Ayhan Okuyan
21601531
CS201-1

CS201 Homework Assignment 2

Time Complexity on the Maximum Subsequence Sum Problem

1- This homework assignment consisted of us implementing the Maximum Subsequence Sum Problem on C++ with four different algorithms, each with a different time complexity. I have created a main driver in order to execute each of the algorithms for any given amount of array size. Since there is high amount of variation in terms of time complexity, from $O(n^3)$ to $O(n)$, I have picked a moderate interval to show them on one plot, which is the interval between 1 and 201, I have inspected the elapsed time in each 5 element addition to the array. Hence, I have taken 40 samples each. However, for the table construction I have used a higher range as the numbers are easily distinguishable with respect to the plots. The comparison table that I have constructed is given below.

Array Size	Algorithm 1 (ms)	Algorithm 2 (ms)	Algorithm 3 (ms)	Algorithm 4 (ms)
1	0.000857	0. 00083	0.00053	0.00054
10	0.004049	0.000381	0.019407	0.000142
20	0.015734	0.000698	0.001659	0.000192
40	0.095525	0.001986	0.002693	0.000315
80	0.711265	0.00659	0.004724	0.000581
100	1.37914	0.009634	0.006031	0.000541
200	2.7631	0.036831	0.012227	0.001225
300	7.34293	0.081524	0.018442	0.001219
400	16.6351	0.143935	0.024305	0.001427
600	54.2655	0.320855	0.037566	0.001866
800	125.327	0.568569	0.049302	0.002836
1000	243.163	0.984598	0.058685	0.002722
1200	424.311	1.6165	0.069645	0.003413
1400	662.281	1.84326	0.084324	0.004055
1600	992.46	2.29211	0.096539	0.004254
1800	1414.78	3.16418	0.107345	0.004926
2000	1927.92	3.50035	0.115053	0.005277
2400	3347.25	5.0682	0.146957	0.006687
3200	7971.52	8.93739	0.198189	0.023647
4000	15410.4	14.3401	0.312791	0.009151

Figure 1 – Comparison Table

Looking at the graph, it is easier to see Algorithm 1 representing the $O(n^3)$ solution, Algorithm 2 representing $O(n^2)$, Algorithm 3 representing the $O(n*\log(n))$ solution and the Algorithm 4 representing the $O(n)$ solution. The graph for this data is shown below.

