# BLG223E Homework-5

Ertuğrul Şahin
150210028

In this assignment, I analyzed the operation of the ADD, UPDATE, and DELETE functions, respectively, based on the codes I wrote in the first 4 assignments. I created datasets containing 1k, 5k, 10k, 20k, 50k, 100k, 250k, 500k, 750k and 1m employees. I analyzed the execution speeds of the ADD, UPDATE, and DELETE functions of the codes in each group, one by one, in a graph. Using the datasets I created while performing the analysis, I examined the change in the running speeds of the functions depending on the number of employees.

**<u>These are my codes:</u>**

- File I/O (Homework 1)
- Array (Homework 1)
- Linked List (Homework 2)
- STL Vector (Homework 2)
- STL List (Homework 2)
- Binary Search Tree (Homework 3)
- STL Map (Homework 3)
- Skip List (Homework 4)

**<u>These are the operations:</u>**

- **insert** : Create a node randomly and insert it into the structure
- **search** : Select a search key randomly and find it in the structure. The key that you are looking for might not be in the structure.
- **remove** : Select a random key, find it in the structure and delete the node. Again, the key that you are looking for might not be in the structure.

**<u>These are the groups:</u>**

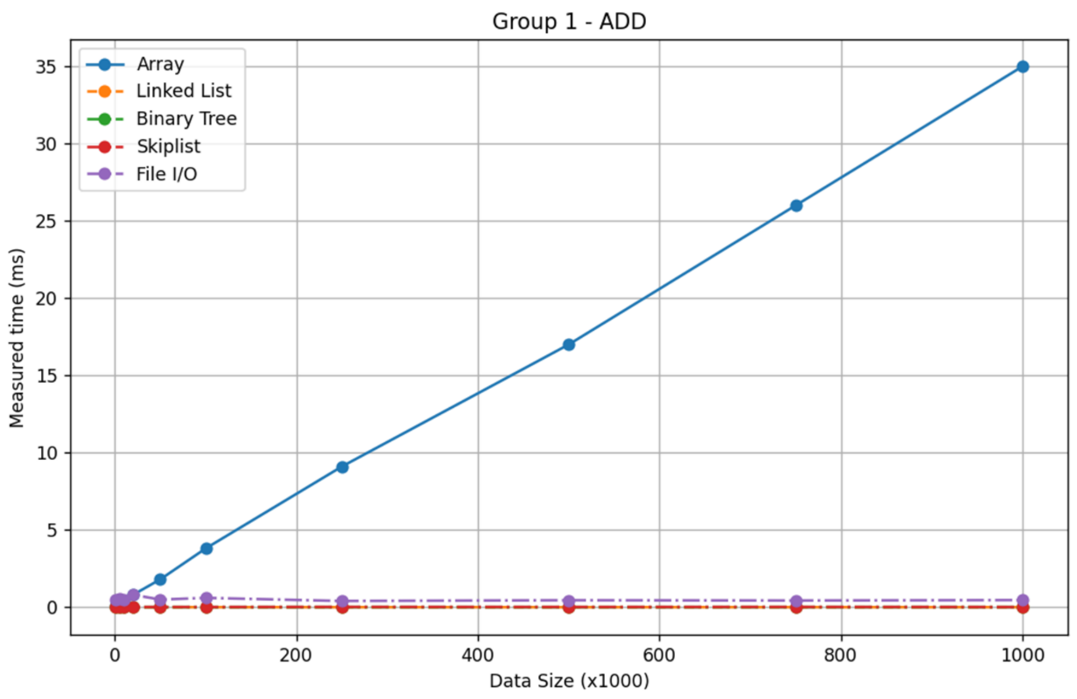
- <u>Group-1:</u> File I/O, Array, Linked List, Binary Search Tree, Skip List
- <u>Group-2:</u> Array, Linked List, STL Vector, STL List
- <u>Group-3:</u> Binary Search Tree, STL Map, Skip List

