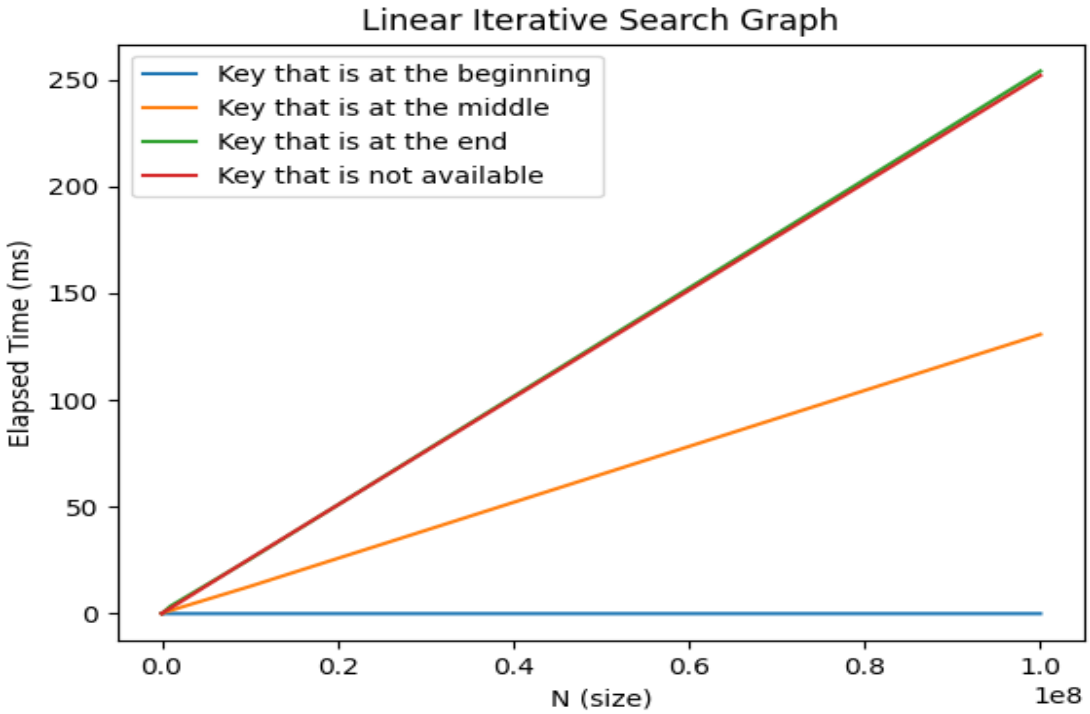
CS201
Section 01
Homework 2
Deniz Semih Özal
21802414

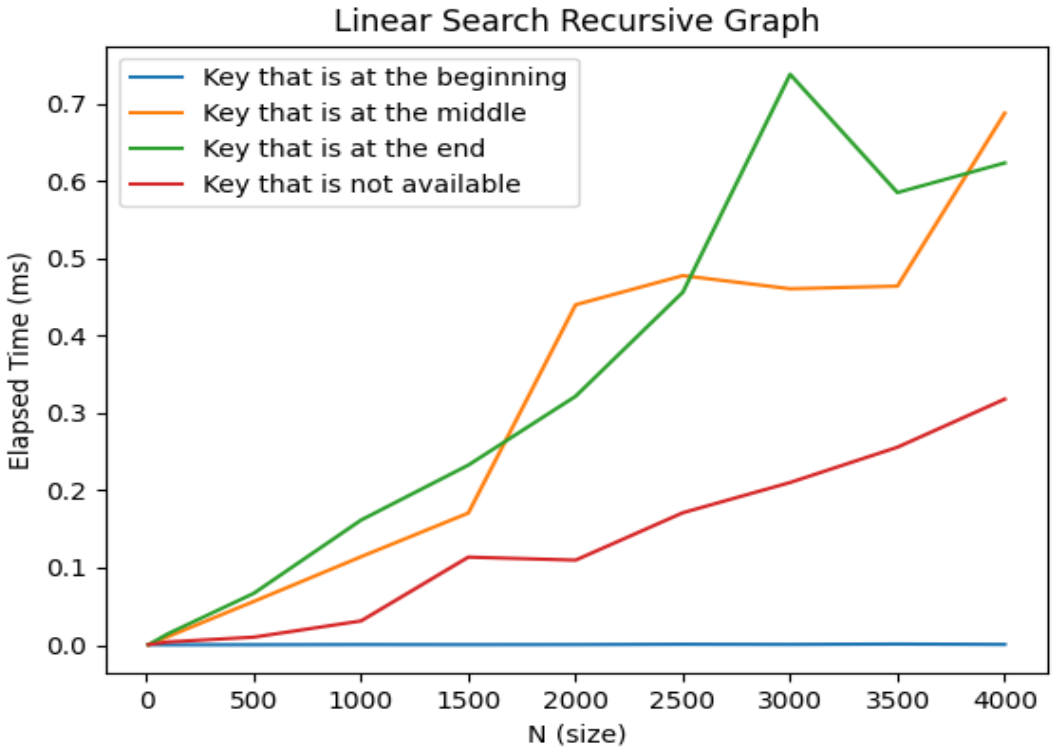
Table)

