# Hybrid Data Structures

# Multi-Level Skip List

# DSA Project

Sorting Spartans



|  |  |  |
| --- | --- | --- |
| S.No | Name | Roll No |
| 1. | Hari Kesav.D | CB.EN.U4CSE21421 |
| 2. | M.K.Eniyan | CB.EN.U4CSE21435 |
| 3. | Mitesh | CB.EN.U4CSE21438 |
| 4. | Raj Kumar | CB.EN.U4CSE21446 |
| 5. | S.Harecharan | CB.EN.U4CSE21450 |

# Introduction:

What is Hybrid Data Structure?

A Hybrid Data Structure is a combination of one or more Data Structures characteristics which is meant for betterment implementation of storing the data in an efficient manner. Which is often designed to leverage the strengths of each component structure to address specific problem requirements efficiently. This brings together different data structures to provide an optimal balance of performance, memory usage and functionality for a given application or problem domain.

The motivation behind using a hybrid data structure is to overcome the limitation of individual data structures and provide a more tailored solution. By combining multiple data structures, a hybrid structure can inherit the advantage of each component while mitigating their weakness. This can result in improved efficiency, reduced memory consumption, and enhanced functionality.

The choice of which data structures to combine in a hybrid structure depends on the problem domain and the specific requirements of the application. The advantage of hybrid data structure lies in their ability to tackle complex problems by using the right tool for the job. They provide a customized solution that is efficient, space-efficient and optimized for the specific requirements of the problem at hand.

Designing and implementing a hybrid data structure requires careful consideration of the problem domain, the characteristics of the data, and the desired trade-offs. It involves identifying the strength and weakness of individual data structures and integrating them in a way that maximizes performance and functionality. This Hybrid Data Structurers is introduced in places

The insight of the project is to understand and implement a hybrid data structure and get exposed to the advantages and disadvantages of solving a complex problem efficiently rather than using a traditional data structure. This Hybrid Data Structure is designed and implemented by creating a Skip List and incorporating into a Tree data structure.

There are many practical applications

1. Database Systems: To efficiently index and query big datasets, database systems frequently use multi-level skip lists. They offer a useful data structure for quickly indexing key-value pairs and performing searches. When a range of keys must be obtained from the database, the speed of range queries can be enhanced using multi-level skip lists.

2.Indexing of Spatial Data: Multi-level skip lists are appropriate for indexing of spatial data. They are effective for storing and retrieving spatial objects like polygons, rectangles, and points. Spatial queries like range searches and nearest neighbour searches can be carried out more quickly by taking advantage of the hierarchical structure of multi-level skip lists.

3.Geographic Information System: Applications for geographic information systems (GIS) require effective indexing and retrieval of spatial data. Multi-level skip lists can be used to store and organise spatial data, enabling operations like proximity searches and spatial overlays as well as quick access to geographic information.

4.Web Search Engines: Web search engines can utilise multi-level skip lists to speed up the retrieval of search results. Search engines can more quickly find pertinent materials based on search queries by indexing web pages or documents using a multi-level skip list, which enhances search speed.

5. File systems: To improve the effectiveness of file retrieval and searching activities, multi-level skip lists can be used. A multi-level skip list is used to organise files in a hierarchical structure so that file systems may easily find and retrieve files based on their names, qualities, or other factors.

Time Complexity: For insertion and deletion of an element in a skip list is of the time complexity O(log n) and for a binary search tree the worst case is O(n) where the hybrid data structure is of the average time complexity of O(n). For searching the elements in the skip list is O(log n) and binary search tree with worst case as O(n). So overall the time complexity

Space Complexity: The space complexity of the hybris data structure combining a skip list and a binary search tree is typically O(n+h), where n is number of elements stored and h is height of the skip list which includes space required for forward pointers in skip list and nodes in binary search tree.

***Overview of the Hybrid Data Structure:***

**Skip List – Overview**

Finding a given entry in a traditional singly-linked list of sorted integers requires scanning the entire list, which results in a linear search with an O(n) time complexity. Contrary to arrays, we cannot rapidly enter the middle of the list to conduct a binary search.

