.20296E+11	1.17668E+11	1.19921E+11	1.23672E+11

Cities = n	Best	1	log(n)	n	nLog n	n^2	n^3	2^n	n!		
3	38400	1474483201	1474523357	1474329609	1474450073	1473868881	1473868881	1473945664	1474099236		
4	96400	9292767201	9292843923	9292188816	9292495697	9289875456	9289875456	9289875456	9288333376		
5	393800	1.55078E+11	1.55078E+11	1.55075E+11	1.55076E+11	1.55059E+11	1.5498E+11	1.11634E+12	1.54984E+11		
6	1056600	1.1164E+12	1.1164E+12	1.11639E+12	1.11639E+12	1.11633E+12	1.11595E+12	1.84267E+13	1.11488E+12		
7	4292700	1.84273E+13	1.84273E+13	1.84272E+13	1.84272E+13	1.84269E+13	1.84243E+13	8.38758E+13	1.8384E+13		
8	9158500	8.38781E+13	8.38781E+13	8.3878E+13	8.3878E+13	8.38769E+13	8.38687E+13	6.47883E+14	8.31412E+13		
9	25453800	6.47896E+14	6.47896E+14	6.47895E+14	6.47895E+14	6.47892E+14	6.47859E+14	1.70077E+16	6.29554E+14		
10	130413900	1.70078E+16	1.70078E+16	1.70078E+16	1.70078E+16	1.70078E+16	1.70075E+16	7.62299E+17	1.60745E+16		
11	873098500	7.62301E+17	7.62301E+17	7.62301E+17	7.62301E+17	7.62301E+17	7.62299E+17	8.53961E+19	6.94192E+17		
12	9241003900	8.53962E+19	8.53962E+19	8.53962E+19	8.53962E+19	8.53962E+19	8.53961E+19	1.38457E+22	7.67727E+19		
13	1.17668E+11	1.38457E+22	1.38457E+22	1.38457E+22	1.38457E+22	1.38457E+22	1.38457E+22	67108864	1.24191E+22	Min	
Sum E^2		1.39319E+22	1.39319E+22	1.39319E+22	1.39319E+22	1.39319E+22	1.39319E+22	1.39319E+22	1.24966E+22	1.24966E+22	Ω(n) = n!

Cities = n	Average	1	log(n)	n	nLog n	n^2	n^3	2^n	n!
3	72920	5317180561	5317256817	5316888889	5317117652	5316013921	5316013921	5316159744	5316451396
4	142340	20260390921	20260504206	20259536896	20259990028	20256120976	20256120976	20256120976	20253843856
5	735519.9	5.40988E+11	5.40988E+11	5.40982E+11	5.40984E+11	5.40953E+11	5.40806E+11	2.24981E+12	5.40813E+11
6	1499970	2.24991E+12	2.24991E+12	2.24989E+12	2.2499E+12	2.2498E+12	2.24926E+12	3.05601E+13	2.24775E+12
7	5528180	3.05608E+13	3.05608E+13	3.05607E+13	3.05607E+13	3.05602E+13	3.0557E+13	1.05996E+14	3.05051E+13
8	10295540	1.05998E+14	1.05998E+14	1.05998E+14	1.05998E+14	1.05997E+14	1.05988E+14	1.08581E+15	1.0517E+14
9	32951890	1.08583E+15	1.08583E+15	1.08583E+15	1.08583E+15	1.08582E+15	1.08578E+15	1.94899E+16	1.06204E+15
10	139606740	1.949E+16	1.949E+16	1.949E+16	1.949E+16	1.949E+16	1.94898E+16	9.1329E+17	1.849E+16
11	955662919.9	9.13292E+17	9.13292E+17	9.13292E+17	9.13292E+17	9.13291E+17	9.13289E+17	9.80622E+19	8.38591E+17
12	9902638940	9.80623E+19	9.80623E+19	9.80623E+19	9.80623E+19	9.80623E+19	9.80622E+19	1.4381E+22	8.88049E+19
13	1.19921E+11	1.4381E+22	1.4381E+22	1.4381E+22	1.4381E+22	1.4381E+22	1.4381E+22	67108864	1.29263E+22
Sum E^2		1.448E+22	1.448E+22	1.448E+22	1.448E+22	1.448E+22	1.448E+22	1.448E+22	1.3016E+22

Min	
1.3016E+22	Θ(n) = n!

