//Kamal Giri
//COSC 3020
//Spring2023
//Assignment03
//Last Modified: 04/28/2023

1.Dynamic Programming
Held- Karp Algorithms.

Asymptotic Analysis:
Worst Time Complexity
The worst time complexity of the this algorithm is O(n^2*(2*n) *n). Here, n^2 is for the iteration through distance matrix. We have to go through each element of the matrix so, worst will be 2^n. Now, 2^n is for the all the possible subsets of the cities that salesman can visit. Whereas, for each subset and starting city, we have to find the distance to all other cities which will give n.

Worst Space Complexity
Memoization takes significant space in this algorithm. Hence , the worst will be O(n*2^n) as we have 2^n possible subsets of city to store and for each city there is n starting cities as well as we have n^2 for distant matrix but they are all asymptotically insignificant compared to O(n*2^n).

The Output for multiple input size:

The Shortest Path for input : 3
When size of input is 3
Running Time:: 9.583ms

The Shortest Path for input : 25
When size of input is 4
Running Time:: 9.752ms

The Shortest Path for input : 13
When size of input is 7
Running Time:: 10.273ms

The Shortest Path for input : 21
When size of input is 10
Running Time:: 14.035ms

The Shortest Path for input : 25

When size of input is 13
Running Time:: 15.62ms

The Shortest Path for input : 23
When size of input is 15
Running Time:: 26.523ms

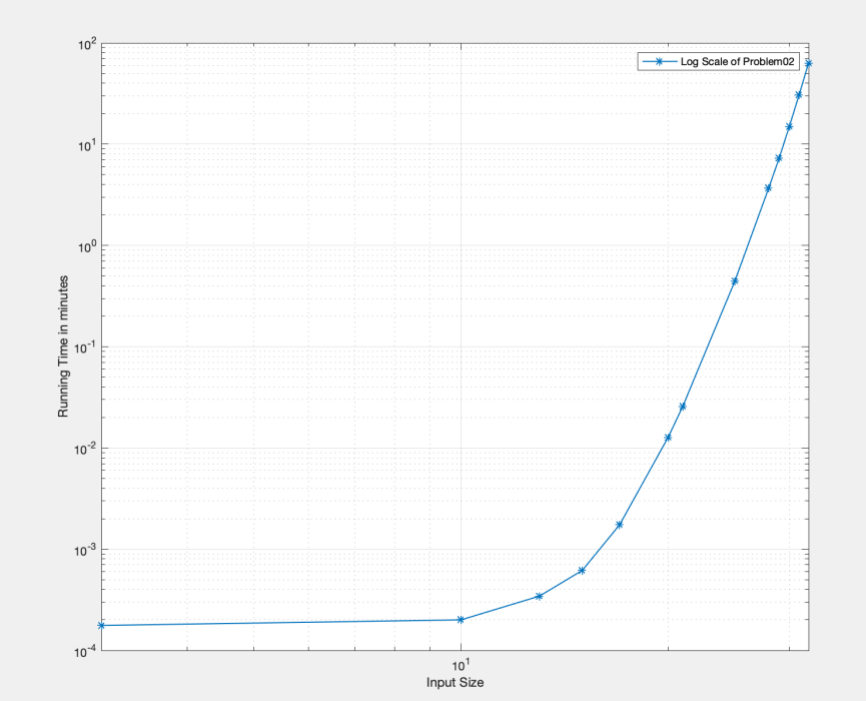
The Shortest Path for input : 21
When size of input is 17
Running Time:: 104.533ms

The Shortest Path for input : 30
When size of input is 20
Running Time:: 1.509s

The Shortest Path for input : 27
When size of input is 21
Running Time:: 3.878s

The Shortest Path for input : 34
When size of input is 22
Running Time:: 9.846s

And the graph:



Looking at the graph, the algorithms seems running as expect. Since, the code has average time complexity of around O(n^2*2^n), the code will take significant amount of time with the increment of the input size which we can see in the above logarithmic scaled graph.