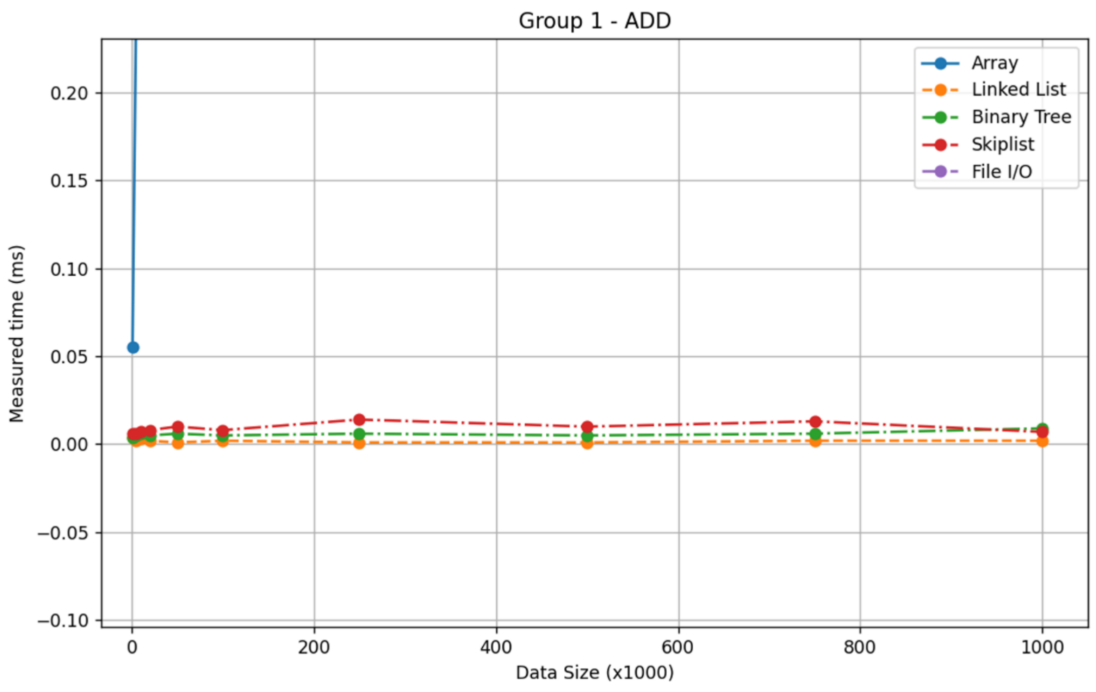
## Graphs

Note: Since the results of some data structures seem to conflict, I enlarged the graph and added a second photo.

## Group-1:

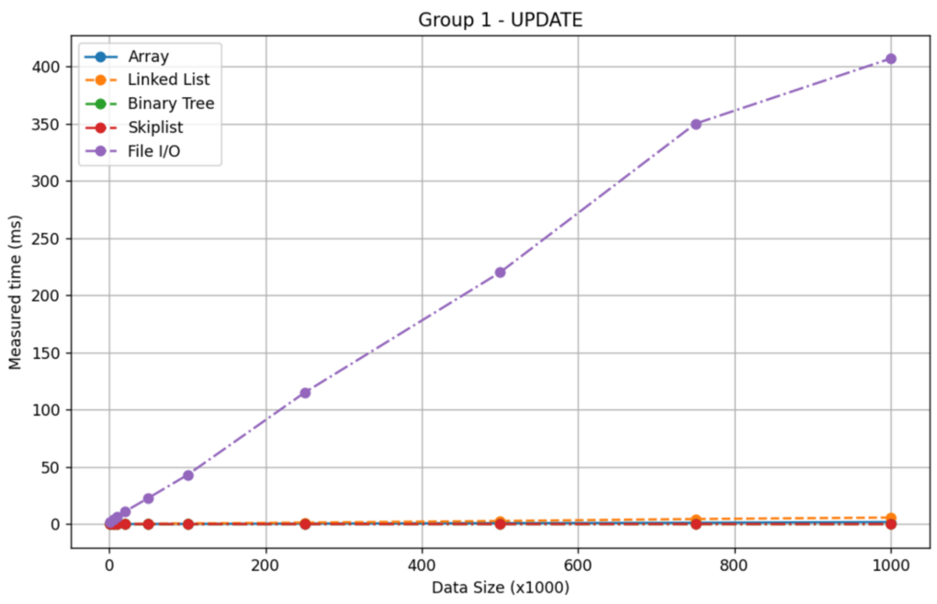
### ADD:

