Sorting

- ➔ ***Selection sort***
  - ➢ *Find the minimum in the unsorted part of the list*
  - ➢ *Swap it with the first element in the unsorted part*
  - ➢ *$O(n^2)$ in time*
  - ➢ *O(1) in space*
- ➔ ***Merge sort***
  - ➢ *Divide the list into 2 half*
  - ➢ *Recursively sort each half*
  - ➢ *Merges the two sorted part*
  - ➢ *O(n log n) in time*
  - ➢ *O(n) in space*
- ➔ ***Insertion sort***
  - ➢ *Build a sorted section of the list*
  - ➢ *Insert element to sorted part from the unsorted part*
  - ➢ *$O(n^2)$ in time*
  - ➢ *O(1) in space*
- ➔ ***Quick sort***
  - ➢ *Select a* pivot *element from the list*
  - ➢ *Partition the array into 2 list*
  - ➢ *Recursively sort the two sub-lists*
  - ➢ *$O(n^2)$ in the worst case, O(n log n) average in time*
  - ➢ *O(log n) in space*
- ➔ ***Bubble sort***
  - ➢ *Compare adjacent elements in the list*
  - ➢ *If they are in the wrong order, swap them*
  - ➢ *Continue iterating through the list*
  - ➢ *$O(n^2)$ in worst-case time*
  - ➢ *O(1) in space*

Implementation of sorting

- ➢ *Array Iteration*
- ➢ *Array Recursion*
- ➢ *Linked list Iteration*
- ➢ *Linked list Recursion*

Notes by Izam Mohammed

Stack

A stack data structure follows the Last-In, First-Out (LIFO). Principle. The last item added to the stack is the first one to be removed

Terminologies

- *Stack Overflow: push when the stack is full*
- *Stack Underflow: pop when there is no elem in the stack*
- *Infix Notation: The operator is written in between the operands -> A + B*
- *Prefix Notation(Polish Notation): The operator is written before the operands -> +AB*
- *Postfix Notation (Suffix Notation): The operator is written after the operands -> AB+*

Operations

- *Push: Add an element to the top -> O(1)*
- *Pop: Remove and return the element at the top -> O(1)*
- *Peek: Return the element to the top -> O(1)*
- *Print: Print all items -> O(1)*
- *Count: Return the number of data points -> O(1)*

Types

- *Register Stack: a set of registers in the CPU*
- *Memory Stack: call stack*

Advantages of Stack

- *Simplicity: LIFO*
- *Fast Operation: Push and Pop*
- *Memory Efficiency: Not require complex memory allocation*
- *Helps in function calls*

Disadvantages of Stack

- *Limited Access*
- *No random Access*
- *No Search*
- *Fixed-size*
- *Lack of flexibility*

Application of Stack

- *Function Call Management: used to keep track of functions*
- *Expression Evaluation*

Notes by Izam Mohammed

- *Backtracking*
- *Undo Functionality*
- *Browser Navigation*
- *Matching Parenthesis*

Problems to solve

- *Valid Parenthesis*
- *Reverse string*
- *Delete middle element*
- *Delete individual words in a sentence*
- *Queue using stack*
- *Infix to postfix and prefix*
- *Evaluate postfix and prefix expression*

Queue

It is a linear data structure that follows the First-In, First-Out (FIFO) principle. The elements that were added first is the one that is removed first

Operations

- *Enqueue: add to the back of the queue -> O(1)*
- *Dequeue: Remove and return the element at the front -> O(1)*
- *Front: Return the first element -> O(1)*
- *Rear or Back: Return the last value -> O(1)*
- *is Empty -> O(1)*
- *Count -> O(1)*
- *Print -> O(n)*

Types

- *Simple Queue: Enque at the end*
- *Circular Queue: Act as a circular ring*
- *Priority Queue: Arrange elements based on some priority*
- *Double Ended Queue (Dequeue): can inset at start and end*

Implementation

- *Array*
- *Linked List*

Advantages

- *FIFO order*
- *Essential for BFS*
- *Prevent Data Loss*

Notes by Izan Mohammed

- ➢ *It can be used to implement other data structures*
- ➢ *Can handle large amounts of data with ease*

Disadvantages

- ➢ *Insert and deleting in the middle is time-consuming*
- ➢ *Searching is not efficient*
- ➢ *Maximum size must be predefined*

Application

- ➢ *Ticket Counter Line*
- ➢ *Network*
- ➢ *Shared Resources*
- ➢ *CPU task scheduling*
- ➢ *Call center system*

Hash Table

Store and manage in a way that allows for efficient retrieval and storage. It is based on the concept of the hash function, which maps data to specific locations or buckets within the table.

Terminologies

- ➢ *Hash function: A function that takes an input (key) and produces a unique numeric value (hash code) that represents the location in the hash table where the associated data will be stored*
- ➢ *Bucket: A storage location within the hash table where data is stored. Multiple key-value pairs with the same hash code may be placed in the same bucket.*
- ➢ *Collision: A situation where two or more keys produce the same hash code, causing a collision in the table.*
- ➢ *Collision Resolution: the process of handling collisions when multiple keys map to the same location in the hash table. Common collision resolution techniques include chaining, open-addressing*
- ➢ *Load Factor: the ratio of the number of stored key-value pairs to the total number of buckets in the hashtable.*
- ➢ *Resizing: The process of increasing and decreasing the number of buckets*
- ➢ *Key: Unique identifier of a data element*
- ➢ *Value: The data associated with the key*
- ➢ *Collision avoidance: Techniques employed to minimize the likelihood of collisions, such as high-quality hash functions*
- ➢ *Hash Code: A numeric value produced by the hash function based on the input key*

Notes by Izam Mohammed

Advantages

- *Efficient Data Retrieval*
- *Constant-Time Average Lookup (searching)*
- *Versatile*
- *Memory-efficient*
- *Collision handling*
- *Dynamic Sizing*

Disadvantages

- *Hash collisions*
- *Hash function dependency*
- *No inherent order*
- *Difficult to sort*
- *Not ideal for small data*

Application

- *Used in searching and indexing massive volumes of data*
- *Used in cryptography to create a digital signature*
- *Phone books*
- *DB indexing*
- *Network routing*
- *Password storage*
- *Frequency counting*

Operations

- *Insertion -> O(1) avg*
- *Lookup (Get) -> O(1) avg*
- *Deletion -> O(1) avg*
- *Size (Count) -> O(1)*
- *Contains -> O(1) avg*
- *Key Enumeration: retrieve all keys -> O(n)*
- *Value Enumeration: retrieve all value -> O(n)*
- *Clear -> O(n)*
- *Rehashing -> O(n)*

Notes by Izan Mohammed

Collision Resolution Techniques

- *Chaining (closed addressing): using a LinkedList*
- *Open Addressing (probing): find the location where no value*
- *Robin Hood hashing: remove the elem based on the distances*
- *Linear Probing: checking the next location*
- *Quadratic probing: use a quadratic function to determine the next step in probing*
- *Double Hashing: use a second hash function to determine the step size in probing*

Notes by Izam Mohammed