An alternative is to use a skip list to get around this restriction. A skip list is a data structure made up of several linked lists, each of which has roughly half the nodes of the one before it. Faster searching is possible thanks to its hierarchical structure.

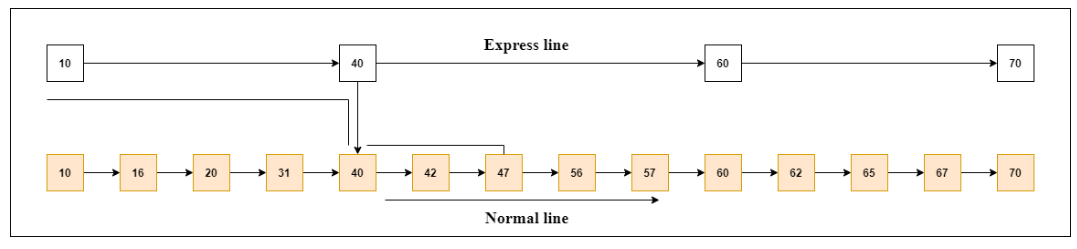
WORKING OF SKIP LIST

Let's take an example to understand the working of the skip list. In this example, we have 14 nodes, such that these nodes are divided into two layers, as shown in the diagram.

The lower layer is a common line that links all nodes, and the top layer is an express line that links only the main nodes, as you can see in the diagram.

Suppose you want to find 47 in this example. You will start the search from the first node of the express line and continue running on the express line until you find a node that is equal a 47 or more than 47.

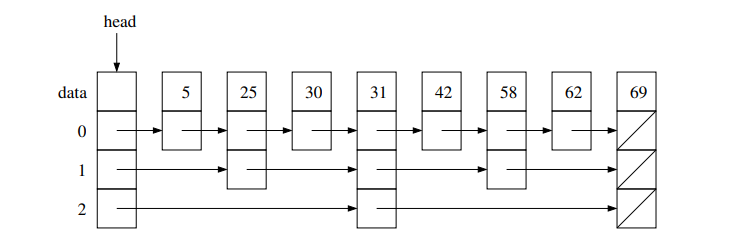
You can see in the example that 47 does not exist in the express line, so you search for a node of less than 47, which is 40. Now, you go to the normal line with the help of 40, and search the 47, as shown in the diagram.



**WORKING OF Multilevel SKIP LIST:**

When looking for an element in a skip list, such as "42" we start at the highest level and compare the element with each node there.

Upon finding the element to be after moving on to the following node in the same level after a node. '42' follows '31' in the example, therefore we move on to the following node, '58', in the subsequent lower level. Up until the lowest level, where a linear search is conducted within a constrained range, the process continues. In this instance, we find '42' and verify that it exists in the structure by returnigtrue. In comparison to a standard singly linked list, a skip list improves search efficiency by including more pointers and employing more layers. It combines the benefits of linked lists and binary search trees, enabling efficient searches even when the list's middle is not directly accessible.



Skip list FAQ:

Que: How are elements inserted and erased in a skip list?

Ans: In a skip list, once the desired location for insertion or deletion is found, the changes are made by modifying the chains at each level. To insert an element, a new node is created and inserted into the appropriate positions within each level. For erasing an element, the corresponding nodes are unlinked and removed from each level.

Que: How are the nodes placed at each level in a skip list?

Ans: When inserting a new element into a skip list, the level at which the nodes are placed is determined randomly. This randomization is typically done by generating a random number for the top level of the new element from the range [0, log n], where n represents the current number of elements in the skip list. This process ensures that the distribution of elements across levels is balanced and helps maintain the overall structure of the skip list.

Que: What is the height of a skip list storing n elements?

Ans: The height of a skip list storing n elements can vary. However, on average, the height of a skip list is logarithmic to the number of elements, log n. This means that as the number of elements increases, the height of the skip list also increases but at a slower rate.

Que: What is the running time for find, insert, and erase operations in a skip list?

Ans: The running time for find, insert, and erase operations in a skip list is efficient. Searching for an element (find operation) takes O (log n) time on average since it involves traversing levels of the skip list. Insertion and deletion (insert and erase operations) also take O (log n) time on average, as they require locating the appropriate position and modifying the chains at each level. The skip list's logarithmic time complexity makes these operations faster compared to linear search in a singly-linked list.