Ayhan Okuyan
21601531
CS201-1

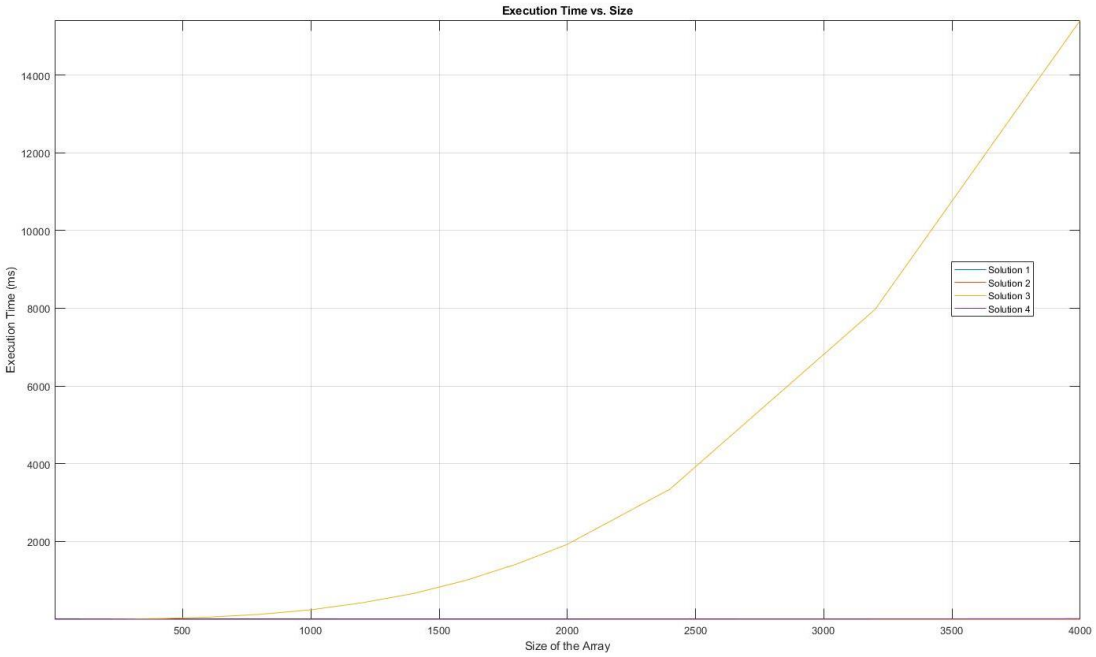

Figure 2 – Plot of the Comparison Table

From here, we can see that Algorithm 1's complexity being so dominant, it suppresses the other outputs and the other solutions are harder to distinguish in between. Hence, the other graph that I have constructed is given below.

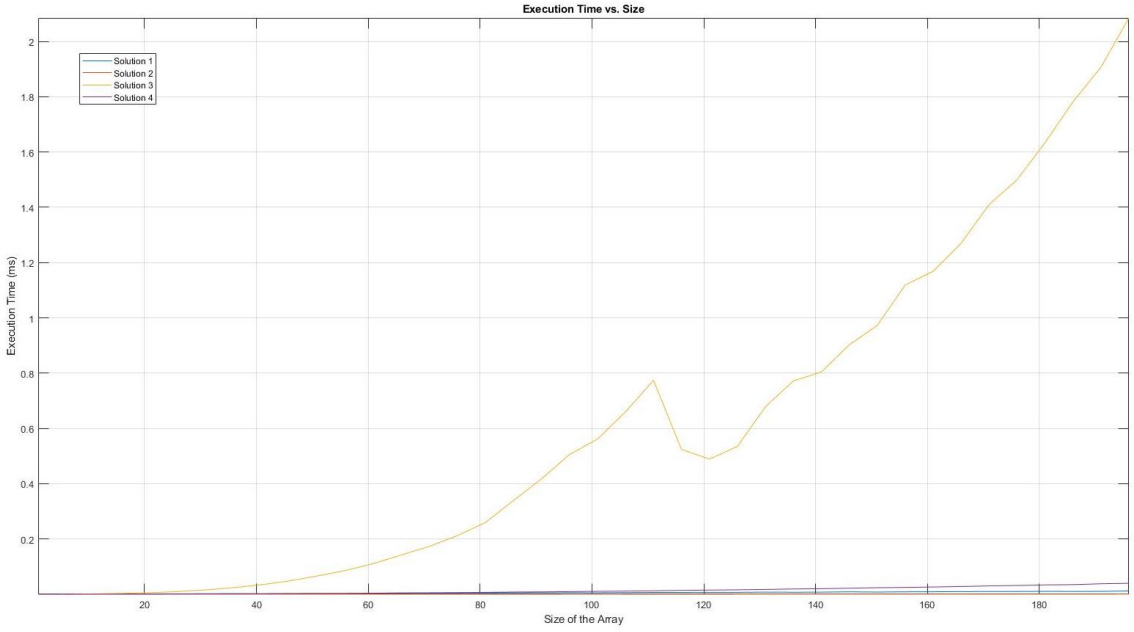

Figure 3 – Time vs Array Size Plot for all Algorithms in the interval 0-200

Here, although the data is more unstable, the algorithms are more distinguishable. We see the difference between the quadratic algorithm and the logarithmic one better. Also, we can see that the linear algorithm being almost constant throughout.

2- For this part, in order to see each algorithm better separately, I have used different intervals for each of the algorithms with smaller step sizes to see the outcome better.
For Algorithm 1, I have changed N in between 0 and 1000 with a step size of 50. The obtained data and the approximation are given below. Since, we know the time complexity comes with n^3, we can approximate the line by choosing an exemplary point on the data and finding the finite constant at the beginning. In this case, I have chosen the point to be ~ (900,2000), hence the constant turned out to be 2.74e-6.