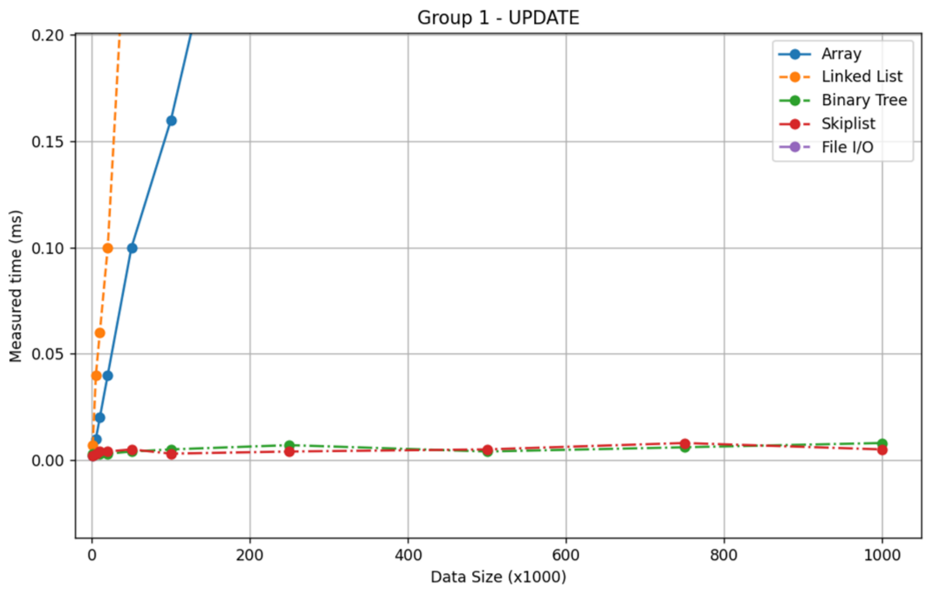




As can be seen from the graph, the ADD function of the Array data structure works in O(n) complexity. Other data structures work with similar complexity and work much more efficiently than the array data structure.

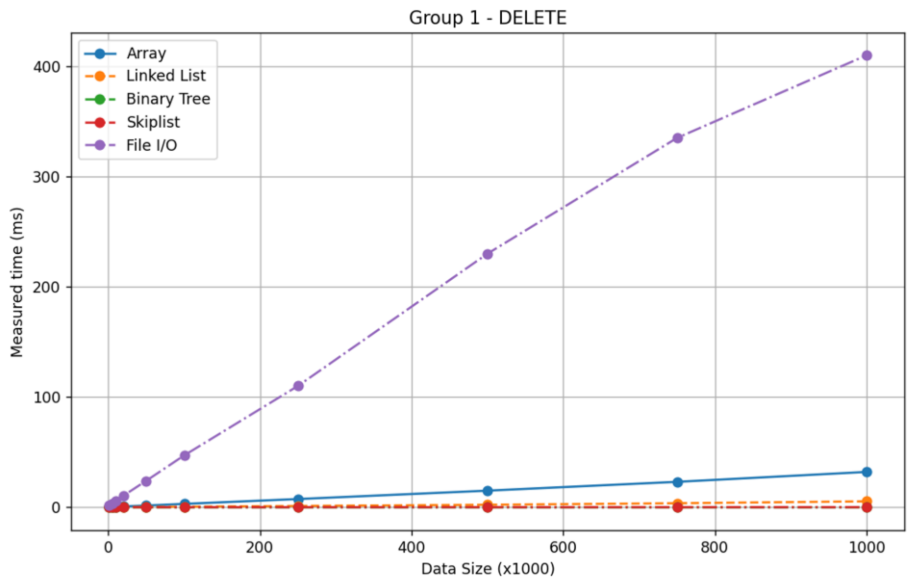
## UPDATE:

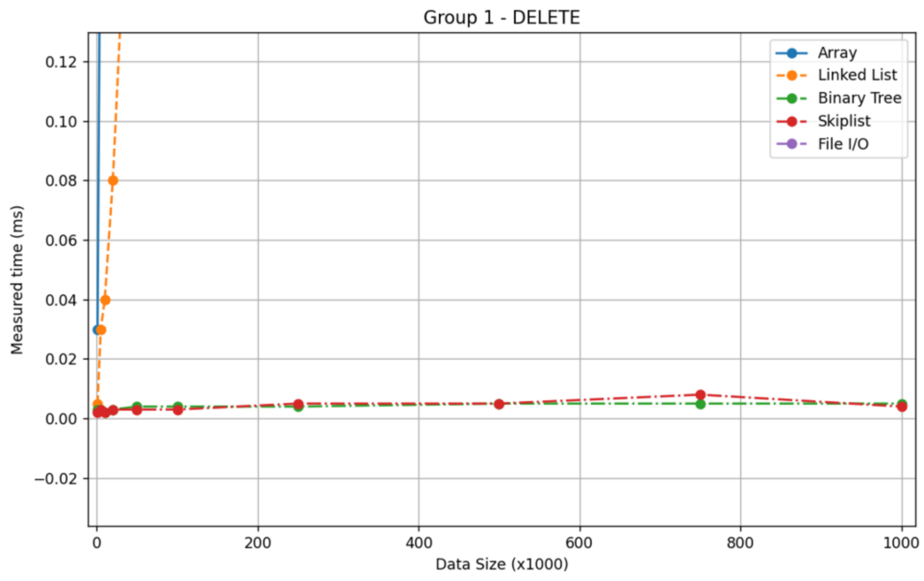




As can be seen from the first photo, File I/O operates at O(n) complexity. When we zoom in on the graph as in the 2nd photo, it can be seen that Linked List and Array data structures operate at O(n) complexity. Skip List and Binary Search Tree data structures have almost no change in their operating speed despite the size of the datasets changing, but in real world, they work in O(logn) complexity.