Que: How does a skip list compare to a balanced binary search tree (BST) in practice?

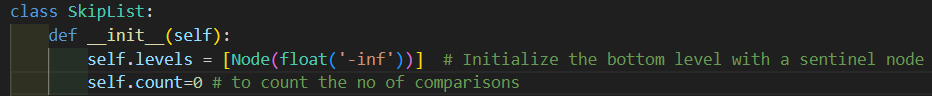
Ans: In practice, skip lists offer several advantages over balanced binary search trees. Skip lists are generally simpler to implement and maintain compared to BSTs. They also tend to have faster operations due to a smaller constant factor in their time complexity, although both structures have the same order notation. Additionally, skip lists require less total memory compared to BSTs since they do not need to store parent pointers. Moreover, skip lists are well-suited for parallel processing, making them a preferred choice in scenarios where concurrent access and modification to the data structure are important considerations. Overall, skip lists provide similar functionality to BSTs but with potential practical advantages in terms of simplicity, performance, memory usage, and parallel processing.

*Implementation Details:*

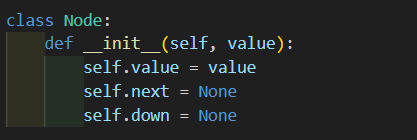
This Hybrid Data Structure (Multi-Level Skip List) is implemented using a Skip List and a multi-level data structure such as Trees and Graphs. Always the skip list is first created before the implementation of tree data structure. The skip list is implemented by using class and objects which include the functionalities of insertion, deletion, searching and printing the elements present in the skip list. Always the skip list is implemented using a Linked List as its base. So first a linked list is meant to be created in order to achieve a Skip list data structure.

A Skip List usually contains many layers of linked list but the only difference is that each layer has some elements skipped and using forward pointers the rest of the elements is connected in each layer by a random possibility. Where each subsequent layer has more elements stored (from top to bottom) in it.

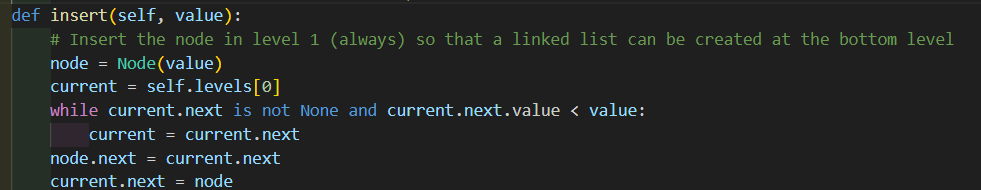
Every layer has a sentinel node which is basically the value infinity. So, a Python List is created and the sentinel node is appended every time a new layer is created. There is also a count variable to count the comparisons.



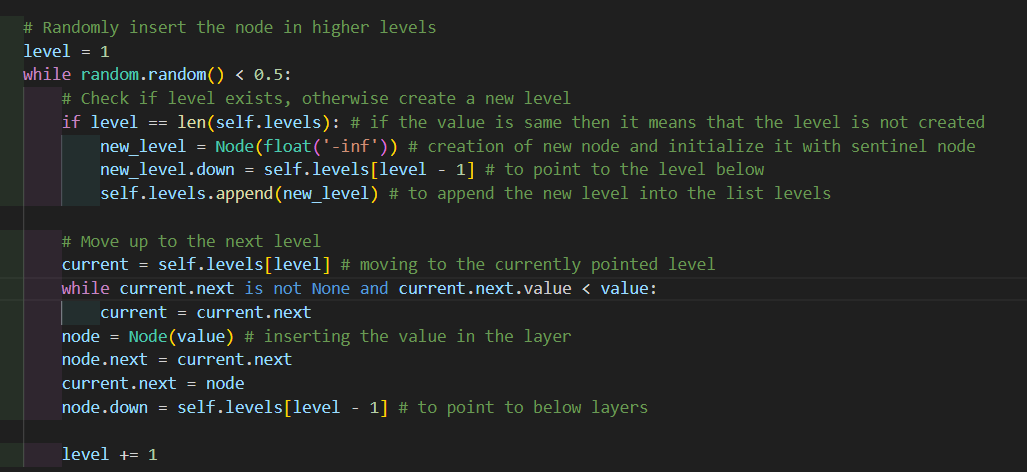
To insert the values in the linked list another class is created which contains the class variables node, next, down.



