High-performance software design in computer science is greatly dependent on the efficiency of algorithms and data structures. Different operations are supported by various data structures and the expenses for these operations can be estimated in terms of time complexity, or space complexity.

Common Data Structure Operations:

1. Access/Search:

Description: Finding the location of a particular element in the data structure.
Examples: Searching for an element in an array, finding a key in a hash table.

2. Insertion:

Description: Adding a new element to the data structure.
Examples: Inserting a node into a linked list, adding an element to an array.

3. Deletion:

Description: Removing an element from the data structure.
Examples: Deleting a node from a linked list, removing an element from an array.

4. Traversal:

Description: Visiting and processing each element in the data structure.
Examples: Iterating through elements in an array, traversing nodes in a tree.

5. Sorting:

Description: Arranging elements in a specified order.

Examples: Sorting an array using algorithms like quicksort or mergesort.

6. Merging:

Description: Combining two data structures into one.
Examples: Merging two sorted arrays or merging two sorted linked lists.

7. Splitting:

Description: Dividing a data structure into multiple parts.
Examples: Splitting an array into two halves or splitting a linked list.

Cost Estimation:

Time Complexity:

- Definition: Measures the amount of time an algorithm takes with respect to its input size.
- Notation: Big O notation (O(f(n))).
- Example: If an algorithm's time complexity is O(n), it means the running time grows linearly with the input size.

Space Complexity:

- Definition: Measures the amount of memory an algorithm uses with respect to its input size.
- Notation: Big O notation (O(f(n))).
- Example: If an algorithm's space complexity is O(1), it means the memory usage remains constant regardless of the input size.

Examples:

Array:

- Access/Search: O(1) – constant time if the index is known.
- Insertion/Deletion: O(n) – linear time for shifting elements.
- Traversal: O(n) – linear time to visit each element.

Linked List:

- Access/Search: O(n) – linear time to traverse the list.
- Insertion/Deletion: O(1) – constant time for adding/removing elements at the beginning (with a reference to the head).
- Traversal: O(n) – linear time to visit each node.

Binary Search Tree:

- Access/Search: O(log n) – logarithmic time for balanced trees.
- Insertion/Deletion: O(log n) – logarithmic time for balanced trees.
- Traversal: O(n) – linear time for in-order traversal.

Hash Table:

- Access/Search/Insertion/Deletion: O(1) – constant time on average for well-distributed hash functions.
- Traversal: O(n) – linear time to visit each element.

Related Posts:

1. Review of C programming language

2. Concepts of Data and Information
3. Abstract Data Types