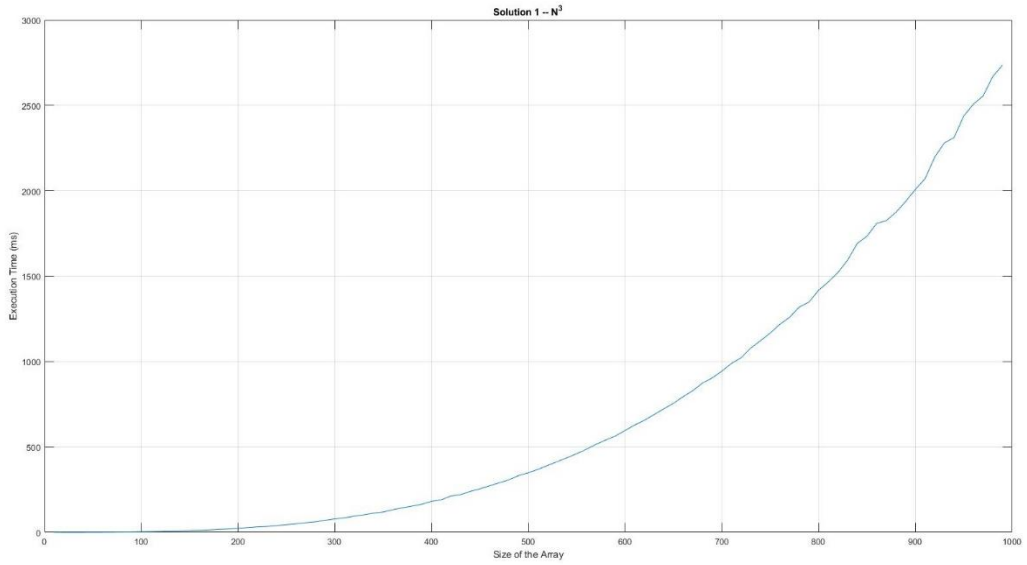

Figure 4 – Time vs Array Size Plot for Algorithm 1

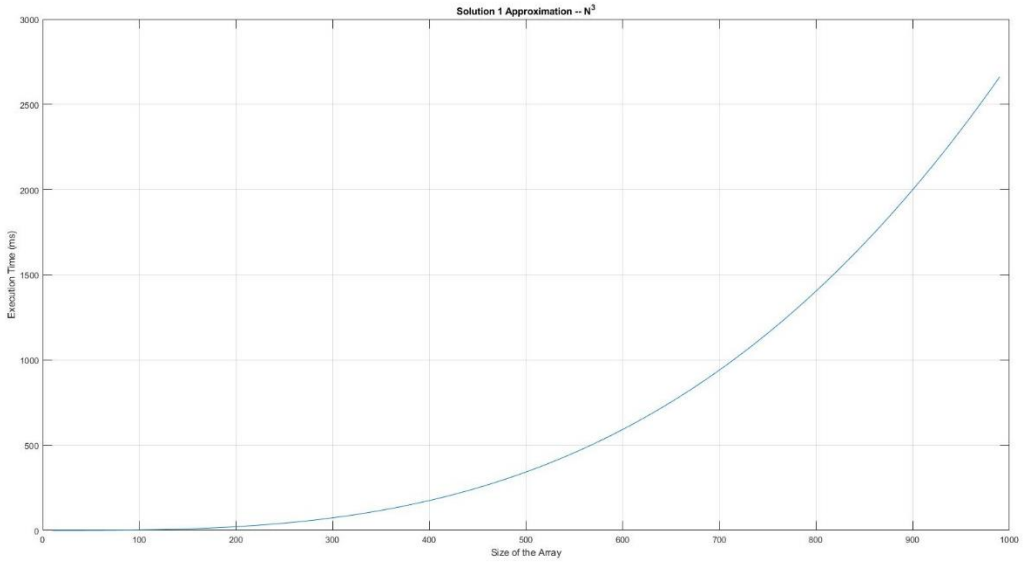

Figure 5 – Approximation for Algorithm 1

Ayhan Okuyan
21601531
CS201-1

For Algorithm 2, I went through the same procedure with what I have done with Algorithm 1, by arranging the x axis from 1 to 1000 with 50 step size. Then I have picked the point ~ (800,6.8) and found the constant as 0.000010625. The plots are given below.

