CS201 – Fall 2025

Homework 2: Algorithm Efficiency and Searching

Ali İhsan Sevindi

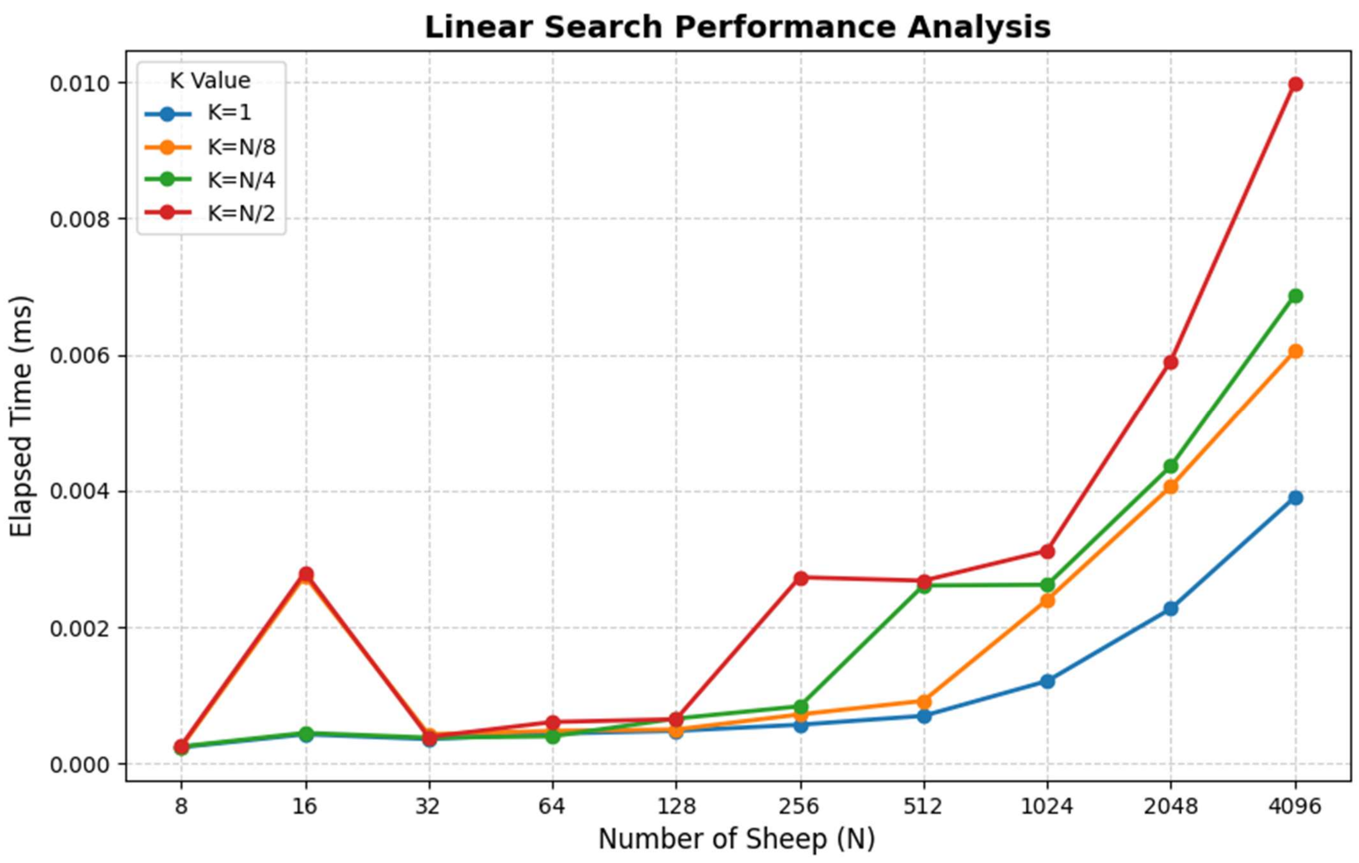
22402055

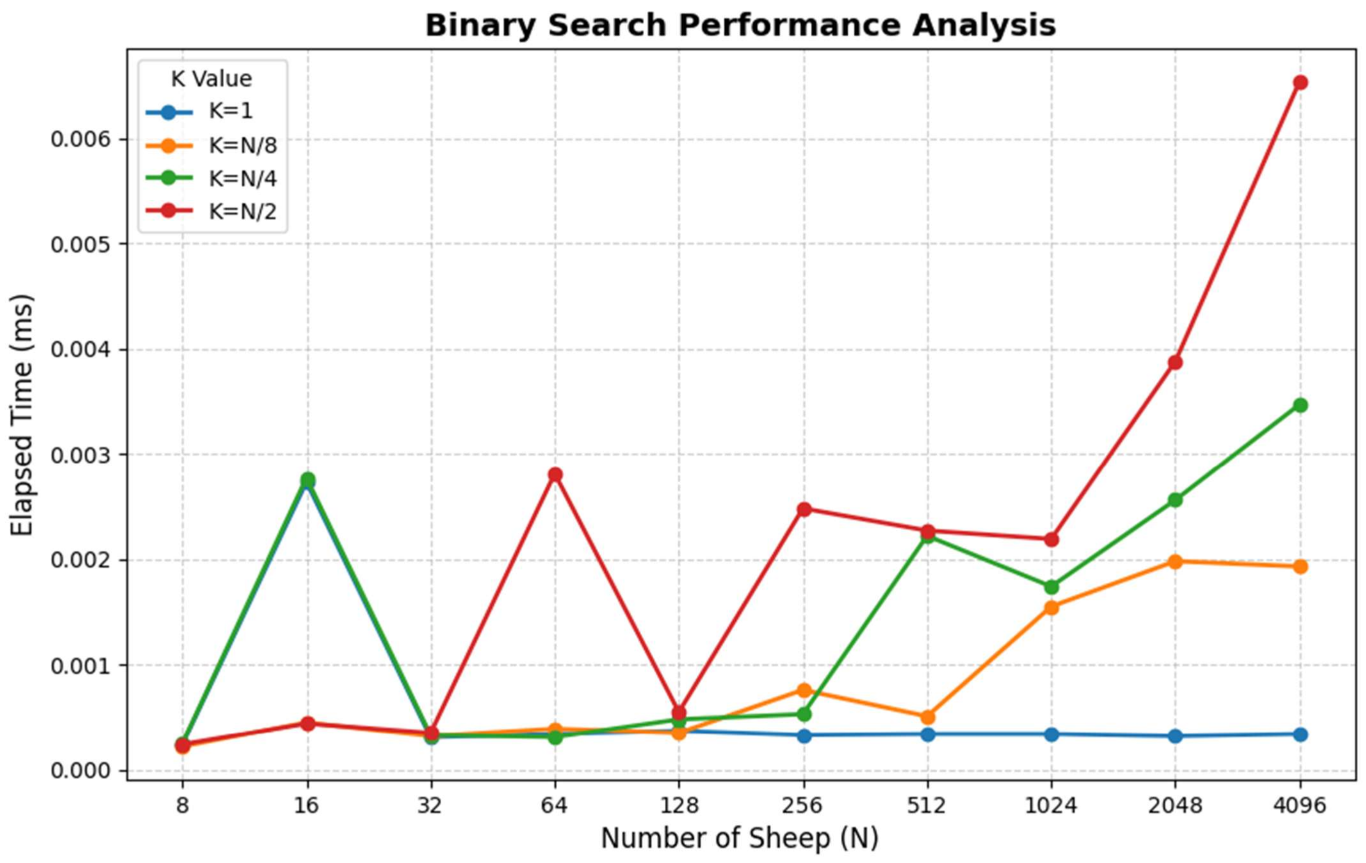
Section: 3

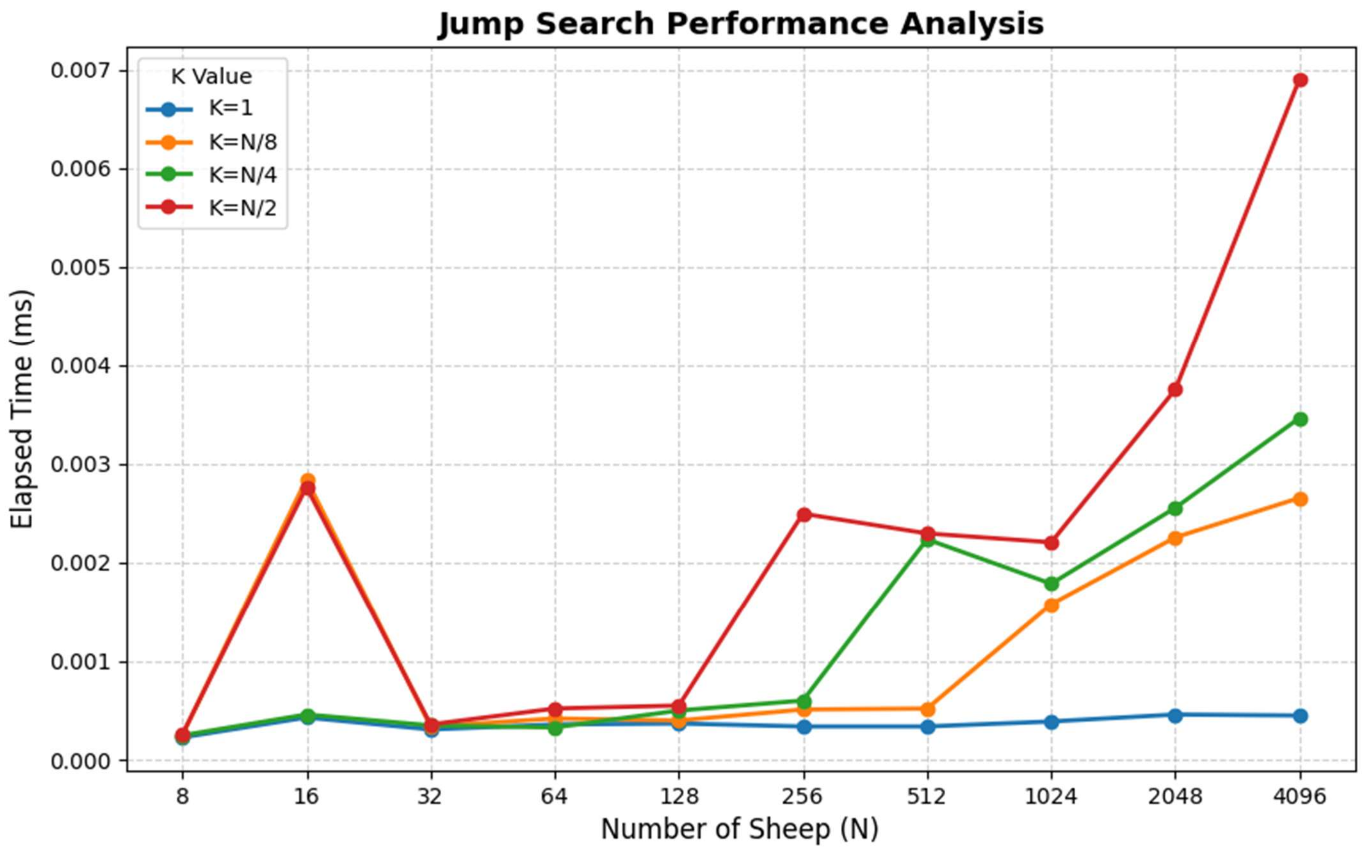
3.4 Results Table

N	Linear Search				Binary Search				Jump Search			
K	K= 1	K= N/8	K= N/4	K= N/2	K= 1	K= N/8	K= N/4	K= N/2	K= 1	K= N/8	K= N/4	K= N/2
8	0.00024	0.00024	0.00025	0.00026	0.00024	0.00022	0.00025	0.00024	0.00023	0.00025	0.00025	0.00026