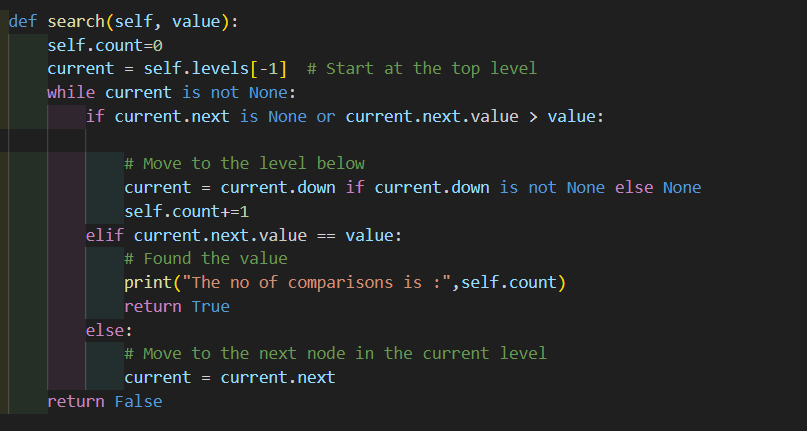
A usual linked list is created as the bottom layer or the lowest layer among all the layers in the skip list and is assigned as first element in the Python List. The user is prompted to insert the values in the linked list.



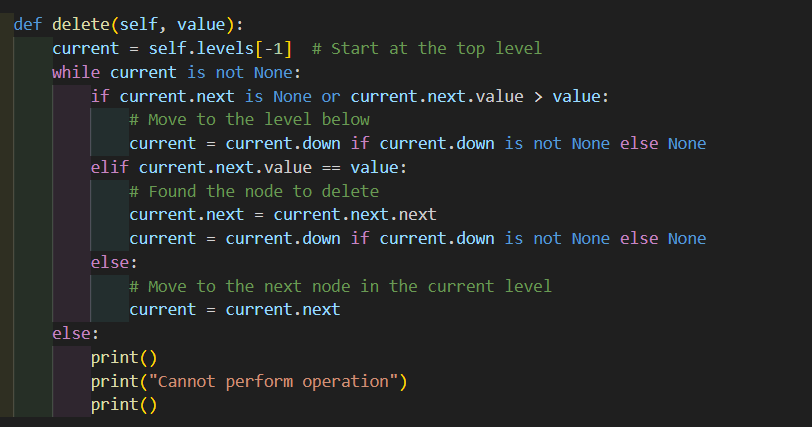
Depending upon the random value generated it is decided whether to insert the value in other layers or not. If it is to be inserted then we check if the layer is created or not, if not new layers are created. We then append the value to the layer and point the class variable down to the immediate below layer.



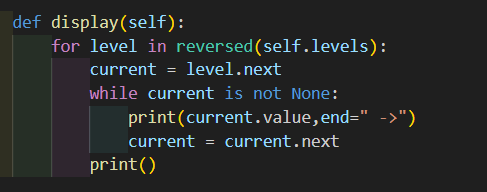
The next function that is implemented is search () meant for searching for an element in the skip list. The iteration starts from the top most layer of the skip list and start to compare the values from the top layer with the value provided by the user if the value is not found then we move to the layer below it and if the value is found then we return True otherwise it returns False.



Then we have the delete () function which starts from the top of the level and the current value is compared with the value that is provided for deleting if it is greater than the value provided then move to the next below layer. If not, then move to the next element and if found then remove the connection between the two elements.



Then there is only another function called display() meant to display the elements of the skip list through each layer.



We then have a menu driven program to get the inputs for the operation to be performed on the skip list and the value to be inserted, deleted, searched in the skip list. Then the skip list is inserted in the binary tree data structure to implement the hybrid data structure and the process is to be repeated till the expectation are met by the user.

