Unit III

Linear list representation, operations insertion, deletion and searching, hash table representation, hash
functions, collision, resolution-separate chaining, open addressing-linear probing, quadratic probing,
double hashing.
Priority Queues: Definition, ADT, Realizing a Priority Queue using Heaps, Definition, insertion,
Deletion, External Sorting-Model for external sorting. Search Trees: Binary Search Trees, Definition,
ADT, Implementation, Operations-Searching, Insertion and Deletion

Linear List Representation

1. **Operations on Linear Lists**:
   - **Insertion**: Adding elements at different positions in the list.
   - **Deletion**: Removing elements from different positions.
   - **Searching**: Finding elements within the list.

Hash Table Representation

1. **Hash Functions**:
   - **Mapping Keys to Buckets**: Techniques to convert keys into indices in a hash table.
   - **Uniform Distribution**: Aim for even distribution to minimize collisions.
2. **Collision Resolution**:
   - **Separate Chaining**: Each bucket contains a linked list of elements hashing to the same index.
   - **Open Addressing**:
     - **Linear Probing**: Checking the next location in case of collision.
     - **Quadratic Probing**: Using a quadratic function to find the next available slot.
     - **Double Hashing**: Using two hash functions to resolve collisions.

Priority Queues

1. **Definition and ADT**:
   - **Abstract Data Type**: Allows insertion and deletion based on priority.
   - **Heaps**: Commonly used to implement priority queues, particularly binary heaps.

2. **Realizing a Priority Queue using Heaps**:
    - **Heap Structure**: Maintaining a binary heap to prioritize elements.
    - **Operations**: Insertion, deletion according to their priority level.

External Sorting

1. **Model for External Sorting**:
    - **Handling Large Data Sets**: Sorting data that doesn't fit into memory.
    - **Disk Access Optimization**: Minimizing disk I/O operations.

Search Trees

1. **Binary Search Trees (BST)**:
    - **Ordered Data Structure**: Left child < Parent < Right child.
    - **Operations**: Searching, Insertion, Deletion maintaining the BST property.

1. Linear List Representation and Operations

Linear List Representation:

Linear lists represent a sequence of elements where each element has a successor and a predecessor (except for the first and last elements).

```
// Example: Linear list representation using arrays

const int MAX_SIZE = 100;

int myList[MAX_SIZE];

int listSize = 0;
```

Operations: Insertion, Deletion, Searching

Insertion: Adding elements to the list at a specified position.

Deletion: Removing elements from the list at a specified position.

Searching: Finding elements within the list.

```
// Example: Insertion, Deletion, Searching in a linear list

void insertElement(int value, int position) { /* ... */ }
void deleteElement(int position) { /* ... */ }

int searchElement(int value) { /* ... */ }
```

2. Hash Table Representation, Hash Functions, Collision, Resolution

Hash Table Representation:

A hash table is a data structure that maps keys to values using a hash function.

```
// Example: Hash table representation using arrays

const int TABLE_SIZE = 100;

int hashTable[TABLE_SIZE];
```

Hash Functions and Collision Resolution:

Hash Function: Maps keys to indices in the hash table.

Collision: Occurs when two keys hash to the same index.

Collision Resolution: Methods like separate chaining, linear probing, quadratic probing, and double hashing resolve collisions.

3. Priority Queues

Definition and ADT:

A priority queue is a data structure where each element has an associated priority.

Realizing a Priority Queue using Heaps:

A heap is a tree-based data structure where the parent node has a higher priority than its children.

```
// Example: Realizing a Priority Queue using Heaps

#include <queue>

using namespace std;


priority_queue<int> myPriorityQueue;

myPriorityQueue.push(5);
myPriorityQueue.push(10);

int topElement = myPriorityQueue.top(); // topElement = 10

myPriorityQueue.pop();
```

4. External Sorting

Model for External Sorting:

External sorting involves sorting large datasets that don't fit entirely in memory.

5. Search Trees: Binary Search Trees

Definition and ADT:

A binary search tree (BST) is a tree-based data structure where the left child is smaller and the right child is greater than the parent.

Operations: Searching, Insertion, and Deletion:

Searching: Finding elements within the tree.

Insertion: Adding elements to the tree while maintaining its properties.

Deletion: Removing elements from the tree while preserving its structure.

```
// Example: Binary Search Tree operations

struct Node {

int key;

Node* left;

Node* right;

};


Node* search(Node* root, int key) { /* ... */ }

Node* insert(Node* root, int key) { /* ... */ }

Node* deleteNode(Node* root, int key) { /* ... */ }
```