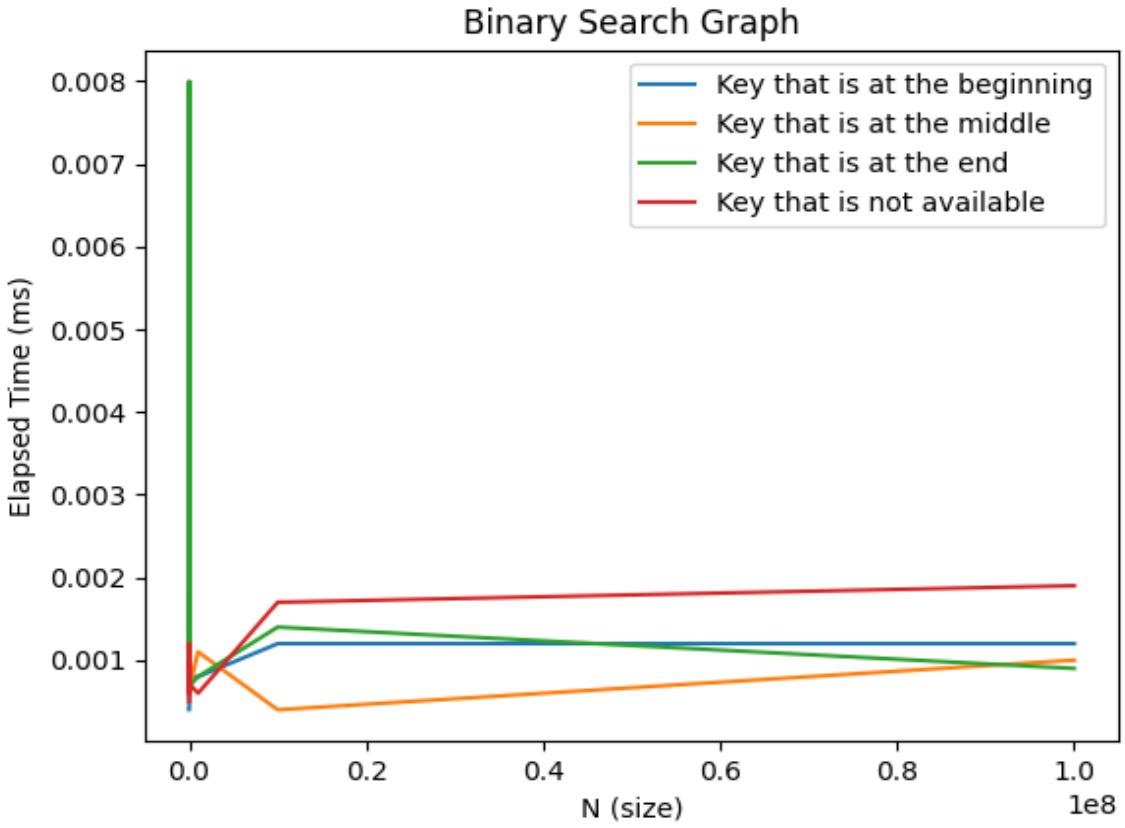
a: Key that is at the beginning
b: Key that is at the middle
c: Key that is at the end
d: Key that is not available