**Design choice and Trade-Offs:**

Our very first design was to implement a python list in which the very first element in the list is a linked list with no of elements preempted by the user and then try to implement a skip list with the help of the linked list created as a basis for the skip list and for the other layers in the skip list. Then by iterating the linked list and using a random value generator will be decided whethere to append to the other layers of the skip list. Eventually it was very bad design to be used for implementation of the data structure

Practical Application:

Multi-level Skip Lists have various practical application in different domains. Here are some of the examples

**1. Database Systems:** To efficiently index and query big datasets, database systems frequently use multi-level skip lists. They offer a useful data structure for quickly indexing key-value pairs and performing searches. When a range of keys must be obtained from the database, the speed of range queries can be enhanced using multi-level skip lists.

**2.Indexing of Spatial Data:** Multi-level skip lists are appropriate for indexing of spatial data. They are effective for storing and retrieving spatial objects like polygons, rectangles, and points. Spatial queries like range searches and nearest neighbour searches can be carried out more quickly by taking advantage of the hierarchical structure of multi-level skip lists.

**3.Geographic Information System:** Applications for geographic information systems (GIS) require effective indexing and retrieval of spatial data. Multi-level skip lists can be used to store and organise spatial data, enabling operations like proximity searches and spatial overlays as well as quick access to geographic information.

**4.Web Search Engines:** Web search engines can utilise multi-level skip lists to speed up the retrieval of search results. Search engines can more quickly find pertinent materials based on search queries by indexing web pages or documents using a multi-level skip list, which enhances search speed.

**5. File systems:** To improve the effectiveness of file retrieval and searching activities, multi-level skip lists can be used. A multi-level skip list is used to organise files in a hierarchical structure so that file systems may easily find and retrieve files based on their names, qualities, or other factors.

**6.Cache Data Structure:** Multi-level skip lists can be used in cache systems to optimise data access and retrieval, according to cache data structures. The cache structure can speed up cache hits by including a multi-level skip list, which lowers the total access time for frequently visited data.

**7. Network Routing:** To increase the effectiveness of routing operations, multi-level skip lists can be used in network routing algorithms. Routing decisions can be made more rapidly and with less overhead by arranging network nodes and routes in a multi-level skip list structure.

These are just a few examples of the practical application of multi-level skip lists. The versatility and efficiency of multi-level skip lists make them well-suited for solving various complex problems in different domains that involve indexing, searching and retrieval of data.

**Advantages of incorporating the skip list in binary tree:**

There are advantages in incorporating a skip list inside a binary tree data structure. This combination can leverage the strength of both data structures to provide improved performance and efficiency for certain types of operation.

**1.** **Effective Search and Update Operations:** With an average time, complexity of O (log n), skip lists excel at quick search, insertion, and deletion operations. The tree can provide effective search paths while the skip list speeds up individual operations within each tree node when a skip list is embedded within a tree structure. When compared to employing each structure separately, this combination can speed up search and update operations overall.

**2. Balanced Design:** Tree architectures with inherent balance, such as AVL trees or red-black trees, guarantee logarithmic time complexity for search operations. You may keep this balance while enhancing the efficiency of individual node operations by including a skip list in the tree. This combination may produce a balanced structure that performs well.

**3. Adaptive Structure:** A skip list inside a tree can provide flexibility to adapt to shifting data access patterns. Based on the distribution and frequency of data access, skip lists are capable of dynamically adjusting their structure. This adaptability, especially when the data access patterns change over time, complements the static structure of a tree and enables more effective access patterns.

**4. Memory Efficiency:** Because skip lists include more forward pointers than tree structures, they often have a larger memory overhead. To reduce the memory burden, a skip list can be incorporated into a tree. The skip list offers effective operations within each tree node, lowering the total memory consumption, while the tree structure can serve as the foundation of the data organisation.

**5. Concurrent Operations:** By combining a skip list and a tree structure, concurrent operations can be supported more effectively. In situations where numerous threads or processes are simultaneously accessing and updating the data structure, skip lists' natural ability to enable concurrent insertions and deletions at various levels can be useful. When performing concurrent tasks, the tree structure can guarantee data consistency and offer extra synchronisation techniques.

