Page 1: Introduction to DSA

Data Structures and Algorithms (DSA) form the foundation of efficient programming.

They help solve problems by organizing and manipulating data effectively.

1. Data Structures:

- Containers for storing data in a structured way.

- Examples: Arrays, Linked Lists, Stacks, Queues, Trees, Graphs.

2. Algorithms:

- Step-by-step instructions to solve problems.

- Examples: Searching, Sorting, Graph Traversal.

Importance of DSA:

- Enhances problem-solving skills.

- Optimizes code performance and memory usage.

Page 2: Common Data Structures

1. Arrays:

- Fixed-size collection of elements of the same type.

- Access: O(1), Insertion/Deletion: O(n).

Example:

```
int arr[] = {1, 2, 3};
```

2. Linked Lists:

- Dynamic collection of nodes.

- Access: O(n), Insertion/Deletion: O(1).

Example:

```
struct Node {
  int data;
  Node* next;
};
```

3. Stacks and Queues:

- Stack: LIFO (Last In First Out).

- Queue: FIFO (First In First Out).

Example:

```
stack.push(10);
queue.enqueue(5);
```

4. Trees:

- Hierarchical structure.

- Types: Binary Trees, Binary Search Trees, AVL Trees.

Example:

```
struct TreeNode {
  int data;
  TreeNode* left;
  TreeNode* right;
};
```

Page 3: Algorithms

1. Searching:

- Linear Search: O(n).

- Binary Search: O(log n).

Example (Binary Search):

```
int binarySearch(int arr[], int x) {
  int low = 0, high = n - 1;
  while (low <= high) {
    int mid = (low + high) / 2;
    if (arr[mid] == x) return mid;
    else if (arr[mid] < x) low = mid + 1;
    else high = mid - 1;
  }
  return -1;
}
```

2. Sorting:

- Bubble Sort, Merge Sort, Quick Sort.

Example (Bubble Sort):

```
void bubbleSort(int arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]);
```

```
    }
  }
}
```

3. Graph Algorithms:

- BFS: Breadth-First Search.

- DFS: Depth-First Search.

Example (DFS):

```
void dfs(int node, bool visited[], vector<int> adj[]) {
  visited[node] = true;
  for (int neighbor : adj[node]) {
    if (!visited[neighbor]) dfs(neighbor, visited, adj);
  }
}
```