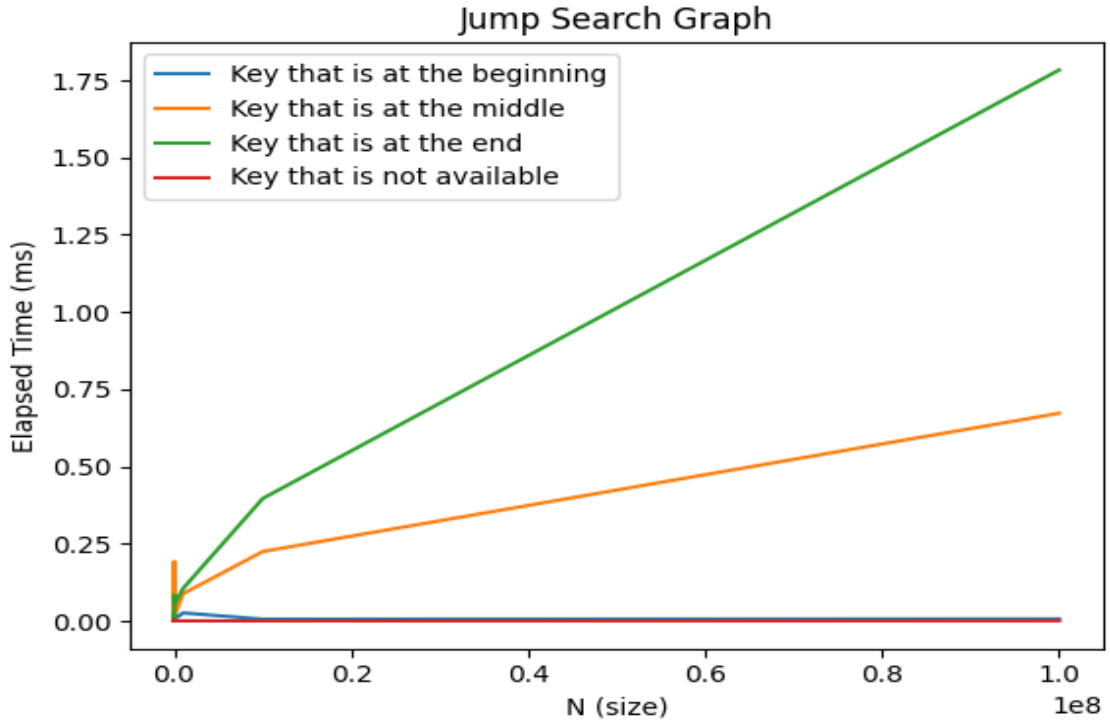
N	Linear (Iterative)				Linear (Recursive)				Binary Search				Jump Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
10	**0.0008**	**0.0003**	**0.0016**	**0.0017**	**0.0004**	**0.0004**	**0.0005**	**0.0006**	**0.0004**	**0.0006**	**0.0006**	**0.0005**	**0.015**	**0.0016**	**0.0007**	**0.0006**
100	**0.0004**	**0.0005**	**0.0006**	**0.0006**	**0.0003**	**0.0109**	**0.0146**	**0.004**	**0.0004**	**0.0007**	**0.0009**	**0.0006**	**0.0094**	**0.0008**	**0.0012**	**0.0007**
500	**0.0004**	**0.001**	**0.0016**	**0.0015**	**0.0004**	**0.0564**	**0.0669**	**0.0102**	**0.0009**	**0.0007**	**0.0009**	**0.0008**	**0.035**	**0.0026**	**0.0035**	**0.0039**
1000	**0.0075**	**0.0021**	**0.0033**	**0.0035**	**0.0006**	**0.1142**	**0.1616**	**0.0312**	**0.0008**	**0.0007**	**0.0008**	**0.0007**	**0.0132**	**0.0025**	**0.0063**	**0.0013**
1500	**0.0011**	**0.0025**	**0.0047**	**0.0044**	**0.0005**	**0.1707**	**0.2327**	**0.1137**	**0.0008**	**0.0008**	**0.0008**	**0.001**	**0.0095**	**0.0051**	**0.0208**	**0.0034**
2000	**0.0005**	**0.0031**	**0.0062**	**0.0062**	**0.0006**	**0.44**	**0.3217**	**0.1098**	**0.0009**	**0.001**	**0.008**	**0.0009**	**0.0136**	**0.193**	**0.0045**	**0.0013**
2500	**0.0008**	**0.0046**	**0.0082**	**0.008**	**0.001**	**0.4777**	**0.4563**	**0.1711**	**0.0011**	**0.0011**	**0.0011**	**0.001**	**0.0126**	**0.0045**	**0.0101**	**0.0015**
3000	**0.002**	**0.0047**	**0.0094**	**0.0078**	**0.0008**	**0.4607**	**0.738**	**0.2102**	**0.0009**	**0.0011**	**0.0007**	**0.0006**	**0.0078**	**0.0043**	**0.0219**	**0.0012**
3500	**0.0007**	**0.0048**	**0.0089**	**0.0325**	**0.0013**	**0.4641**	**0.5851**	**0.2558**	**0.0013**	**0.0009**	**0.0008**	**0.001**	**0.0091**	**0.0043**	**0.0104**	**0.0012**
4000	**0.0007**	**0.0054**	**0.0274**	**0.0106**	**0.0008**	**0.6877**	**0.6234**	**0.318**	**0.001**	**0.0017**	**0.0007**	**0.0007**	**0.0091**	**0.0054**	**0.0083**	**0.0011**
10000	**0.0006**	**0.013**	**0.0247**	**0.0234**	-	-	-	-	**0.0007**	**0.0018**	**0.0008**	**0.0012**	**0.0089**	**0.0092**	**0.084**	**0.0026**
100000	**0.0004**	**0.2272**	**0.2567**	**0.4113**	-	-	-	-	**0.0008**	**0.0007**	**0.0007**	**0.0007**	**0.0079**	**0.0225**	**0.0577**	**0.0007**
1000000	**0.0003**	**1.4718**	**3.5187**	**2.6005**	-	-	-	-	**0.0008**	**0.0011**	**0.0008**	**0.0006**	**0.0264**	**0.0883**	**0.1054**	**0.0006**
10000000	**0.0004**	**12.7114**	**25.4189**	**25.6346**	-	-	-	-	**0.0012**	**0.0004**	**0.0014**	**0.0017**	**0.0061**	**0.225**	**0.3966**	**0.0007**
100000000	**0.0004**	**130.719**	**254.009**	**251.97**	-	-	-	-	**0.0012**	**0.001**	**0.0009**	**0.0019**	**0.0068**	**0.6726**	**1.7834**	**0.001**