1. Stochastic Local Search
   The 2-opt Algorithm for solving the Travelling Salesperson Problem.

   Asymptotic Complexity

   a. Worst Time Complexity

      The worst time complexity of this code is O(2*n *n^2). This might be slightly better than the previous algorithm but they are somewhat similar. The worst complexity comes from the 2*n which is the stopping condition and n^2 as it performs n^2 operation in each while loop.

   b. Worst Space Complexity
      SO, the worst memory complexity is O(n^2) which is the the space complexity for creating the adjacency matrix and other than that it is just O(n) which is much better than the first algorithms.

Stopping Criteria:
I choose the value of 2^n inside the while loop for the stopping criteria. So, the code will try to find 2^n out of (n-1)! Permutations and calculate the distance of each permutation. I choose 2^n, because there is the good chance that the permutation with the shortest distance may lies inside that 2^n permutation.

i and k:
I used JavaScript random number generator to numbers i and k. For i, generated number from max n-1 to minimum 1 and then for j, generated the number with max as n and min as the i which we have already calculate above such that j>i for all case. I think this is efficient in getting unique i and j for unique permutation.

Output:
Shortest Distance : 3
When size of matrix is = 3
Running Time: 10.592ms

Shortest Distance : 24
When size of matrix is = 10
Running Time: 12.027ms

Shortest Distance : 35
When size of matrix is = 13
Running Time: 20.699ms

Shortest Distance : 32
When size of matrix is = 15
Running Time: 36.986ms

Shortest Distance : 37
When size of matrix is = 17
Running Time: 104.491ms

Shortest Distance : 44
When size of matrix is = 20
Running Time: 764.203ms

Shortest Distance : 53
When size of matrix is = 21
Running Time: 1.542s

Shortest Distance : 57
When size of matrix is = 25
Running Time: 26.635s

Shortest Distance : 65
When size of matrix is = 28
Running Time: 3:41.597 (m:ss.mmm)

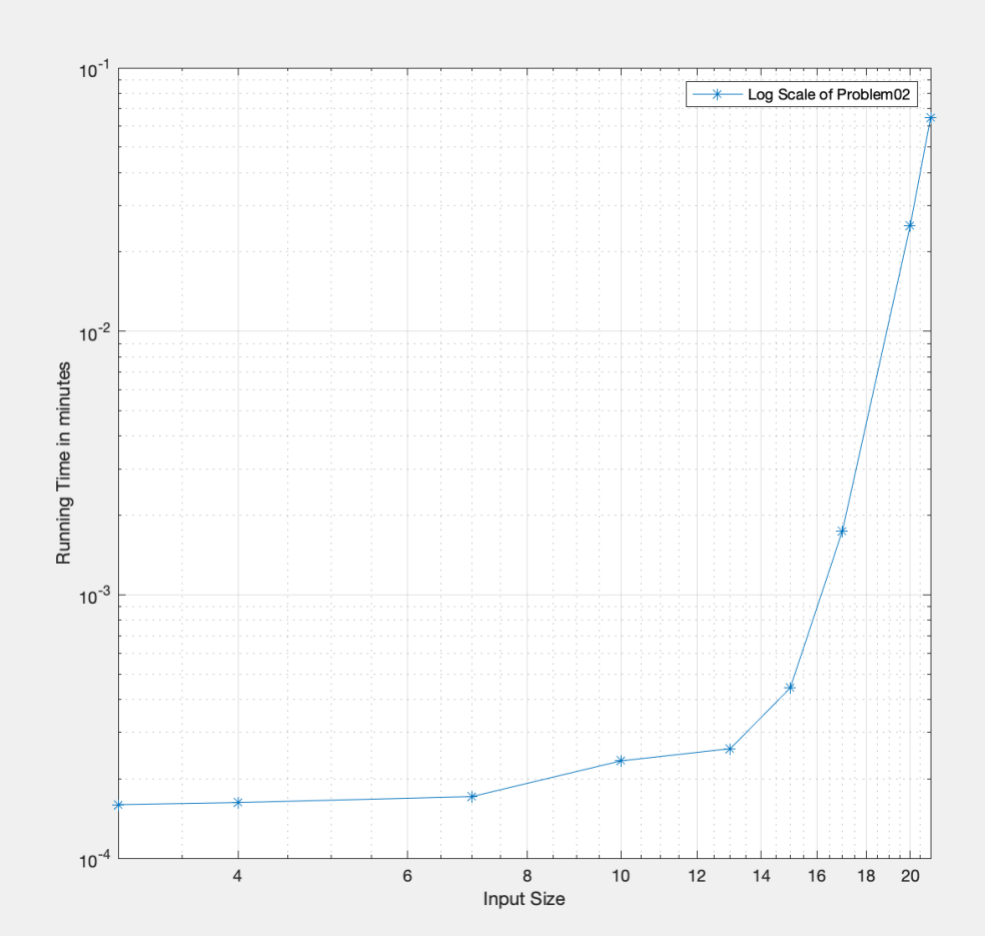
Shortest Distance : 74
When size of matrix is = 29
Running Time: 7:23.659 (m:ss.mmm)