16	0.00043	0.00275	0.00045	0.0028	0.00273	0.00045	0.00277	0.00044	0.00043	0.00284	0.00046	0.00276
32	0.00036	0.00043	0.00038	0.00039	0.00031	0.00032	0.00033	0.00035	0.00031	0.00034	0.00035	0.00036
64	0.00044	0.00048	0.0004	0.00061	0.00034	0.00039	0.00031	0.00281	0.00036	0.00042	0.00033	0.00052
128	0.00048	0.0005	0.00066	0.00065	0.00037	0.00035	0.00048	0.00055	0.00037	0.0004	0.0005	0.00055
256	0.00057	0.00072	0.00084	0.00273	0.00033	0.00076	0.00053	0.00248	0.00034	0.00051	0.0006	0.00249
512	0.0007	0.00092	0.00261	0.00268	0.00034	0.00051	0.00222	0.00227	0.00034	0.00052	0.00223	0.00229
1024	0.00121	0.0024	0.00262	0.00312	0.00034	0.00155	0.00174	0.00219	0.00039	0.00157	0.00178	0.0022
2048	0.00227	0.00406	0.00436	0.00591	0.00032	0.00198	0.00256	0.00388	0.00046	0.00225	0.00255	0.00375
4096	0.0039	0.00605	0.00687	0.00998	0.00034	0.00193	0.00347	0.00654	0.00045	0.00265	0.00346	0.0069