Experimental Graphs)






Binary Search Graph
Key that is at the beginning
Key that is at the middle
Key that is at the end
Key that is not available
Elapsed Time (ms)
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.0
0.2
0.4
0.6
0.8
1.0
1e8
N (size)


Jump Search Graph
Key that is at the beginning
Key that is at the middle
Key that is at the end
Key that is not available
Elapsed Time (ms)
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
1e8
N (size)

5.1)

Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.81 GHz
Ram: 16.0 GB
Operating System: Windows 10 Pro
Version: 2004
System Type: 64-bit operating system, x64-based processor

5.2)

a) Linear Search Iterative Algorithm
- Theoretical Worst-Case Performance: O(N)
- Theoretical Average-Case performance: O(N)
- Theoretical Best-Case Performance: O(1)
- Theoretical Unsuccessful Search Performance: O(N)

b) Linear Search Recursive
- Theoretical Worst-Case Performance: O(N)
- Theoretical Average-Case performance: O(N)
- Theoretical Best-Case Performance: O(1)
- Theoretical Unsuccessful Search Performance: O(N)

c) Binary Search
- Theoretical Worst-Case Performance: O(log(N))
- Theoretical Average-Case performance: O(log(N))
- Theoretical Best-Case Performance: O(1)
- Theoretical Unsuccessful Search Performance: O(log(N))

d) Jump Search
- Theoretical Worst-Case Performance: O(√N)
- Theoretical Average-Case performance: O(√N)
- Theoretical Best-Case Performance: O(1)
- Theoretical Unsuccessful Search Performance: O(√N)

5.3) According to the graphs, my experimental results are similar to the theoretical ones.

a) Linear Search Iterative Algorithm

- The expected worst case is O(N)and it happens when the item is located at the last index of an array. Again, when key is not available in the array the result is quite similar to the previous one, as I am expected. The best case happens when an item is located at the first index of an array. As it can be seen from the graph, the line is almost constant, that is O(1).

b) Linear Search Recursive Algorithm

- The expected worst case is O(N) and it happens when the item is located at the last index of an array. This graph is quite a bit different compared to iterative one because size can be at most 4300, and it gives a stack overflow error when I

exceed the limit. The reason for that since stack has a limited memory, the trees of a recursion exceeds the limit. And another interesting result is that the unsuccessful search takes less time compared to key at the end. The reason might be properties of my computer or may be visual studio terminal. But again the worst case is O(N) and best case is O(1).