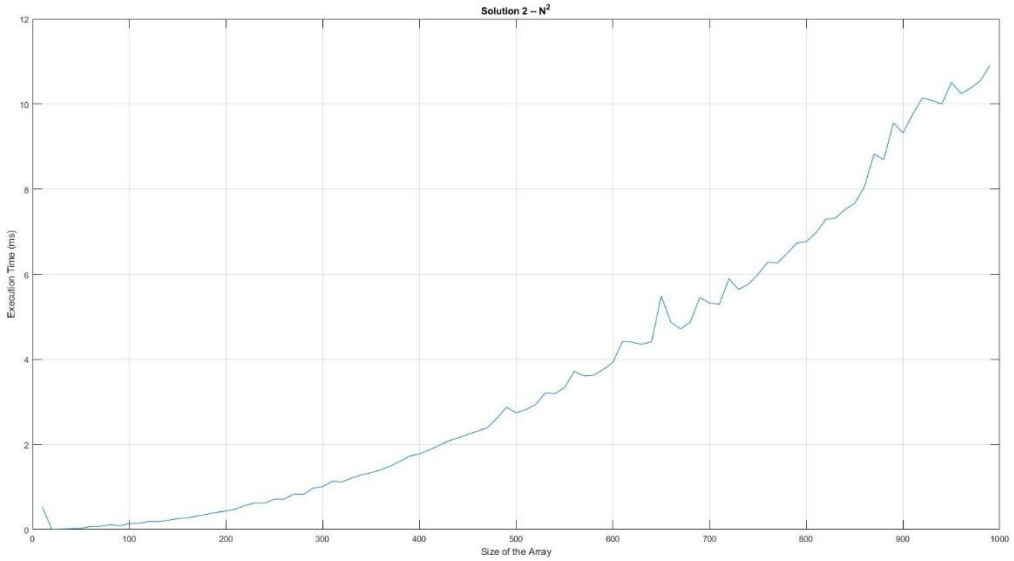

Figure 6 – Time vs Array Size Plot for Algorithm 2

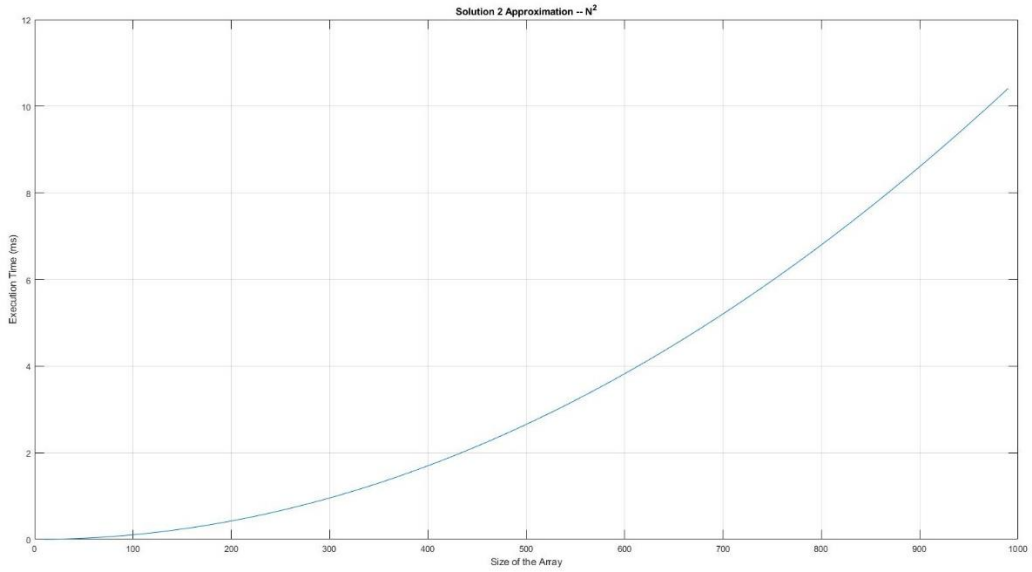

Figure 7 – Approximation for Algorithm 2

For the third Algorithm, I have used a different approach, since I know the execution times were not as much as the ones that I have described, for this one I have picked the interval from 1 to 1000000 with step size again 50. Then I have approximated the graph from the point ~ (800000,90) which made the constant to be 1.27e-5. The graphs are given below.