3.5 Plots




Binary Search Performance Analysis
K Value
K=1
K=N/8
K=N/4
K=N/2
Elapsed Time (ms)
0.000
0.001
0.002
0.003
0.004
0.005
0.006
8
16
32
64
128
256
512
1024
2048
4096
Number of Sheep (N)


Jump Search Performance Analysis
K Value
K=1
K=N/8
K=N/4
K=N/2
Elapsed Time (ms)
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
8
16
32
64
128
256
512
1024
2048
4096
Number of Sheep (N)

4.1 Computer Specifications:

Processor: 13th Gen Intel(R) Core(TM) i9-13900H, 2600 Mhz

Ram: 32 GB

OS: Windows 11 Home

4.2

Linear Search goes through each element in the array one by one and checks if the element in that index equals to the key value. Therefore it does not make a difference whether the array is sorted or not. However Binary Search assumes that the array is sorted in ascending order and finds the key value by comparing it to the middle value and deciding which half should contain it. In a similar way Jump Search assumes that the array is sorted and jumps between blocks of a certain size until it finds a larger value than the key. It then linearly searches through the previous block to find the exact index of key. Hence while Linear Search can work with unsorted arrays, Binary and Jump Search only work with sorted arrays.

4.3

The best case for Linear Search is when the key value is at the first index so the running time is O(1). The worst case is where the key is at the last index of the array beacuse the algorithm must search through all the elements in the array making the time complexity O(N). The average case is where the key is in the middle so it searches thorugh N/2 elements making the time complexity O(N) again.

The best case for Binary Search is when the key value is in the middle because that's where the search begins therefore O(1) time complexity. The worst case for it is where the key is at the beginning or the end of the array forcing the algorithm to keep dividing into half making the time complexity O(logN). In the average case the value is neither at the middle nor the ends, it is somewhere inside the array and the time complexity is still O(logN).

The best case for Jump Search is also when the key value is in the first index with O(1) running time. The worst case is where the algorithm jumps √N times until the last block and then goes through all √N elements in the last block until finding the key in the last index making the time complexity O(√N). The average case is when the key is somewhere in the middle of the array with average combined jumping and linear searching sequences making the time complexity O(√N).

In each of the adapted algorithms we first find where the dead sheep used to be and then collect the K closest values therefore the collection part takes O(K) running time and since K is a function of N the time complexity of the collecting part is O(N). However if K = 1 then collecting also occurs in O(1).

When adapting all these algorithms into finding the K closest values in a sorted array we first check if the key is smaller than the first element or larger than the last element of the array. If that is the case which makes the best case the time complexity for searching is O(1) and after the collecting part mentioned above the total time complexity for the best case is O(1) for all 3 algorithms.

In other cases, instead of searching only for the key the adapted algorithms search for a pair of consecutive elements where one element is smaller than the key while the other one is larger which takes the same running time with the classic version in the average and the worst cases. This indicates where the dead sheep used to be so the algorithms start collecting the K closest values as mentioned above taking another O(N) time. In total:

Adapted Linear Search

Best Case: O(1)

Average Case: O(N + K) = O(N)

Worst Case: O(N + K) = O(N)

Adapted Binary Search

Best Case: O(1)

Average Case: O(logN + K) = O(N)

Worst Case: O(logN + K) = O(N)

Adapted Jump Search

Best Case: O(1)

Average Case: O(√N + K) = O(N)

Worst Case: O(√N + K) = O(N)