## DELETE:

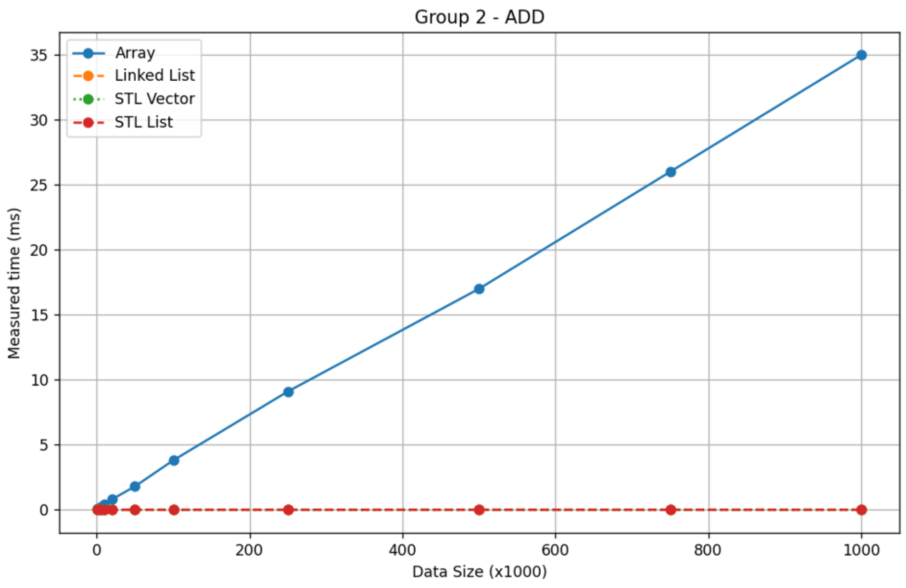


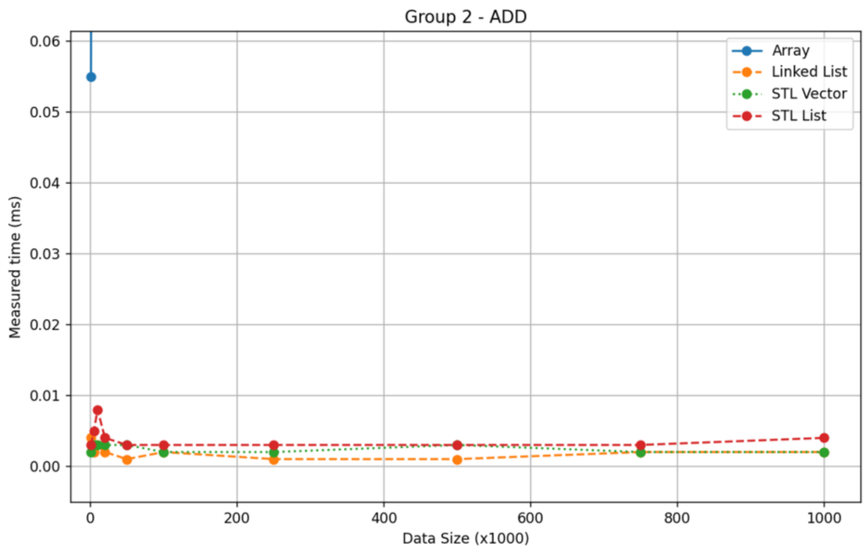


As can be seen from the first photo, File I/O operates at O(n) complexity. When we zoom in on the graph as in the 2nd photo, it can be seen that Linked List and Array data structures operate at O(n) complexity. Skip List and Binary Search Tree data structures have almost no change in their operating speed despite the size of the datasets changing, but in real world, they work in O(logn) complexity.

## Group-2:

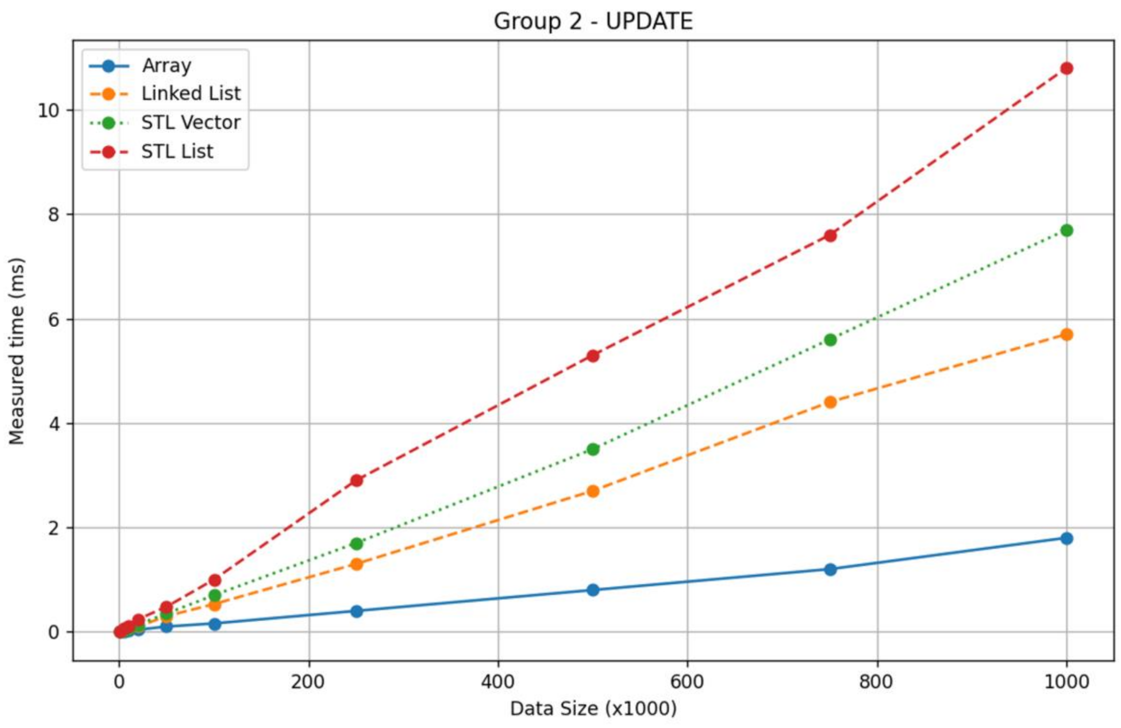
### ADD:





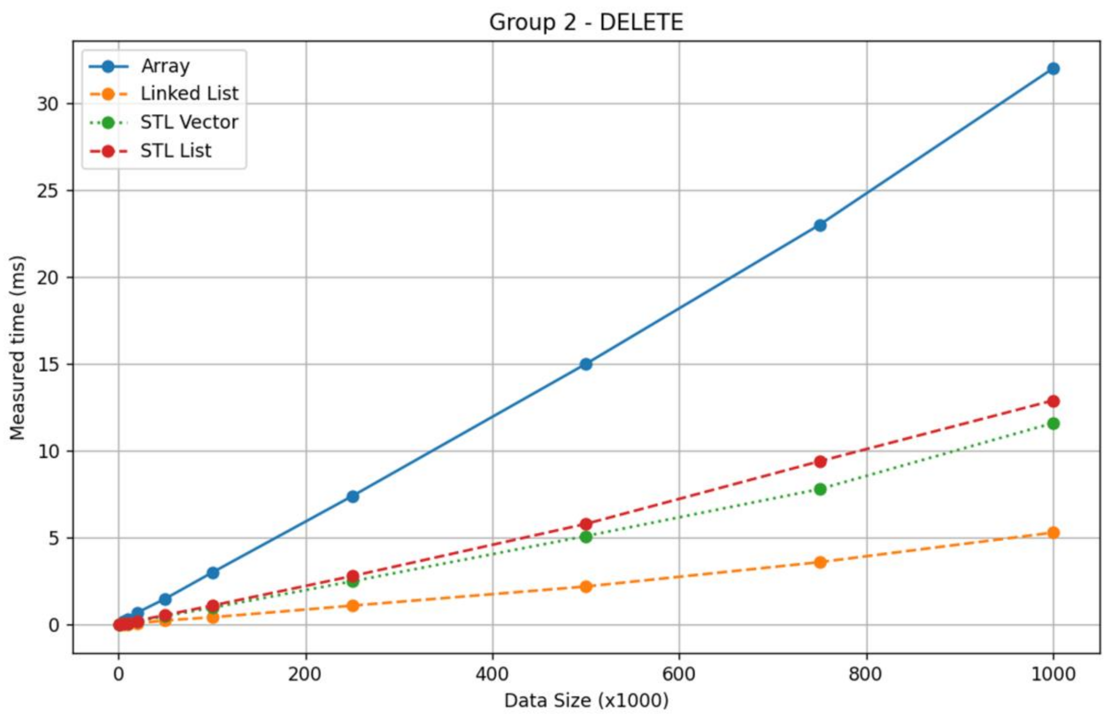
As can be seen from the first photo, the Array data structure operates at O(n) complexity. When we enlarge the graph as in the 2nd photo, there is almost no change in the operating speed of the other 3 data structures in the group, although the datasets have changed.

UPDATE:



As can be clearly seen from this graph, all data structures operate at O(n) complexity. As the size of the dataset increases, that is, as the number of employees increases, the time it takes for each of them to update the specified ID changes in direct proportion.