Performance Analysis:

**Time complexity:**

When incorporating a skip list inside a binary search tree data structure, the time complexity of operations such as insertion, deletion and searching will depend on the specific implementation and the characteristics of the data.

**1.Insertion:** Inserting an element into a hybrid data structure combines a skip list with a binary search tree typically involves two steps. First, the element needs to be inserted into the skip list, which has an average time complexity of O (log n) for an element to reach the bottom level. Second, the skip list insertion is followed by inserting the element into the appropriate position in the binary tree data structure which has in general a time complexity of O(h) where h is the height of the BST. Therefore, the overall time complexity for insertion in a hybrid structure is typically O (log n)

**2.Deletion:** Deletion in a hybrid data structure follows a similar process as insertion. First, the element needs to be located and removed from the skip list, which has an average time complexity of O (log n). Then the element is deleted from the binary tree structure, which has a time complexity of O(h). Thus, the overall time complexity for deletion in a hybrid structure is typically O (log n).

**3.Searching:** Searching in a hybrid structure involves searching within the skip list and then within the binary tree structure. In the skip list, the average time complexity for searching an element is O(logn). If the element is found in the skip list, the search terminates. Otherwise, the search continues in the binary tree data structure, which has a time complexity of O(h). Thus, the overall time complexity for searching in a hybrid structure is typically O (log n).

**Space Complexity:**

The space complexity of a hybrid data structure that combines a skip list with a binary search tree depends on several factors, including the numbers of elements stored, the height of the tree, the number of levels in the skip list, and any additional overhead introduced by the implementation.

**1.Skip List:** The space complexity of a skip list depends on the number of elements and the number of levels it has. In a skip list with n elements and h levels, where h is the height of the skip list, the space complexity is O (n +h). The space required for the forward pointers between nodes in each level contributes to the overall space usage.

**2.Binary Search Tree:** The space complexity of a binary search tree is determined by the number of elements stored and the height of the tree. In a binary search tree with n elements, the height is typically logarithmic, resulting in a space complexity of O(n).

**3.Hybrid Data Structure:** When combining a skip list with a binary tree, the space complexity of the hybrid data structure will be the sum of the space complexities of the skip list and the binary search tree. So, the overall space complexity would be O(n+h) + O(n) = O(n+h), where n is the number of elements stored and h is the height of the skip list.

In final, the space complexity of a hybrid data structure combining a skip list with a binary search tree is typically O(n+h), where n is the no of elements stored and h is the height of the skip list.

**Analysis of performance of skip list and binary tree:**

When comparing the performance of skip list and binary trees, its important to consider various factor such as search, insertion, deletion and space efficiency.

**1.Search Efficiency:**

**Skip List:**  With an average time complexity of O(log n), where n is the number of elements, skip lists often provide effective search operations. The usage of several levels and the hierarchical structure of the skip list enable quick traversal and skipping of elements during search.

**Binary Tree**: Balanced binary trees having an O(log n) time complexity, such as AVL trees or red-black trees, also offer effective search functionality. Because the tree is balanced, the height is guaranteed to remain logarithmic, resulting in effective search performance.

**2.Insertion and Deletion Efficiency:**

**Skip List:** Skip lists typically have an average time complexity of O(log n), making them excellent for insertion and deletion operations. The skip list's structure makes it possible to insert and remove nodes quickly without having to rebalance the tree, as is necessary with binary trees**.**

**Binary Tree:** Balanced binary trees, which typically have an O(log n) time complexity, also provide effective insertion and deletion operations. However, under some circumstances, balancing operations could be necessary to keep the tree balanced, adding extra time overhead.

**3.Space Efficiency:**

**Skip List:** The forward pointers that connect components across levels must be stored in additional space, which is needed for skip lists. In terms of the quantity of elements stored, the space complexity is O(n).

**Binary Tree:** Binary Tree: To store each node, which contains data and pointers to its left and right child nodes, binary trees need space. When it comes to the quantity of elements stored, the space complexity is similarly O(n).

**4.Simplicity of Implementation:**

**Skip List:** When compared to balanced binary trees, skip lists may be easier to implement. The data structure is simpler to construct and maintain because of its hierarchical structure and skip pointers.