4.4

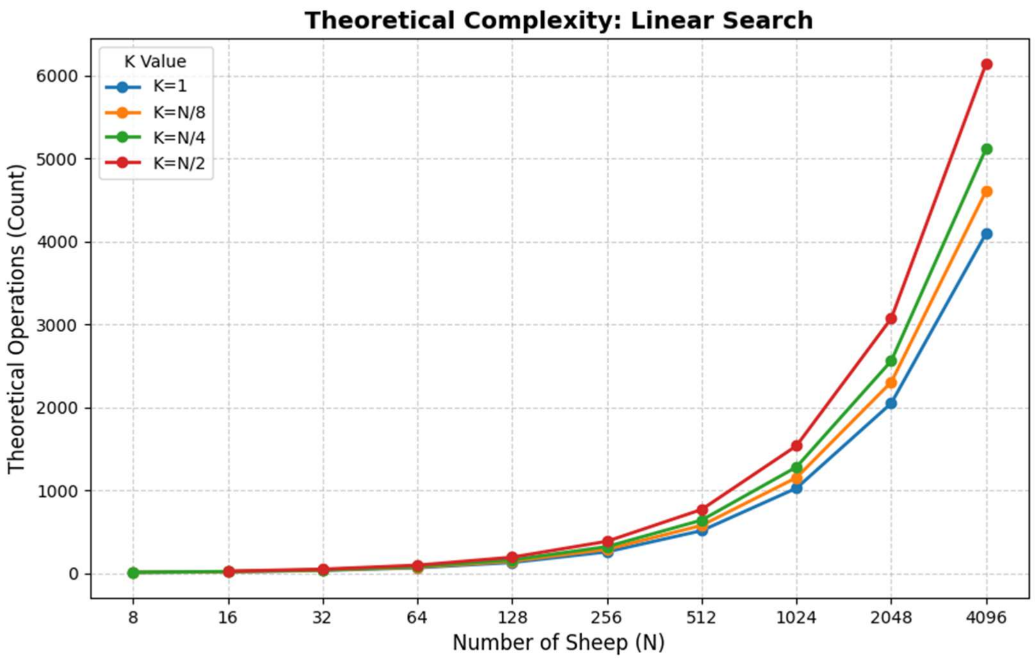
Since the key value is chosen randomly the best cases for each 3 algorithms are when K = 1 instead of being a function of N. In the observed results Linear Search still grows linearly and this agree with the time complexity O(N+K) when K = 1. For other values of K including the worst case K = N/2 the linear search algorithm still grows linearly aligning with the theoretical values O(N + K) = O(N).

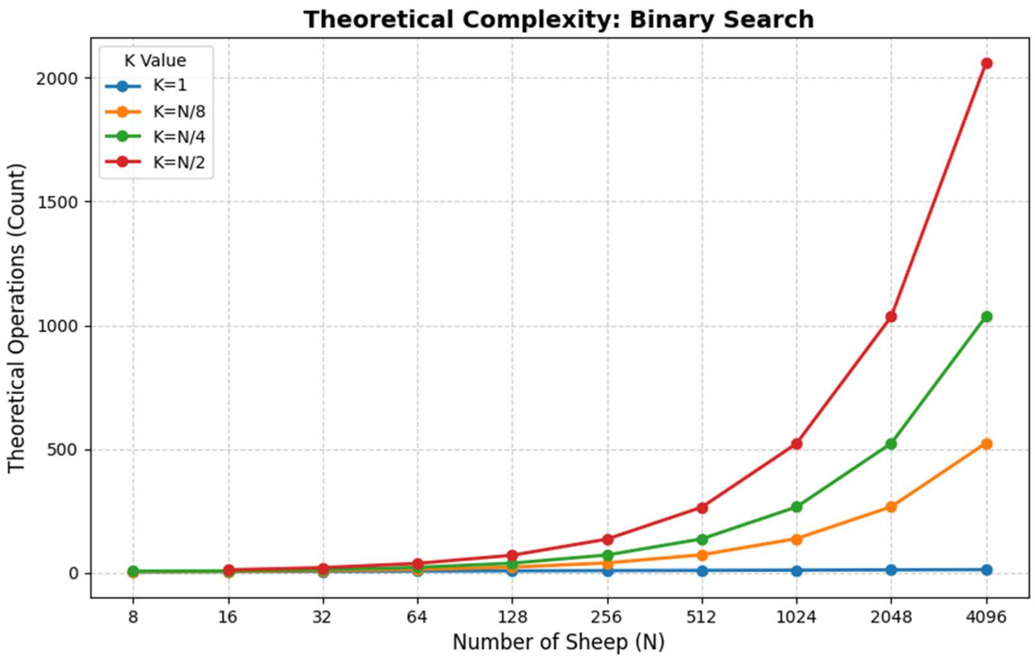
Binary Search algorithm is extremely fast and the graph is almost flat in the best case where K = 1 completely agreeing with the expected result O(logN + K) = O(logN) when K = 1. When K becomes a function of N, binary search starts growing linearly and the worst case occurs when K = N/2. This agrees with the theoretical prediction that when K is a function of N, time complexity O(logN + K) = O(N) so the running time must grow linearly.