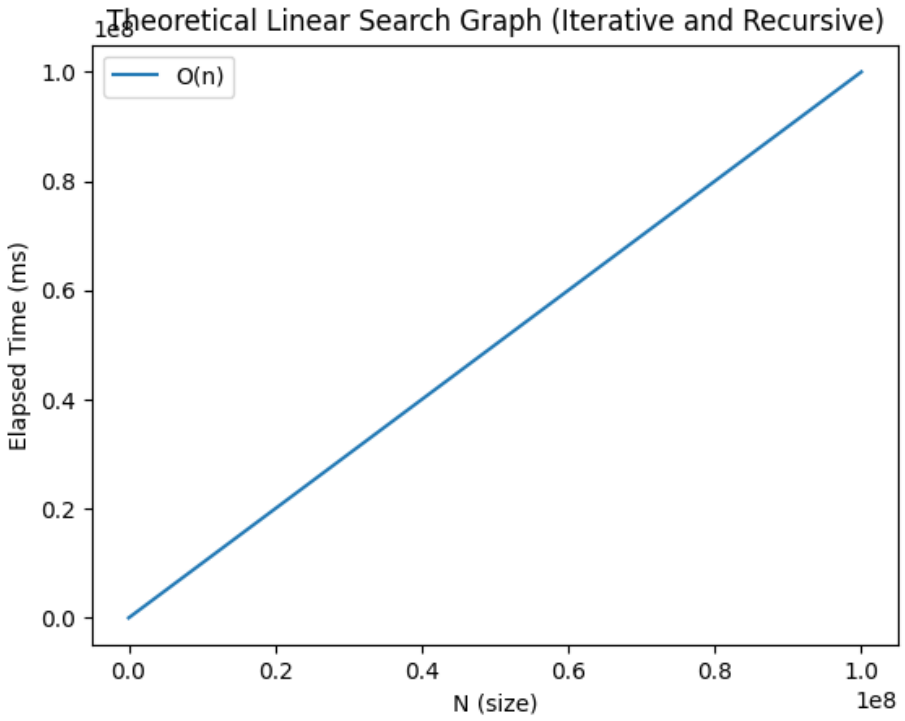
c) Binary Search Algorithm

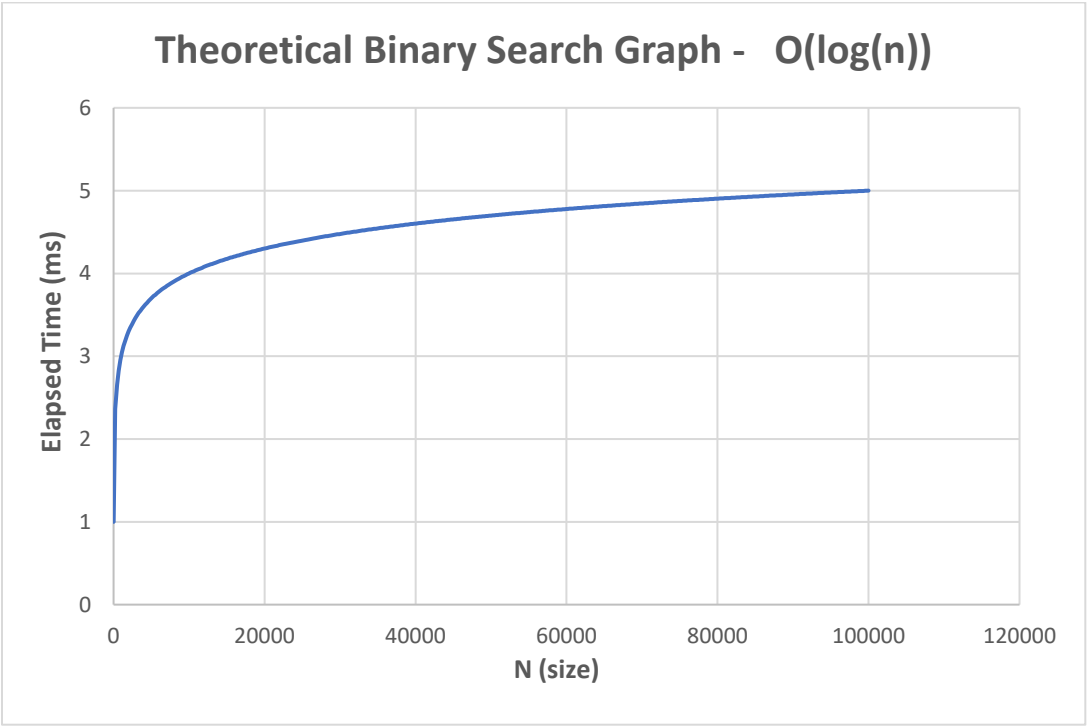
- The expected worst case is log(N) and it happens when the key is not available. My experimental graph resembles log(N) graph except for the rocket-ish part, and I don't know the reason. Overall, it resembles log(N) graph, and the worst case is log(N), when the key is not available as it can be seen the graph. The best case is when the key is at the middle and it is very logical because since binary search algorithm halves a portion in every iteration, the middle part corresponds to the first halved part. Therefore, O(1) is the best case for binary search algorithm.

