|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **LAB** | **Pg.No** | **Signature** |
| **1.** | **Lab** **1:** Array Operations  **AIM:** Implement array operations.  **PROGRAM:**  Write a Python program to:   * Traverse a list and print all elements. * Insert an element at a given index (shift elements using list slicing). * Delete an element at a given index (remove and shift). * Search for an element using linear search (return index or -1). * Update an element at a given index. * Test with input list [5, 3, 8, 1, 9]: insert 7 at index 2, delete index 1, search 8, update index 3 to 4. |  |  |
| **2.** | **Lab 2:** Singly Linked List and Stack  **AIM:** Build linked lists and stack operations.  **PROGRAM:**  Implement a singly linked list in Python with a Node class (data, next) and methods:   * Insert at front and end. * Delete at front and traverse. * Implement a stack using a linked list with push, pop, peek, and is empty. * Test linked list: insert [10, 20, 30], delete front. Test stack: push [5, 10], pop once, peek. |  |  |
| **3.** | **Lab 3:** Queue Implementations  **AIM:** Implement queue variants.  **PROGRAM:**  Implement in Python:   * Array-based queue using a list with enqueue, dequeue, and front. * Circular queue (list-based) handling wrap-around. * Test with: enqueue [10, 20, 30], dequeue twice, enqueue 5 in circular queue. |  |  |
| **4.** | **Lab 4:** Sorting Algorithms  **AIM:** Implement and compare sorting algorithms.  **PROGRAM:**  Implement insertion sort and merge sort in Python:   * Compare performance using time module on a random list of   50 integers.   * Test with input: [64, 34, 25, 5, 22]. |  |  |
| **5.** | **Lab 5**: Binary Search Tree and Traversals  **AIM:** Code tree traversals and BST operation.  **PROGRAM:**  Implement a binary tree in Python with a Node class and:   * In-order traversal. * Implement a BST with insert, search, and delete (handle leaf and one-child cases). * Test with: insert [50, 20, 70], search 20, delete 20, print in-order traversal. |  |  |
| **6.** | **Lab 6:** Heap and Priority Queue  **AIM:** Implement heap-based priority queues  **PROGRAM:**  Implement a min-heap in Python using a list with:   * Insert, extract-min, and heapify (use heapq optionally). * Implement heapsort using min-heap. * Test with: [4, 30, 3, 25, 16, 9]. |  |  |
| **7.** | **Lab 7:** Hash Table Implementation  **AIM:** Code hash tables with collision handling.  **PROGRAM:**  Implement a hash table in Python with:   * Open addressing (linear probing) using a list. * Test with: insert keys [12, 44, 13, 88, 23], search 44, delete 13. |  |  |
| **8.** | **Lab 8:** B-Tree Implementation  **AIM:** Implement B-trees for indexing.  **PROGRAM:**  Implement a B-tree in Python (order m=3) with:   * Insert and search operations. * Test with: insert [10, 20, 5, 6, 12, 30], search 6. |  |  |
| **9.** | **Lab 9:** Graph Representation and Traversals  **AIM**: Implement graph representations and traversals.  **PROGRAM:**  Implement a graph in Python using an adjacency list (dictionary of lists).  • Implement depth-first search (DFS) and breadth-first search (BFS).  • Test with graph: nodes [A, B, C, D], edges [A-B, B-C, C-D, A-D]. |  |  |
| **10.** | **Lab 10**: Dijkstra’s and Bloom Filter  **AIM:** Implement shortest path algorithm and advanced data structure.  **PROGRAM:**  Implement Dijkstra’s algorithm in Python for shortest paths using heapq.   * Implement a Bloom filter with multiple hash functions (hashlib) for membership testing. * Test Dijkstra’s with graph: nodes [A, B, C, D], edges [A-B:4, A-C:2, B-D:5, C-D:8]. Test Bloom filter with: insert [apple, banana], query [apple, orange]. |  |  |

**Lab** **1:**

**AIM:** To implement array operations.

**PROGRAM:**

Write a Python program to:

* Traverse a list and print all elements.
* Insert an element at a given index (shift elements using list slicing).
* Delete an element at a given index (remove and shift).
* Search for an element using linear search (return index or -1).
* Update an element at a given index.
* Test with input list [5, 3, 8, 1, 9]: insert 7 at index 2, delete index 1, search 8, update index 3 to 4.

