Faculty of Informatics

and

Digital Technologies,

Rijeka

**Benchmark report of Unordered array vs ordered linked list vs hash table vs balanced tree for inserting, finding, and deleting elements**

**Course: Infrastructure for large volume dana**

**dr. sc. Rok Piltaver**

**Ivan Fajta**

**Rijeka, 25.12.2023.**

Content

[1. Tested data structures, 3](#_Toc154424609)

[2. Load simulation 3](#_Toc154424610)

[3. Response time measurement 3](#_Toc154424611)

[4. Rezultati usporedbe performansi 4](#_Toc154424612)

[5. Measurement of Response Times 4](#_Toc154424613)

[6. Limitations 4](#_Toc154424614)

[7. Conclusion 5](#_Toc154424615)

[8. References 5](#_Toc154424616)

[9. Inclosure 5](#_Toc154424617)

## Tested data structures,

Four distinct data structure implementations—the Unordered Array, Linked List, Hash Table, and Balanced Tree—were tested in this benchmark. Depending on the requirements of the system, each one offers a distinct set of benefits and can be used in various situations.

Unordered Array: The data in this data structure is stored as an unordered array, or list. Although it reads quickly, insert and delete operations get slower as the size of the array grows. Examples of applications include straightforward systems with minimal data flow.

Linked List: How a linked list is implemented. Although data access may be slower with this format, insertion and deletion can happen more quickly. When there are frequent data insertion and deletion processes, it is employed.

Hash Table: The data in this data structure is stored in a dictionary, or hash table. Although it reads, inserts, and deletes data quickly, if not used properly, it may result in collisions. It is frequently employed when a fast data search is necessary.

The implementation of a sorted dictionary is known as a balanced tree. It offers a fair mix of speed for data deletion, insertion, and search. Applications where all three processes are necessary and the data is dynamic frequently employ it.

In one of the systems, we also applied the Unittest algorithm. The purpose of the unittest technique benchmarked in this report is to assess the performance of various data structures, including balanced trees, linked lists, hash tables, and unordered arrays. Python's unittest framework is used to implement and test these data structures. Every data structure offers a unique method of handling and modifying data, with varied benefits and drawbacks. Evaluating their effectiveness in insertion, retrieval, and deletion processes is the aim.Arhitektura sustava

The SQLite database is used by the system architecture to store data. The SQLite3 module was used to establish a connection with the database. Transactions are used by the system to guarantee data consistency.

## Load simulation

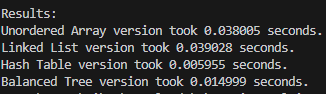
The generated data was used to mimic the system load. To mimic actual workloads in dynamic systems, insert, search, and delete operations are carried out at random while keys are generated dynamically.

## Response time measurement

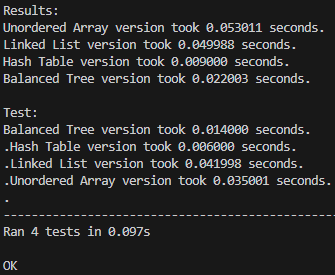
The Python time module was used to calculate each implementation's response time. To mimic the real-world application of data structures, transactions are encircled by operations. The results display average response times.

## Rezultati usporedbe performansi

The following graph displays the results of the performance measurement without unittest:



The following graph displays the results of the performance measurement with unittest:



After examining the data, we find that the Hash Table can insert, search for, and remove data the quickest. Balanced Tree executes somewhat quicker than Unordered Array and Linked List but slower than Hash Table. Unordered Array and Linked List have higher execution durations. These findings suggest that, depending on the kind of operations the system must carry out, the selection of data structure is essential.

## Measurement of Response Times

Response times are measured by recording the elapsed time for each data structure operation (insertion, retrieval, deletion) using the time module in Python. The results are aggregated to provide insights into the relative performance of each data structure.

## Limitations

- One computer with certain resources was used to perform the measurement.

## Conclusion

The goal of the benchmark report is to present a thorough examination of the unit-tested data structures, revealing details about their performance attributes. The outcomes of simulations and measurements provide useful information for choosing the best data structure according to particular application needs.

- Applications requiring a lot of search and delete operations are advised to use balanced trees and hash tables.

Applications requiring frequent insertion operations are better served by linked lists and unordered arrays.

## References

- Python documentation.

- SortedContainers documentation.

## Inclosure

- The source code of the implementations