**Binary Tree:** Binary Tree: In order to guarantee that the tree maintains its balance during insertion and deletion operations, implementing balanced binary trees, such as AVL trees or red-black trees, requires additional complexity.

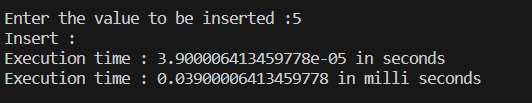
Overall, skip lists and binary trees have similar performance characteristics in terms of search, insertion and deletion operation. Skip lists may have a slight advantage in terms of insertion and deletion operations. Skip lists may have slight advantage in terms of insertion and deletion efficiency due to their skip pointers, while binary trees may have better space efficiency.

Experimental Evaluation:

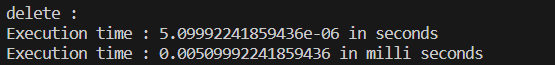
To measure the time taken for the performance of the hybrid data structure we use the inbuilt python library ‘time’ which measures the time of each operations.

We have used the time function time.perf\_counter() which is especially used for measuring the time taken for the code to execute completely for every function present in the skip list. So we have measured the insert, delete, search function in our hybrid data structure.

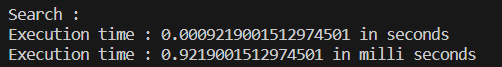
**Insert Function:**



**Delete Function:**



**Search Function:**



DataSets:

A dataset for a skip list and binary search tree hybrid data structure is a group of data points arranged in a way that incorporates the benefits of both data structures. In this hybrid dataset, the binary search tree's capacity to preserve data sorting and handle range queries is combined with the skip list's capacity to offer quick search and insertion operations. The dataset contains key-value pairs for each data point, where the key is utilised for sorting and searching. The skip list element of the hybrid data structure adds more linked list levels, each level containing a portion of the components in the dataset. The temporal complexity of actions like searching and adding new items is decreased by these levels' creation of shortcuts that make traversal and searching faster.

The binary search tree component makes sure that the elements are sorted according to their keys by maintaining the elements in a hierarchical structure with each node having a left and right child. This enables effective range queries, which enable the speedy retrieval of a subset of data inside a certain range of keys. The hybrid dataset strikes a balance between search performance and sorted organisation by merging the skip list and binary search tree. It takes advantage of the skip list's logarithmic search time and the binary search tree's logarithmic insertion and deletion time to reduce time complexity. The hybrid dataset is ideal for applications that involve range-based queries or data analysis, databases, indexing systems, or other circumstances that call for both quick search operations while keeping the data's sorted order.

DISCUSSION:

The primary advantage of multi-level skip lists is their ability to handle complex problems like range queries and spatial data indexing efficiently. Range queries involve retrieving all elements within a specified range. In a multi-level skip list, the levels can be used to quickly identify the starting and ending positions of the range, reducing the number of elements that need to be examined.

Spatial data indexing involves organizing and searching for data based on spatial relationships. In multi-level skip lists, each level can represent a different level of spatial granularity. This enables efficient retrieval of spatial objects by navigating through the levels and zooming in on the desired region.

By incorporating multiple levels, multi-level skip lists provide a balance between fine-grained access and faster search operations. They allow for more focused searches by traversing fewer elements in higher levels, while still maintaining a broader view of the data in lower levels. This makes them well-suited for handling complex problems that require efficient range queries and spatial data indexing.

**Here are some instances of real-world application for multi-level skip lists:**

Streaming data in real time can be stored and processed using multi-level skip lists. This is because they may be effectively added to and searched, even when new information is added to the list. They are therefore perfect for uses like fraud detection, stock market analysis, and keeping an eye on social media. Large datasets can be stored and analysed using multi-level skip lists in big data analytics. This is due to the fact that they may be effectively searched even for extremely rare or particular data pieces. They are therefore perfect for programmes like genome sequencing, natural language processing, and machine learning. The following are some restrictions on multi-level skip lists: multi-level skip lists may require more storage than other data structures, like hash tables, do. This is due to the fact that they store several pointers for each node. **Performance:**

For some operations, such as deletion, multilevel skip lists can be slower than other data structures. They need more pointer updates, which is why. Implementing multi-level skip lists presents the following difficulties: selecting the appropriate number of levels ,For some operations, multi-level skip lists are used.

