DATA STRUCTURES & THEIR APPLICATIONS- REPORT

Submitted By, Mukund Prasad H S

TASK 2

1. Arrays

Applications:

- Image processing
- Spreadsheets
- Memory allocation in computers
- Database records

Why Arrays are Optimal:

What needs to be stored:

- Homogeneous data elements of fixed size
- Data that requires frequent random access

Cost of operations:

- O(1) for access and modification of elements
- O(n) for insertion and deletion (except at the end)

Memory use:

- Efficient for fixed-size data
- Contiguous memory allocation

Ease of implementation:

- Simple and straightforward
- Supported natively in most programming languages

2. Stacks

Applications:

- Function call management (call stack)
- Undo mechanisms in text editors
- Expression evaluation and syntax parsing
- Backtracking algorithms

Why Stacks are Optimal:

What needs to be stored:

- Data that follows Last-In-First-Out (LIFO) principle
- Temporary data that needs to be processed in reverse order

Cost of operations:

- O(1) for push and pop operations

Memory use:

- Efficient for managing temporary data
- Can be implemented using arrays or linked lists

Ease of implementation:

- Simple conceptually
- Easy to implement with just push and pop operations

3. Queues

Applications:

- Task scheduling in operating systems
- Breadth-First Search in graphs
- Print job spooling
- Handling of requests in web servers

Why Queues are Optimal:

What needs to be stored:

- Data that follows First-In-First-Out (FIFO) principle
- Tasks or data that need to be processed in the order they arrive

Cost of operations:

- O(1) for enqueue and dequeue operations

Memory use:

- Efficient for managing data in order
- Can be implemented using arrays or linked lists

Ease of implementation:

- Straightforward concept
- Basic operations (enqueue and dequeue) are simple to implement

4. Trees

Types with their applications:

Binary Search Trees (BST):

- Implementing associative arrays
- Database indexing

AVL Trees and Red-Black Trees:

- Self-balancing data structures for maintaining sorted data
- Implementing sets and maps in programming languages

B-Trees and B+ Trees:

- File systems
- Database management systems

Trie (Prefix Tree):

- Autocomplete features
- IP routing tables

Heap:

- Priority queues
- Heap sort algorithm

Why Trees are Optimal:

What needs to be stored:

- Hierarchical data
- Data that requires fast search, insert, and delete operations

Cost of operations:

- O(log n) for search, insert, and delete in balanced trees
- O(h) where h is the height of the tree (can be O(n) in worst case for unbalanced trees)

Memory use:

- More memory-intensive than linear data structures
- Efficient for storing and retrieving hierarchical data

Ease of implementation:

- More complex than linear data structures
- Different types of trees have varying levels of implementation complexity