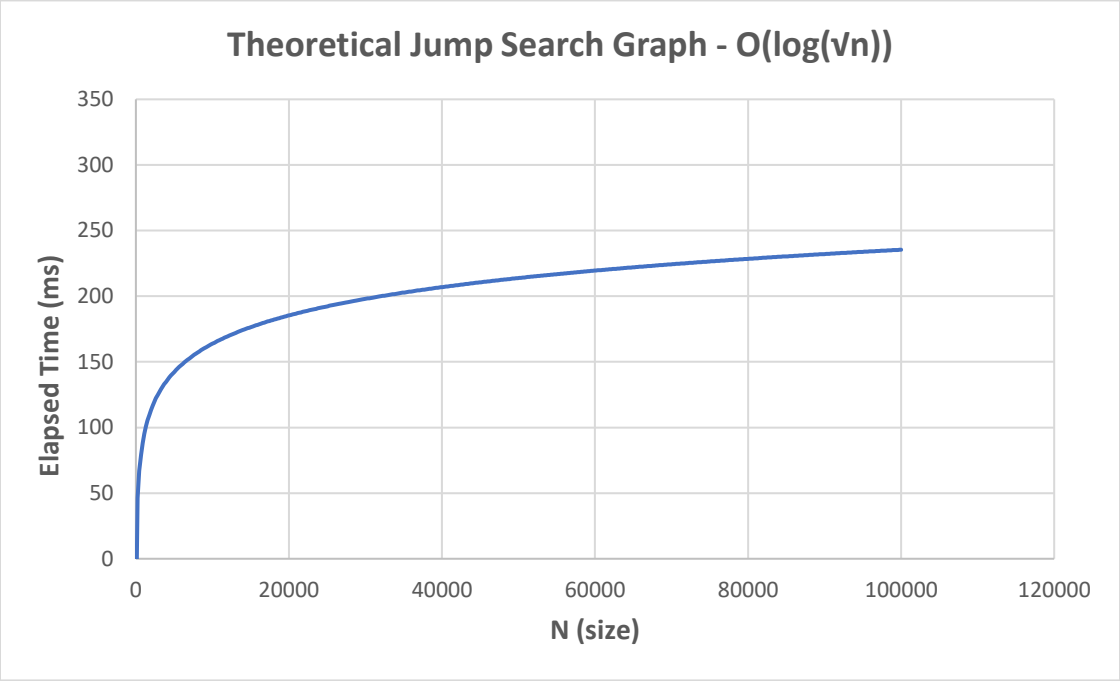
d) Jump Search Algorithm

- The expected worst case is O(√N) and it happens when the item is located at the last index of an array. Since jump search algorithm starts from head and goes to the tail gradually, it is very reasonable to the worst case. The graph resembles O(√N) graph if the lines were more inclined. The interesting part of this algorithm since I have used for the wrong key -1, it is similar to the best case, which is O (1). However, if I use an item that bigger than the last index of an array, again the expected unsuccessful search line would be similar to the end key line. The reason for that since jump search algorithm compares the first index of an array and the value, and if the value smaller than the first index it returns -1, so it becomes the best case.

5.4) Theoretical Graphs







- Linear search iterative and recursive have same time complexities, which are O(N)
- Binary search iterative has time complexity O(log(N)
- Jump search graph has time complexity O(log(√n))

Comment: Linear search iterative looks similar to the theoretical one but linear search recursive a bit different from the theoretical one. Time delays, computer properties and difference between code editors may happen this problem. Binary search algorithm resembles theoretical one except the rocket part for key at the end. Jump search resembles the theoretical one but it needs to be more inclined but since I have used to plot graphs Python, I could not find a way to plot more corrugated plot.