**DESCRIPTION:** The program uses list slicing for insertion and deletion, sequential scanning for linear search, and index-based assignment for update. These operations collectively simulate basic array manipulation techniques used in data structures.

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 2:**

**AIM:** To build linked lists and stack operations.

**PROGRAM:**

Implement a singly linked list in Python with a Node class (data, next) and methods:

* Insert at front and end.
* Delete at front and traverse.
* Implement a stack using a linked list with push, pop, peek, and is empty.
* Test linked list: insert [10, 20, 30], delete front. Test stack: push [5, 10], pop once, peek.

**DESCRIPTION:** This program demonstrates dynamic storage with singly linked lists and stack operations. Lists support insertion at both ends and deletion at the front, while stacks follow LIFO (Last-In-First-Out) behaviour implemented using linked nodes.

**CODE:**

**OUTPUT:**

**CONCLUSION**: This lab illustrates singly linked list fundamentals and stack behavior. Using dynamic nodes for both structures enables flexible insertion, deletion, and efficient stack management following LIFO principles.

**Lab 3:**

**AIM:** To implement queue variants.

**PROGRAM:**

Implement in Python:

* Array-based queue using a list with enqueue, dequeue, and front.
* Circular queue (list-based) handling wrap-around.
* Test with: enqueue [10, 20, 30], dequeue twice, enqueue 5 in circular queue.

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 4:**

**AIM:** To implement and compare sorting algorithms.

**PROGRAM:**

Implement insertion sort and merge sort in Python:

• Compare performance using time module on a random list of 50 integers.

• Test with input: [64, 34, 25, 5, 22].

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 5**:

**AIM:** To code tree traversals and BST operation.

**PROGRAM:**

Implement a binary tree in Python with a Node class and:

* In-order traversal.
* Implement a BST with insert, search, and delete (handle leaf and one-child cases).
* Test with: insert [50, 20, 70], search 20, delete 20, print in-order traversal.

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 6:**

**AIM:** To implement heap-based priority queues

**PROGRAM:**

Implement a min-heap in Python using a list with:

* Insert, extract-min, and heapify (use heapq optionally).
* Implement heapsort using min-heap.
* Test with: [4, 30, 3, 25, 16, 9].

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 7:**

**AIM:** To code hash tables with collision handling.

**PROGRAM:**

Implement a hash table in Python with:

* Open addressing (linear probing) using a list.
* Test with: insert keys [12, 44, 13, 88, 23], search 44, delete 13.

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 8:**

**AIM:** To implement B-trees for indexing.

**PROGRAM:**

Implement a B-tree in Python (order m=3) with:

* Insert and search operations.
* Test with: insert [10, 20, 5, 6, 12, 30], search 6.

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 9:**

**AIM**: To implement graph representations and traversals.

**PROGRAM:**

Implement a graph in Python using an adjacency list (dictionary of lists).

• Implement depth-first search (DFS) and breadth-first search (BFS).

• Test with graph: nodes [A, B, C, D], edges [A-B, B-C, C-D, A-D].

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**

**Lab 10**:

**AIM:** To implement shortest path algorithm and advanced data structure.

**PROGRAM:**

Implement Dijkstra’s algorithm in Python for shortest paths using heapq.

* Implement a Bloom filter with multiple hash functions (hashlib) for membership testing.
* Test Dijkstra’s with graph: nodes [A, B, C, D], edges [A-B:4, A-C:2, B-D:5, C-D:8]. Test Bloom filter with: insert [apple, banana], query [apple, orange].

**DESCRIPTION:**

**CODE:**

**OUTPUT:**

**CONCLUSION:**