**New data structures**: Similar to multilevel skip lists, new data structures are also being developed by researchers. Compared to multilevel skip lists, these new data structures might be more advantageous due to their improved performance or reduced spatial complexity. Multi-level skip lists are an all-around strong and adaptable data structure that may be applied in a wide range of RealWorld scenarios. Hey provide an excellent compromise between ease of use, space complexity, and performance.

**Limitations of Multi-Level Skip Lists**

Multi-level skip lists are a type of probabilistic data structure that can be used to store and retrieve data efficiently. They are similar to linked lists, but they have multiple levels, which allows them to perform operations such as insertion and deletion more quickly.

However, multi-level skip lists also have some limitations. One limitation is that they can be more difficult to implement than other data structures, such as balanced binary trees. Additionally, multi-level skip lists can be less efficient for storing and retrieving data that is not evenly distributed.

**Difficulty of implementation**: multi-level skip lists are more complex than other data structures, such as linked lists or balanced binary trees. This means that they can be more difficult to implement correctly, and they may require more memory.

**Inefficiency for unevenly distributed data:** multi-level skip lists are designed to be efficient for storing and retrieving data that is evenly distributed. However, they can be less efficient for storing and retrieving data that is not evenly distributed. This is because the data may not be evenly distributed across the different levels of the skip list.

**Future Improvements for Multi-Level Skip Lists:**

There are a number of potential future improvements for multi-level skip lists. One improvement would be to develop a more efficient implementation of the data structure. Additionally, researchers could explore ways to make multi-level skip lists more efficient for storing and retrieving data that is not evenly distributed.

More efficient implementation: Researchers could develop a more efficient implementation of the multi-level skip list data structure. This could be done by using more efficient data structures for storing the data, or by using more efficient algorithms for performing operations on the data structure.

More efficient for unevenly distributed data: Researchers could explore ways to make multi-level skip lists more efficient for storing and retrieving data that is not evenly distributed. This could be done by using a different data structure for storing the data, or by using a different algorithm for performing operations on the data structure.

Overall, multi-level skip lists are a promising data structure that has the potential to be used in a variety of applications. However, there are some limitations to the data structure that need to be addressed before it can be widely adopted. Researchers are actively working on improving the performance and efficiency of multi-level skip lists, and it is likely that the data structure will become more widely used in the future.

CONCLUSION:

We can conclude by saying that we have seen the overview of what is a hybrid data structure, its advantages in general, a Data structure called Multi level skip list implemented using linked lists which is a skip list with multiple levels that provide a checkpoint of sorts that allow us to skip various elements while traversing , thus greatly reducing the time complexity with average case of time complexity being O(Log N) and worst case scenario with O(Log N) . We have seen its actual implementation and the concept of including large skip list and storing them as nodes of a binary search tree. We have also discussed general limitations, the specific limitations of our chosen data structure, and how to improve the limitations in the near future, such as improving the probabilistic data structure we form the hybrid with, and every point has been elaborated in detail howe report.

We have researched where our particular data structure can be used in real life situations, and gaining some insights on how the modern technological advancements demand more technological insights and innovation, and thus creating the birth of the branch of data science called Big Data, and probabilistic data structure as such. We have successfully llearned insertion,mented multilevel skip list with insertion, deletion and searching.

We, members of Sorting Spartans appreciate the DSA faculty for taking the time to give us project that really makes one think outside the box and grow interest in topics that were beyond the scope of our syllabus and creating areas in fields of data science that are practically applied. We also thank Mata Amritanandamayi Devi for providing us with a wonderful facility where we can nourish and grow.

Reference:

<https://www.javatpoint.com/skip-list-in-data-structure>

<https://www.cs.rpi.edu/academics/courses/fall20/csci1200/lectures/21_hybrid_data_structures.pdf>

<https://dzone.com/articles/introduction-probabilistic-0>

<https://www.geeksforgeeks.org/skip-list/>

<https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/>