Shortest Distance : 75
When size of matrix is = 30
Running Time: 14:56.413 (m:ss.mmm)

Shortest Distance : 81
When size of matrix is = 31
Running Time: 30:41.166 (m:ss.mmm)

Shortest Distance : 77
When size of matrix is = 32
Running Time: 1:03:25.810 (h:mm:ss.mmm)

Graph:



This has also similar running time as the previous algorithms. This seems as expected as the complexity increase and the input size increases.

2. Comparison

The combined graph is:

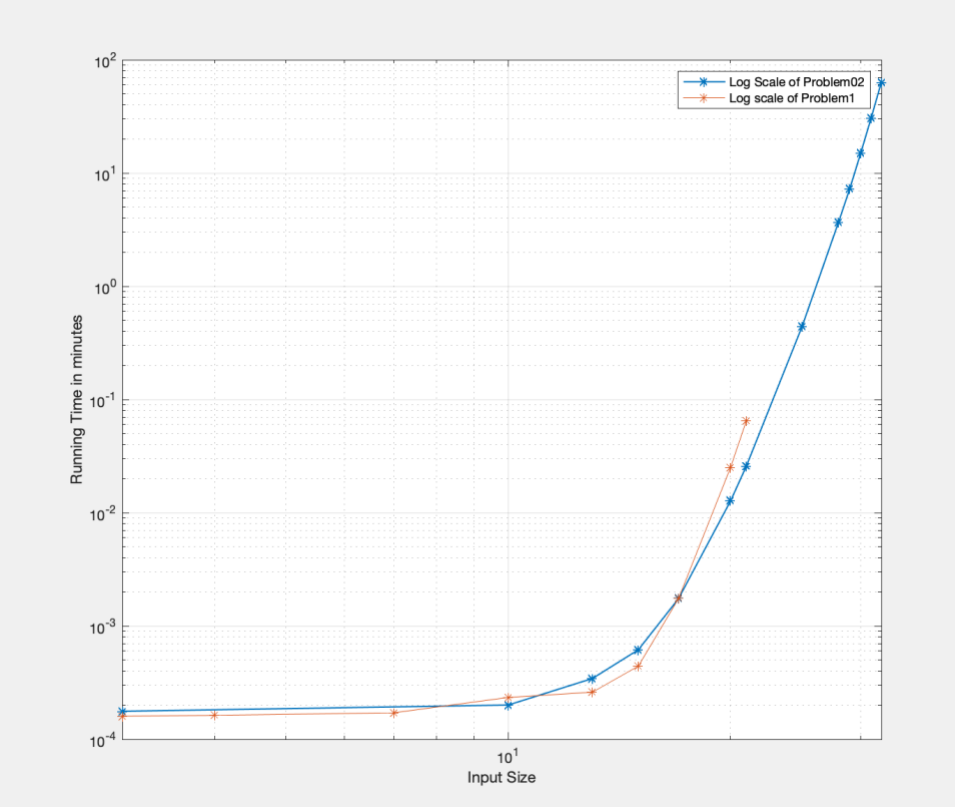


Fig 3.1

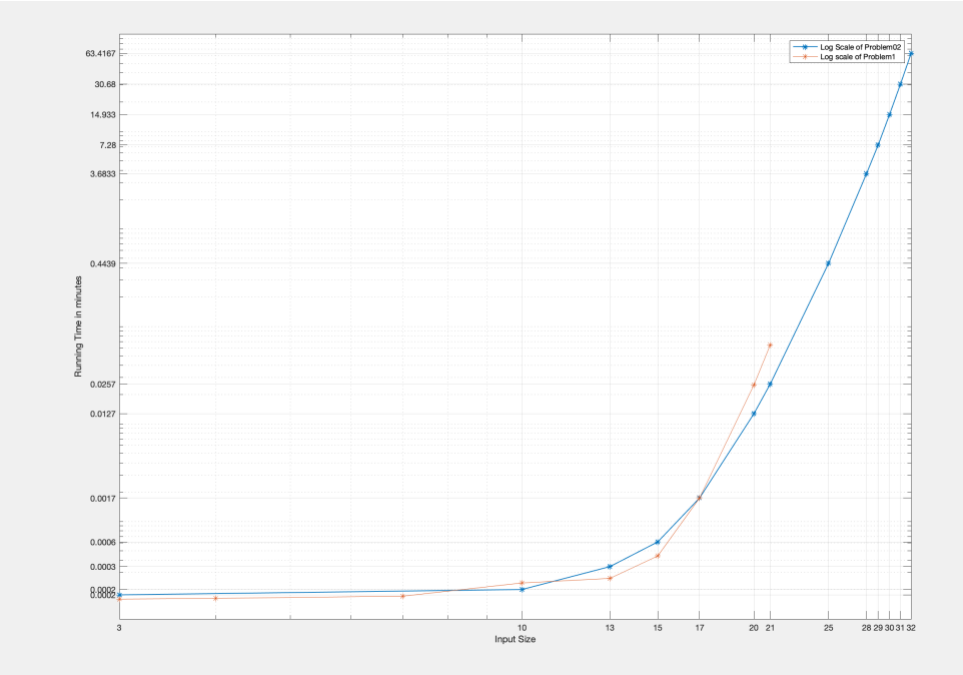


Fig3.2

Code for above graph using matlab:

```
clear all;
format long;

%need to divide by mi
mi = 60000;
%For Problem 2
xplot = [3,10,13,15,17,20,21,25,28,29,30,31,32];
yplot = [10.592/60000, 12.027/60000,20.63/mi 36.986/60000,104.491/mi 764.62/60000 ,1.542/60,
26.635/60, 3.68333,7.28,14.933,30.68 63.41666667];

%For Problem 1

xplot1 = [3,4,7,10,13,15,17,20,21];
yplot1 = [9.58/mi,9.752/mi,10.273/mi, 14.035/mi, 15.62/mi, 26.52/mi, 104.53/mi, 1.509/60, 3.873/60 ];

loglog(xplot, yplot, '- *', 'LineWidth',1);
hold
loglog(xplot1,yplot1, '-*');
grid on;

xlabel("Input Size");
ylabel("Running Time in minutes");
xticks(xplot);
yticks(yplot);

legend("Log Scale of Problem02","Log scale of Problem1" );
```

Comparison:

Doing the direct comparison between 2 algorithms using the graph, we can see that they are not differing by much at least for first around 22 inputs.

Since, the average time complexity for both algorithms may have been similar, while looking at the graph data, the 2-opt algorithm seems to be running little slower at first and there first overlay around input size 7 and another around input size 11 and finally we can see that 2-opt is taking less time than Held-karp

from input size 17. This might not be actually true for all the input size after input size 17; however, we can conclude for the given input size than 2-opt is performing better than the Held-karp as well as it is not suffering from large memory complexity.

As for finding the shortest length for both algorithms we have used different approach. For Held-karp , we have use recursion with memorization,or dynamic programming. We start with starting city and find the distance to the all other cities.
There is very simple formula for this
$g(i,s) = \min\{Cik + g(k,s – \{k\})\}$

where, I is the starting city,

k is next city

s = set of cities except the starting city

Cik is the distance between city I and k

G(I,s) is fuction to get the smallest distance.

Hence, I used the same approach using the recursion and memorization which stores the already calculated distance to get the shortest distance.
As for the 2-opt algorithm, it is the iterative process which uses while loop to get the random permutations and for each each permutation, we calculate the distance of the cities and keep the minimum distance at the end after stopping criteria is implemented.

So, first algorithms use the recursion to get the shortest distance where it has to visit every city to get the shortest path. However, 2nd algorithms use the random permutation which may not generate the smallest path permutation in some case giving bad result.