# MsDS - Data Structures and Algorithms Module No: 5

## Revision from Last Lecture

1. 1. Linear search and binary search are essential algorithms for finding data, with binary search offering a significant reduction in time complexity.
2. 2. Sorting algorithms have different designs tailored to specific requirements, including factors like stability, adaptivity, and in-place functioning.
3. 3. The choice of sorting algorithm depends on the physical storage location of the data and the size of the list, with in-memory sorting being faster but limited by RAM capacity, while sequential storage sorting can handle larger datasets but with potentially longer access times.

## Expected Learning Outcomes

1. 1. Understand the fundamental principles and uses of hash tables for effective data storage and retrieval.
2. 2. Understand the differences in the features and applications of HashSets and HashMaps.
3. 3. Understand the fundamental ideas behind Binary Trees and Binary Search Trees, as well as their importance in the representation of hierarchical data.
4. 4. Understand the nuances of height-balanced BSTs, especially AVL Trees, and appreciate how crucial balancing is to achieving peak performance.

## HashTables

• HashTables are data structures that enable efficient storage and retrieval of data through a process called "hashing."

• Hashing involves generating an output from an input key, which is used as an index to locate the corresponding data in an array.

• HashTables effectively manage collisions, which occur when multiple keys generate the same hash index, through techniques like chaining and open addressing.

## HashMaps and HashSet

• HashMaps are widely used data structures that associate keys with corresponding values, enabling efficient data retrieval.

• HashMaps are designed to efficiently manage key-value pairs and use a hashing algorithm to determine the storage location for data.

• HashMaps are dynamic in size, allowing them to adjust and redistribute key-value pairs as they expand, maintaining efficiency in space utilisation and access time.

## Binary Trees & Binary Search Trees

• Binary trees are hierarchical data structures with nodes that can have a maximum of two offspring.

• Balanced binary trees minimise differences in depth between left and right subtrees, ensuring optimum efficiency for various operations.

• Binary trees are adaptable and effective for operations like insertion, deletion, and retrieval of data.

## Height-balanced BSTs – AVL Trees

• Efficiency is prioritised in data structures, and AVL Trees provide a solution to the issue of imbalance in Binary Search Trees (BSTs).

• AVL Trees maintain balance through their balancing factor, which falls between -1 and 1, and undergo a rebalancing process if the factor exceeds this range.

• Rebalancing in AVL Trees is achieved through rotations, including single right, single left, right-left, and left-right rotations, based on the identification of the imbalanced child and the location of the imbalance.

## Priority Queues

• Priority queues assign priority levels to elements, allowing for the efficient insertion and deletion of items based on their importance.

• Implementation of priority queues can be done using arrays, linked lists, or more efficient data structures like heaps.

• Priority queues are used in various fields such as computer algorithms, operating systems, and simulation systems to prioritise and manage tasks effectively.

## Heaps

• Heaps are tree-based data structures that guarantee a hierarchical arrangement of elements, facilitating efficient operations.

• Heaps have a heap property where the value of a node is either larger or smaller than its descendants, allowing for efficient determination of the largest or smallest element.

• Heaps are highly regarded in computer science due to their efficiency in essential tasks like inserting, removing, and restoring the heap property, with a logarithmic time complexity. They are commonly used in algorithms like heap sort.

## Important Terminologies

**HashTables:** Hash-based data structures are used to store key-value pairs by using a hash function to calculate an index inside an array.

**HashSet:** The set is a data structure that consists of distinct elements, without explicit identifiers, and is based on the underlying concepts of HashTables.

**Binary Trees:** Hierarchical structures, characterised by nodes possessing a maximum of two offspring, are often used for the purposes of data storage and retrieval.

**Binary Search Trees (BSTs):** A distinct variant of binary trees is characterised by the presence of a value assigned to each node, with the property that all nodes in the left subtree possess values that are less than the value of the root node, while all nodes in the right subtree possess values that are larger.

**AVL Trees:** Self-balancing binary search trees (BSTs) are data structures that preserve balance by guaranteeing that the height difference between the left and right subtrees is no more than one.

**Heaps:** Tree-based data structures that possess the heap property and are specifically designed to meet the requirements of priority queues.

## Summary

❖ HashTables are efficient data structures that use distinct keys to establish a correspondence with values, while HashMaps allow the inclusion of null values and keys.

❖ Binary Search Trees (BSTs) guarantee the ordering of nodes, with all nodes in the left subtree having values less than the root, and all nodes in the right subtree having values larger than the root.

❖ Priority queues handle elements based on their priority, with efficient access or removal of the element with the highest or lowest priority. Heaps, particularly binary heaps, are commonly used for priority queues.

THANK YOU