Ayhan Okuyan
21601531
CS201-1

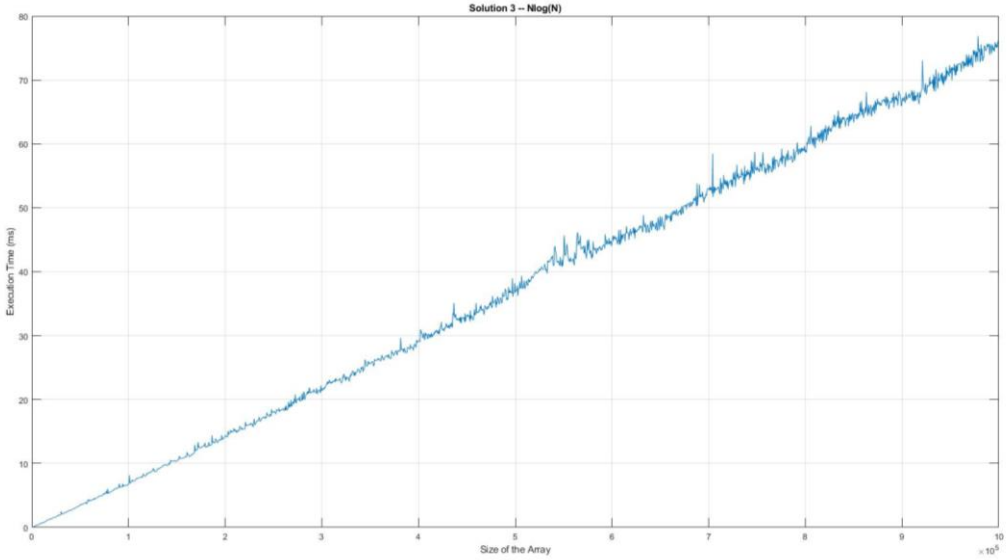

Figure 8 – Time vs Array Size Plot for Algorithm 3

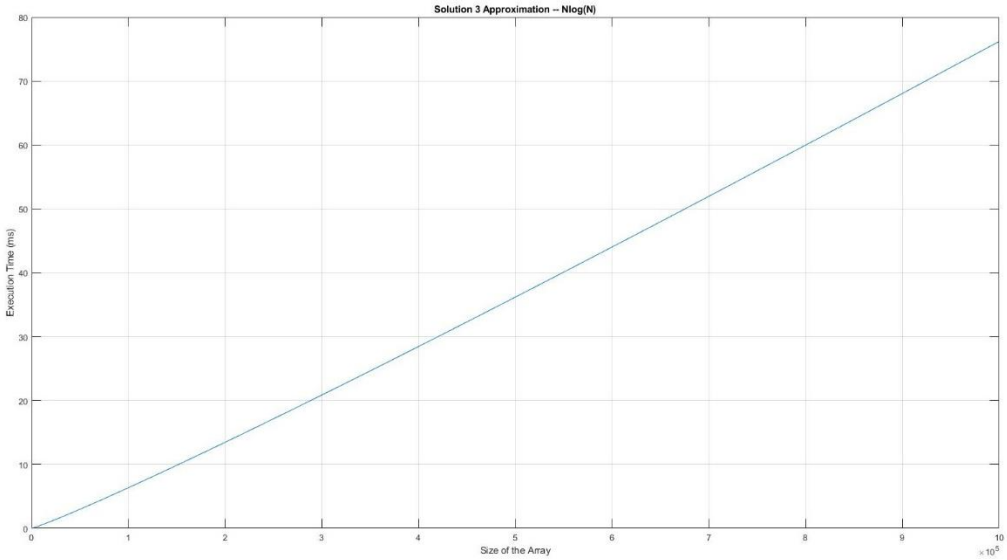

Figure 9 – Approximation for Algorithm 3

For the last Algorithm, I have gone through the same motions with the n*log(n) algorithm in which I have chosen the interval in between 1 and 1000000 with 50 step size. Furthermore, I have picked the sample point ~ (7,1.5) and found the constant to be 0.2142. The data and the output graph are given below.