Cities = n	Average	1	log(n)	n	nLog n	n^2	n^3	2^n	n!
3	127900	16358154201	16358287953	16357642609	16358043859	16356107881	16356107881	16356363664	16356875236
4	199700	39879690601	39879849538	39878492416	39879128155	39873699856	39873699856	39873699856	39870504976
5	1475900	2.17828E+12	2.17828E+12	2.17827E+12	2.17827E+12	2.17821E+12	2.17791E+12	4.14598E+12	2.17793E+12
6	2036200	4.14611E+12	4.14611E+12	4.14609E+12	4.14609E+12	4.14596E+12	4.14523E+12	4.98879E+13	4.14318E+12
7	7063200	4.98888E+13	4.98888E+13	4.98887E+13	4.98887E+13	4.98881E+13	4.98839E+13	1.28196E+14	4.98176E+13
8	11322500	1.28199E+14	1.28199E+14	1.28199E+14	1.28199E+14	1.28198E+14	1.28187E+14	1.61584E+15	1.27288E+14
9	40197800	1.61586E+15	1.61586E+15	1.61586E+15	1.61586E+15	1.61586E+15	1.6158E+15	2.56794E+16	1.58682E+15
10	160248300	2.56795E+16	2.56795E+16	2.56795E+16	2.56795E+16	2.56795E+16	2.56792E+16	1.08353E+18	2.45297E+16
11	1040929700	1.08353E+18	1.08353E+18	1.08353E+18	1.08353E+18	1.08353E+18	1.08353E+18	1.26618E+20	1.00203E+18
12	11252484200	1.26618E+20	1.26618E+20	1.26618E+20	1.26618E+20	1.26618E+20	1.26618E+20	1.52948E+22	1.16068E+20
13	1.23672E+11	1.52948E+22	1.52948E+22	1.52948E+22	1.52948E+22	1.52948E+22	1.52948E+22	67108864	1.37933E+22
Sum E^2		1.54225E+22	1.54225E+22	1.54225E+22	1.54225E+22	1.54225E+22	1.54225E+22	1.54225E+22	1.39104E+22

Min	
1.39104E+22	O(n) = n!

Dynamic Programming Algorithm

Cities	Trail												
	trail 1	trail 2	trail 3	trail 4	trail 5	trail 6	trail 7	trail 8	trail 9	trail 10	Best	Average	Worst
3	41200	114400	73000	41300	69400	41100	73400	42800	89900	41999	41100	62849.9	114400
4	172700	131000	261200	127100	172800	124800	207100	192100	189100	135600	124800	171350	261200
5	440600	631800	444500	510100	499700	1182500	379700	492600	682200	950400	379700	621410	1182500
6	1016500	1299800	1225500	872301	897900	924700	694100	882200	826300	737700	694100	937700.1	1299800
7	2337000	3660100	2342200	1607500	2013100	1659800	2320300	2244200	1803200	2356600	1607500	2234400	3660100
8	5899300	4664600	4824000	4236700	5150700	4426600	4642000	5483500	5090900	4892400	4236700	4931070	5899300
9	8369600	9947800	4826800	4687400	6387600	6292500	6351700	6673000	6914600	6256200	4687400	6670720	9947800
10	12933000	9263400	8729900	6720700	8553500	6416700	7114500	7035200	7678100	8364500	6416700	8280950	12933000
11	11601200	16940400	9667000	15027200	11378700	12780600	14581000	15451500	15275800	14120000	9667000	13682340	16940400
12	21380800	24345000	18769200	18562700	17786500	17497800	19914200	27654600	21191400	18581500	17497800	20568370	27654600
13	32270300	53757900	36483900	44493100	37195000	34343300	44917900	32588100	30562800	30087800	30087800	37670010	53757900

Cities = n	Best	1	log(n)	n	nLog n	n^2	n^3	2^n	n!
3	41100	1689127801	1689170781	1688963409	1689092344	1688470281	1688470281	1688552464	1688716836
4	124800	15574790401	15574889726	15574041616	15574438909	15571046656	15571046656	15571046656	15569050176
5	379700	1.44171E+11	1.44172E+11	1.44168E+11	1.44169E+11	1.44153E+11	1.44077E+11	4.8173E+11	1.44081E+11
6	694100	4.81773E+11	4.81774E+11	4.81766E+11	4.81768E+11	4.81725E+11	4.81475E+11	2.58385E+12	4.80776E+11
7	1607500	2.58405E+12	2.58405E+12	2.58403E+12	2.58404E+12	2.5839E+12	2.58295E+12	1.79485E+13	2.56788E+12
8	4236700	1.79496E+13	1.79496E+13	1.79496E+13	1.79496E+13	1.79491E+13	1.79453E+13	2.19693E+13	1.76096E+13
9	4687400	2.19717E+13	2.19717E+13	2.19716E+13	2.19716E+13	2.1971E+13	2.19649E+13	4.11675E+13	1.87015E+13
10	6416700	4.1174E+13	4.1174E+13	4.11739E+13	4.11739E+13	4.11728E+13	4.11612E+13	9.34311E+13	7.77239E+12
11	9667000	9.34509E+13	9.34509E+13	9.34507E+13	9.34507E+13	9.34485E+13	9.34252E+13	3.06101E+14	9.1505E+14
12	17497800	3.06173E+14	3.06173E+14	3.06173E+14	3.06173E+14	3.06168E+14	3.06113E+14	9.05029E+14	2.12986E+17
13	30087800	9.05276E+14	9.05276E+14	9.05275E+14	9.05275E+14	9.05266E+14	9.05144E+14	67108864	3.8402E+19
Sum E^2		1.38922E+15	1.38922E+15	1.38922E+15	1.38922E+15	1.3892E+15	1.38898E+15	1.38873E+15	3.86159E+19