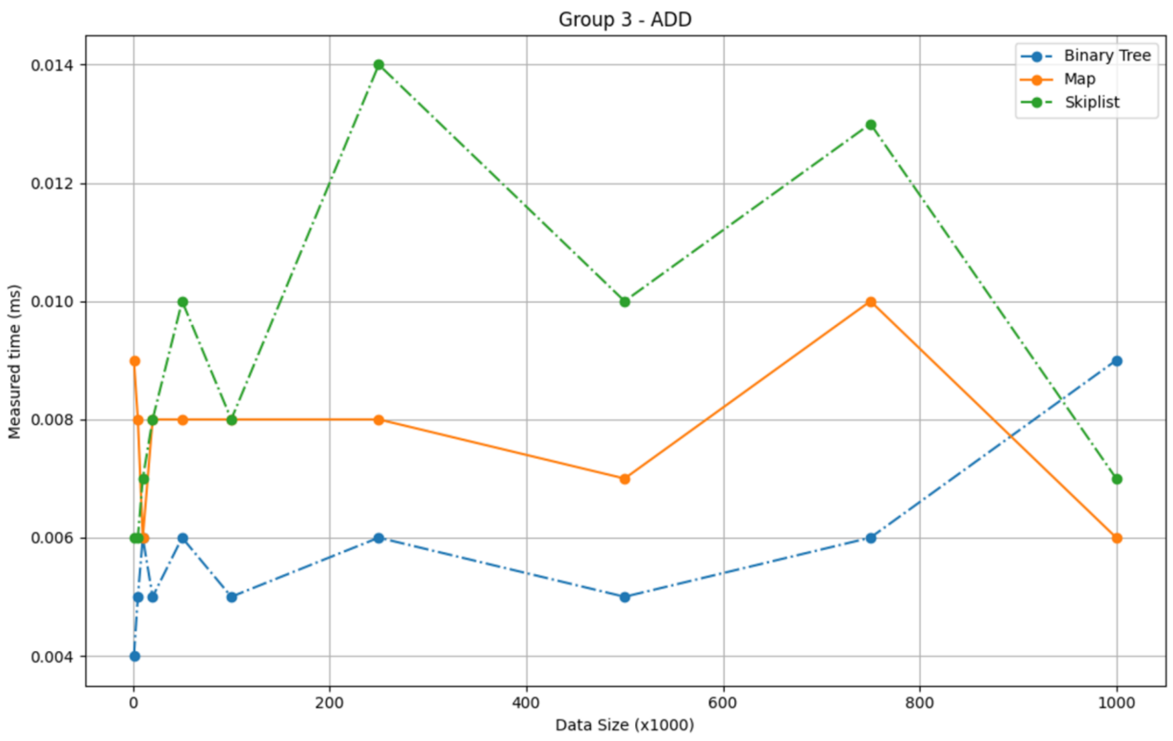
DELETE:



As can be clearly seen from this graph, all data structures operate at O(n) complexity. As the size of the dataset increases, that is, as the number of employees increases, the time it takes for each of them to delete the specified ID changes in direct proportion.

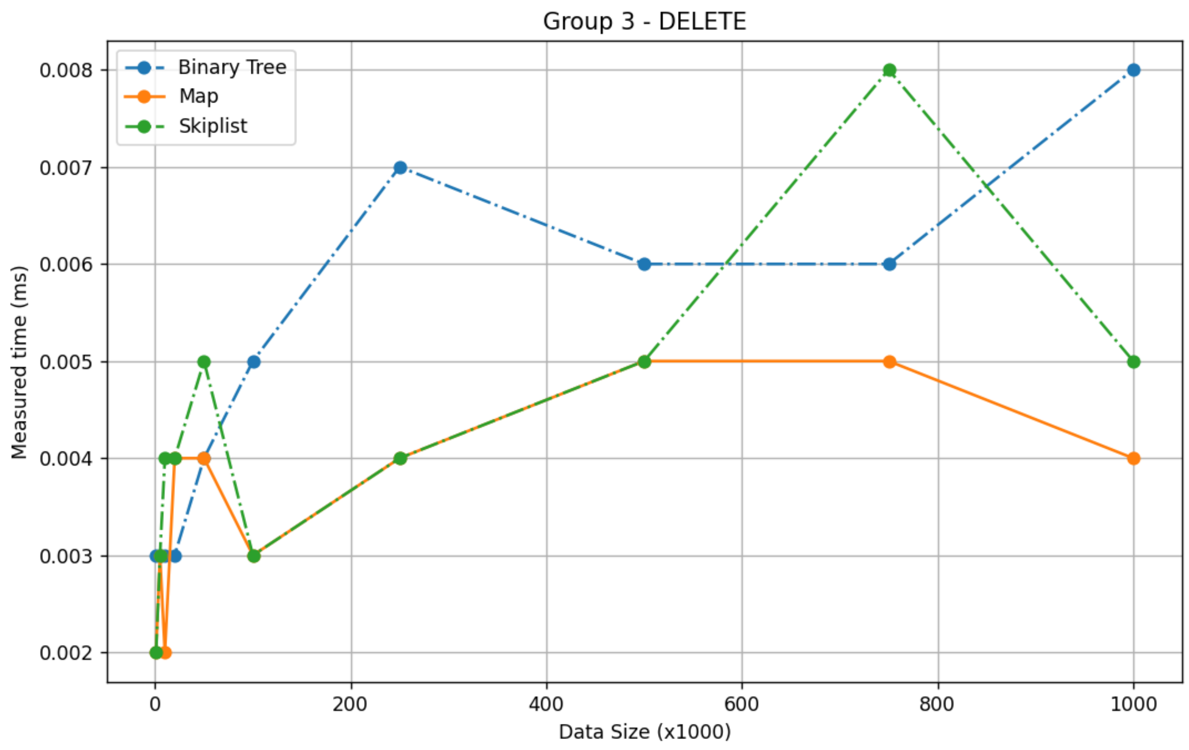
## Group-3:

### ADD:



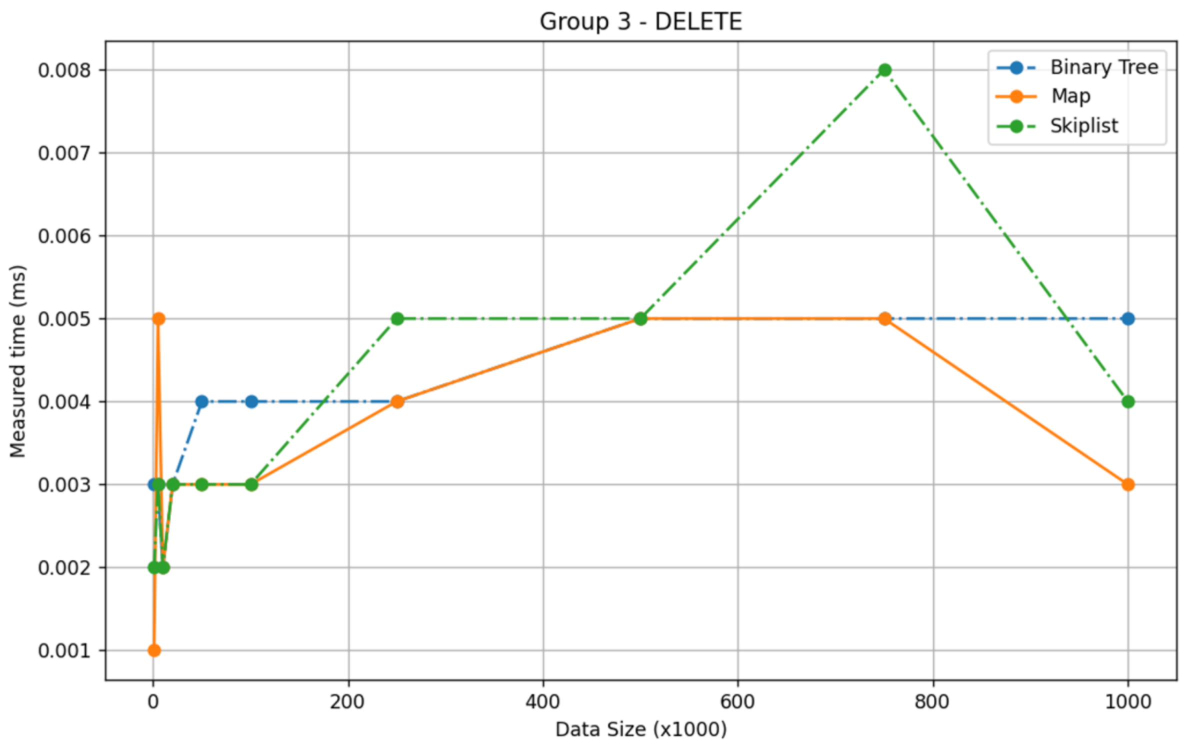
Since the level of each data is determined randomly in the working principle of the Skip List data structure, we cannot draw a clear conclusion. The running time of each data structure has changed very little even as the size of the dataset changes. Since the periods covered by the measured time section in the graph are very small numbers, the results in the graph seem not precise, but if we increase the coverage area, we can observe that these results almost do not change.

## UPDATE:



As I explained in the ADD function, the same situations apply in this graph. The amount of change in each data structure is very, very low.

DELETE:



As I explained in the ADD function, the same situations apply in this graph. The amount of change in each data structure is very, very low.