Ayhan Okuyan
21601531
CS201-1

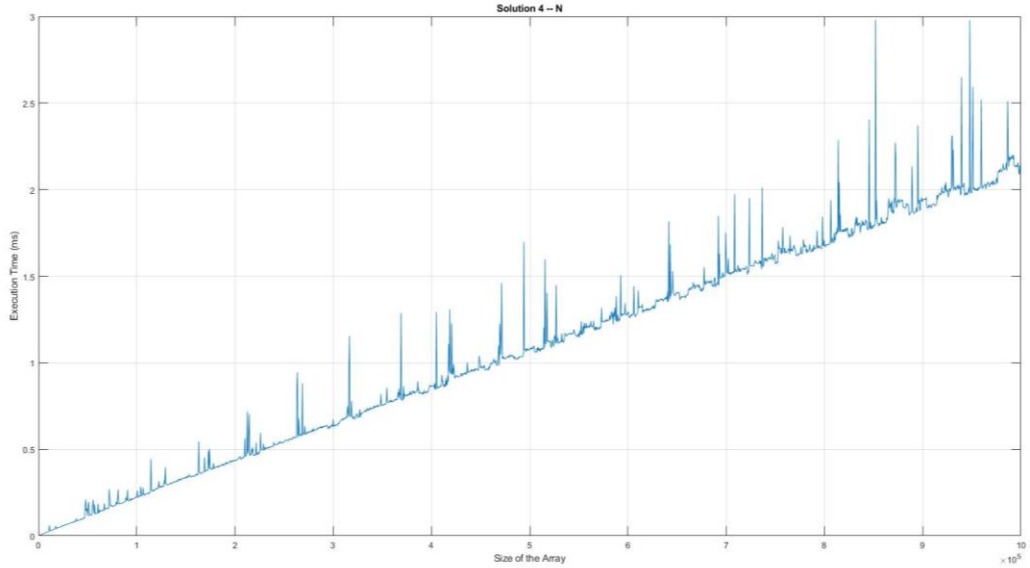

Figure 10 – Time vs Array Size Plot for Algorithm 4

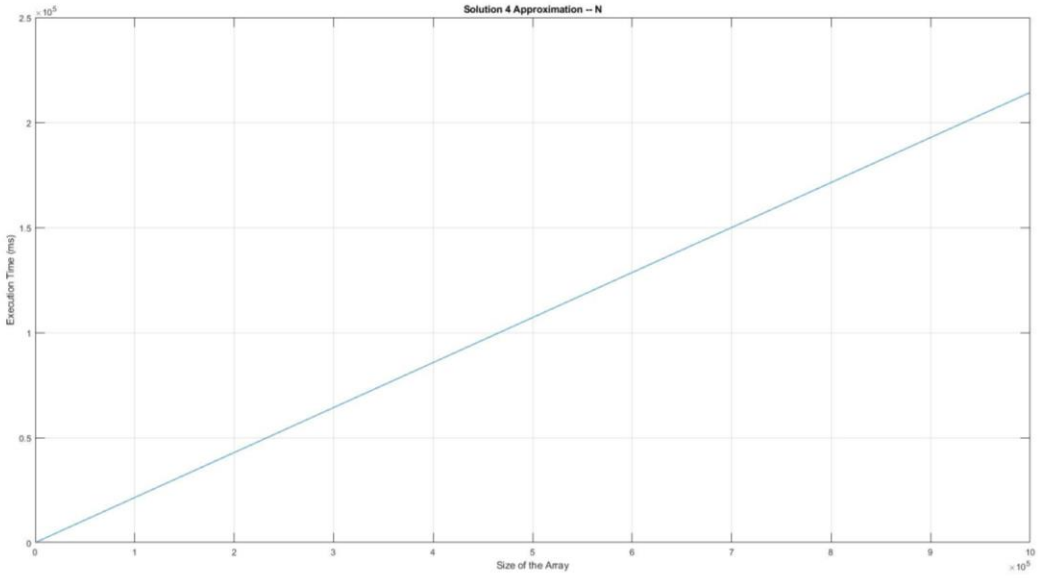

Figure 11 – Approximation for Algorithm 4

By looking at the plot pairs, we can see that the observed growth rates show very high similarity to the expected growth rates. Since we don't use the real constants in the Big-OH notation, we needed to calculate the from the plots and include them into out approximated model. We have naturally observed some noise, since the operating system at the time is multithreading with other up and running applications. Furthermore, I was able to observe the enormous differences between different time complexities and their execution times.

3- For this assignment, I have used my personal computer with the following specifications. The hardware specifications are given below.

Ayhan Okuyan
21601531
CS201-1

Processor: Intel Core i7-8750H CPU @ 2.20GHz
RAM: 16GB
Memory: 256GB SSD – 2TB HDD
Display Card: NVIDIA GTX1060 Max-Q
Operating System: Dual Boot Windows-Linux System

- Windows 10 Home
- Ubuntu GNOME 16.04 LTS (Linux)

This homework assignment has been done on Ubuntu by using VSCode and gcc/g++ version 7. The assignment has been made using chrono class which is a feature added with c++ version 11. So, in order to be able to run the assignment on the Dijkstra server, the command “g++ -std=c++11 main.cpp -o a.out” should be used.