Min	
1.38873E+15	Ω(n) = 2^n

Cities = n	Average	1	log(n)	n	nLog n	n^2	n^3	2^n	n!
3	62849.9	3949984231	3950049956	3949732840	3949930010	3948978713	3948978713	3949104396	3949355767
4	171350	29360479801	29360616174	29359451716	29359997202	29355339556	29355339556	29355339556	29352598276
5	621410	3.86149E+11	3.8615E+11	3.86144E+11	3.86146E+11	3.86119E+11	3.85995E+11	8.79221E+11	3.86001E+11
6	937700.1	8.7928E+11	8.7928E+11	8.7927E+11	8.79273E+11	8.79214E+11	8.78876E+11	4.99226E+12	8.77932E+11
7	2234400	4.99254E+12	4.99254E+12	4.99251E+12	4.99252E+12	4.99232E+12	4.99101E+12	2.43142E+13	4.97005E+12
8	4931070	2.43154E+13	2.43154E+13	2.43154E+13	2.43154E+13	2.43148E+13	2.43104E+13	4.44951E+13	2.39194E+13
9	6670720	4.44985E+13	4.44985E+13	4.44984E+13	4.44984E+13	4.44974E+13	4.44888E+13	6.85657E+13	3.97888E+13
10	8280950	6.85741E+13	6.85741E+13	6.8574E+13	6.8574E+13	6.85725E+13	6.85576E+13	1.87178E+14	2.16425E+13
11	13682340	1.87206E+14	1.87206E+14	1.87206E+14	1.87206E+14	1.87203E+14	1.8717E+14	4.22974E+14	6.88247E+14
12	20568370	4.23058E+14	4.23058E+14	4.23057E+14	4.23057E+14	4.23052E+14	4.22987E+14	1.41872E+15	2.10161E+17
13	37670010	1.41903E+15	1.41903E+15	1.41903E+15	1.41903E+15	1.41902E+15	1.41886E+15	67108864	3.83081E+19
Sum E^2		2.17297E+15	2.17297E+15	2.17297E+15	2.17297E+15	2.17295E+15	2.17267E+15	2.17215E+15	3.8519E+19

Min	
2.17215E+15	Θ(n) = 2^n

Cities = n	Average	1	log(n)	n	nLog n	n^2	n^3	2^n	n!
3	114400	13087131201	13087250835	13086673609	13087032506	13085300881	13085300881	13085529664	13085987236
4	261200	68224917601	68225125484	68223350416	68224181941	68217081856	68217081856	68217081856	68212902976
5	1182500	1.3983E+12	1.3983E+12	1.39829E+12	1.3983E+12	1.39825E+12	1.39801E+12	1.6894E+12	1.39802E+12
6	1299800	1.68948E+12	1.68948E+12	1.68946E+12	1.68947E+12	1.68939E+12	1.68892E+12	1.33959E+13	1.68761E+12
7	3660100	1.33963E+13	1.33963E+13	1.33963E+13	1.33963E+13	1.3396E+13	1.33938E+13	3.48002E+13	1.33595E+13
8	5899300	3.48017E+13	3.48017E+13	3.48016E+13	3.48017E+13	3.4801E+13	3.47957E+13	9.89536E+13	3.43276E+13
9	9947800	9.89587E+13	9.89587E+13	9.89585E+13	9.89586E+13	9.89571E+13	9.89442E+13	1.67249E+14	9.18707E+13
10	12933000	1.67262E+14	1.67262E+14	1.67262E+14	1.67262E+14	1.6726E+14	1.67237E+14	2.86942E+14	8.65681E+13
11	16940400	2.86977E+14	2.86977E+14	2.86977E+14	2.86977E+14	2.86973E+14	2.86932E+14	7.64664E+14	5.27915E+14
12	27654600	7.64777E+14	7.64777E+14	7.64776E+14	7.64776E+14	7.64769E+14	7.64681E+14	2.88947E+15	2.03714E+17
13	53757900	2.88991E+15	2.88991E+15	2.88991E+15	2.88991E+15	2.88989E+15	2.88968E+15	67108864	3.81092E+19
Sum E^2		4.25925E+15	4.25925E+15	4.25925E+15	4.25925E+15	4.25922E+15	4.25883E+15	4.25725E+15	3.83136E+19

Min	
4.25725E+15	O(n) = 2^n

7. Refrences

- GeeksforGeeks. (2022, June 5). *What is algorithm and why analysis of it is important*. https://www.geeksforgeeks.org/what-is-algorithm-and-why-analysis-of-it-is-important/
- Walker, A. (2022, December 19). *Travelling Salesman Problem: Python, C++ Algorithm*. Guru99. https://www.guru99.com/travelling-salesman-problem.html
- Levitin, A. (2023). *Introduction to the Design and Analysis of Algorithms 3th (third) edition*.