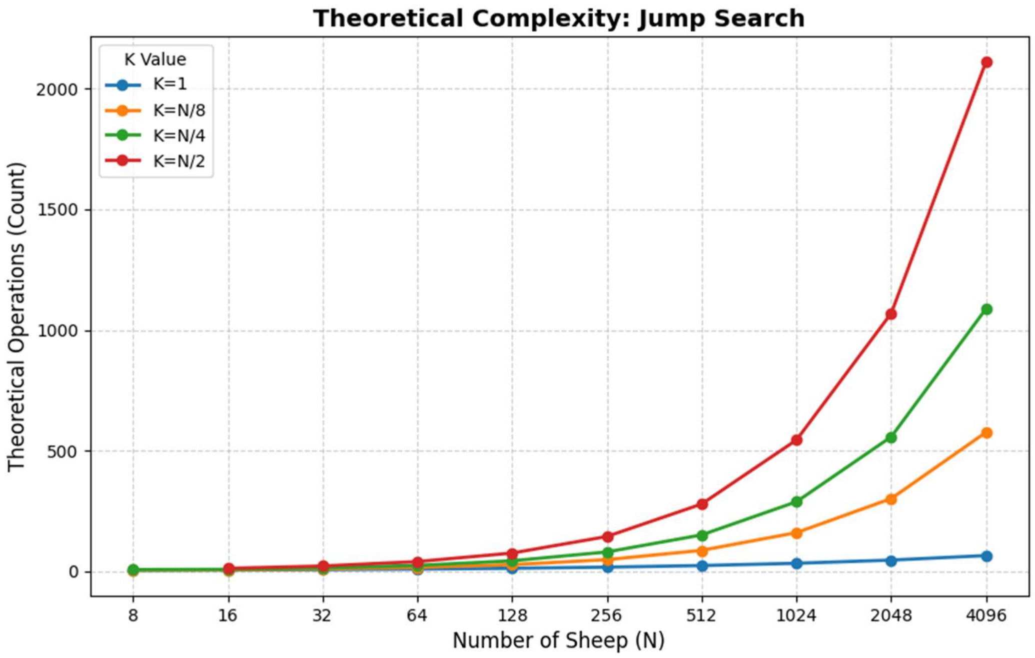
In the best case of the jump search when K = 1 the algorithm runs slower than the Binary Search but faster than the Linear Search, this agrees with its time complexity O(√N + K) = O(√N) when K = 1. In average cases and the worst case where K = N/2 the graph of Jump Search also grows linearly as expected in theoretical time complexity O(√N + K) = O(N) where K is a function of N.

Even though the observed data and the theoretical values mostly agree, there are a few spikes in the running time on the graphs. Especially, a noticeable spike occurs when N = 16, but it disappears at N = 32. This anomaly is most likely due to system noise or OS scheduling, such as background processes during measurement, making it an environmental issue.

4.5




Theoretical Complexity: Binary Search
K Value
K=1
K=N/8
K=N/4
K=N/2
Theoretical Operations (Count)
0
500
1000
1500
2000
8
16
32
64
128
256
512
1024
2048
4096
Number of Sheep (N)


Theoretical Complexity: Jump Search
K Value
K=1
K=N/8
K=N/4
K=N/2
Theoretical Operations (Count)
0
500
1000
1500
2000
8
16
32
64
128
256
512
1024
2048
4096
Number of Sheep (N)

Theoretical graphs and the observed performance analyses are consistent by growing linearly in all algorithms when K is a function of N and also consistent when $K = 1$ by growing linearly on linear search, logarithmically on binary search and proportional to $\sqrt{N}$ on jump search. However as mentioned above the only inconsistencies are the unexpected spikes, observed specifically at $N = 16$ in the generated graphs which may again be caused by the system noise or OS scheduling, such as background processes during measurement, making it an environmental issue.