**DATA STRUCTURES AND ITS OPERATIONS**

**1. Array**

Advantages:

- Fast access to elements using the index (O(1) time complexity).

- Simple and easy to use.

Disadvantages:

- Fixed size, which can lead to wasted space or lack of space.

- Insertion and deletion of elements can be expensive (O(n) time complexity).

Operations:

- Access: O(1)

- Search: O(n)

- Insertion: O(n) (worst case, shifting elements)

- Deletion: O(n) (worst case, shifting elements)

**2. Linked List**

Advantages:

- Dynamic size.

- Easy insertion and deletion of elements.

Disadvantages:

- No random access, have to traverse from the head (O(n) time complexity).

- Extra memory space for the pointers.

Operations:

- Access:O(n)

- Search: O(n)

- Insertion: O(1) (at the head)

- Deletion: O(1) (if deleting the head)

**3. Stack**

Advantages:

- Last-In-First-Out (LIFO) structure.

- Easy to implement with arrays or linked lists.

Disadvantages:

- Limited access to only the top element.

- Can cause stack overflow if the stack size limit is exceeded.

Operations:

- Push (insert): O(1)

- Pop (delete): O(1)

- Peek (top): O(1)

**4. Queue**

Advantages:

- First-In-First-Out (FIFO) structure.

- Useful in scenarios like task scheduling.

Disadvantages:

- Limited access to only the front and rear elements.

- Can cause overflow if the queue size limit is exceeded.

Operations:

- Enqueue (insert): O(1)

- Dequeue (delete): O(1)

- Peek (front): O(1)

**5. Hash Table**

Advantages:

- Very fast access, insertion, and deletion on average (O(1) time complexity).

- Efficiently handles large datasets.

Disadvantages:

- Potential for collisions, which need to be handled (e.g., with chaining or open addressing).

- Can be less efficient if poorly designed or if a poor hash function is used.

Operations:

- Insert: O(1)

- Delete: O(1)

- Search: O(1)

**6. Binary Tree**

Advantages:

- Hierarchical structure.

- Efficient insertion, deletion, and search operations (O(log n) on average for balanced trees).

Disadvantages:

- Can become unbalanced, leading to O(n) operations.

- Requires additional overhead for pointers.

Operations:

- Insert: O(log n) (on average)

- Delete: O(log n) (on average)

- Search: O(log n) (on average)

1. **Binary Search Tree (BST)**

Advantages:

- Sorted structure.

- Efficient search, insert, and delete operations for balanced trees.

Disadvantages:

- Can become unbalanced, leading to O(n) operations.

- Requires additional overhead for pointers.

Operations:

- Insert: O(log n) (on average)

- Delete: O(log n) (on average)

- Search: O(log n) (on average)

**8. Heap**

Advantages:

- Efficiently supports the priority queue operations.

- Good for algorithms like heapsort and Dijkstra's algorithm.

Disadvantages:

- Does not support search operations efficiently (O(n) time complexity).

- Can be complex to implement.

Operations:

- Insert: O(log n)

- Delete (extract max/min): O(log n)

- Peek (max/min): O(1)

**9. Graph**

Advantages:

- Can model complex relationships between entities.

- Supports a wide range of algorithms (e.g., shortest path, spanning tree).

Disadvantages:

- Can be complex to implement and understand.

- Requires significant memory for adjacency matrices or lists.

Operations:

- Add Vertex: O(1)

- Add Edge:O(1) (adjacency list), O(1) (adjacency matrix)

- Remove Vertex: O(V + E) (adjacency list), O(V^2) (adjacency matrix)

- Remove Edge: O(E) (adjacency list), O(1) (adjacency matrix)

**10. Trie**

Advantages:

- Efficiently handles dynamic sets of strings.

- Provides fast prefix search.

Disadvantages:

- Requires significant memory space for large alphabets.

- Can be complex to implement.

Operations:

- Insert: O(L) (L is the length of the string)

- Delete: O(L)

- Search: O(L)