**Syllabus** (structured):

(the ones marked in red have to be discussed again)

Algorithms

design

asymptotic

math operations

Searching

linear

binary

jump

Sorting:

Bubble

insertion

selection

quick

merge

heap

DS (general):

Array

String

Linked List

Stack

Queue

Heap

Maps/Dictionary/Hash

Trees:

Binary Tree

Search

Traversal

pre

post

inorder

search

Graphs:

Directed

weighted

BFS

DFS

Hash:

Collision

methods for handling

Advanced:

recursion

dynamic programming

greedy algorithms

Syllabus (from Mettle)

**Detailed focus on first four modules**

|  |  |
| --- | --- |
| Sub-Module 1 | 1. Introduction to DS and algorithms: Definition, background. |
| Algorithm Fundamentals and basic operations | 2. Analyze and design algorithms |
|  | 3. Asymptotic notations |
|  | 4. Basic math operations on integer and float datatypes (add, sub, mul, div, sqrt, counting, exponentiation). |
|  | 5. Basic recursion algorithm: example factorial. |
|  |  |
|  |  |
| Sub-Module 2 | 1. Array, String, Linked lists. (detail) |
| Introduction to Data Structures |  |
|  |  |
|  |  |
|  |  |
|  | 2. Stack, Queue, Heap: Using Arrays and lists. (detail) |
|  | 3. Maps/Dictionary/Hash table. (detail) |
|  | 4. Trees, Binary Tree, Binary search tree. (simple creation) |
|  | 5. Graph: Different representation of graph, BFS, DFS. |
|  |  |
|  |  |
| Sub-Module 3 | 1. Notion of Hashing and Collision |
| Hash Tables |  |
|  |  |
|  |  |
|  |  |
|  | 2. Hash Functions - properties, simple has function |
|  | 3. Methods for Collision Handling - separate chaining, rehashing |
|  | 4. Ordered Dictionary - ADT specification, applications |
|  |  |
|  | ASSESSMENT 3 - 20 objective-type questions |
|  |  |
| Sub-Module 4 | 1. Counting sort, insertion sort, bubble sort. |
| Searching and Sorting | 2. Heap sort |
|  | 3. Quick sort merge sort. |
|  |  |
|  |  |
|  |  |
|  | 4. Search element: unordered array, ordered array, Heap |
|  |  |
| Sub-Module 5 | 1. Different representation of trees. |
| Tree | 2. Tree basic operations: Create tree, insert node, delete node etc. |
|  | 3. Different traversal of trees: in-order, pre-order, post-order, level-order. |
|  | 4. Introduction to binary search tree |
|  | a. Create, insert new node in binary search tree |
|  | b. Search in BST. |
|  | 5. Example of balanced binary tree: AVL, RB (**10 minutes** discussion) |
|  |  |
|  |  |
| Sub-Module 6 | 1. Different representation of graph. |
| Graphs | 2. BFS, DFS, Topological sort explanation |
|  |  |
|  |  |
|  |  |
|  | 3. Importance and Applications of graph: shortest path  4. Simple graph creation (directed, weighted) |
|  |  |
|  |  |
| Sub-Module 7 | Introduction to advance Algorithms (max **45 minutes** discussion) |
| Introduction to advance Algorithms | 1. Recursion(Prerequisite of DP) |
|  | 2. Dynamic Programing Algorithm: |
|  | a. Tabulation vs Memoization |
|  | b. Optimal Substructure Property |
|  | c. Overlapping sub problems |
|  | d. How to sovle dynamic programic problems |
|  | e. Examples: cutting rod problem, matrix- chain multiplication |
|  |  |
|  |  |
|  |  |
|  | 3. Greedy algorithms: introduction and algorithms only. |
|  |  |
|  |  |
| Sub-Module 8 | Doubts session |
| Doubts session |  |
|  |  |

# Algorithms

## Algorithm Analysis:

how many resources (space, time)

# Asymptotic function:

Big O

Big (theta)

Big (omega)

# Big O Notation

representation of the complexity of an algorithm

performance of algorithm

scalability

worst case (upper bound)

O(1) : Constant Time

O(n) : Linear Time Algorithms

complexity increases linearly

directly proportion to the number of inputs

Polynomial

O(n²) : Quadratic

directly proportion to the square of inputs

O(n³) : Cubic

directly proportion to the cube of inputs

O(2^n) : Exponential

complexity doubles for every element in the input

O(n!) : Factorial Time Algorithms

TSP (travelling salesman problem)

O(log n): Logarithmic

complexity increases logarithmically

O(n log n): Linear Logarithmic

# Searching Algorithms

Linear Search (Sequential Search)

Binary Search

Jump Search

Interpolation Search

Exponential Search

Fibonacci Search

## Linear Search (Sequential Search)

- simplest

- inefficient

- (brute-force)

- time O(n)

- space O(1)

- unsorted & small set data

## Binary Search (Logarithmic Search)

- divide & implement

- data set should be sorted

- time O(log(n))

- space O(1)

## Jump Search (Logarithmic Search)

- sqrt(length)

# Sorting algorithms:

Bubble Sort

Insertion

Selection

Merge

Quicksort

Heapsort

74 88 11 20 18

74 11 88 20 18

**2 5 8 9** 4 2 0 11

**2 5 8 x 9**  2 0 11

**2 5 x 8 9** 2 0 11

**2 x 5 8 9** 2 0 11

**2 4 5 8 9** 2 0 11

**3** 5 1 2 4

**1**  5 3 2 4

**1** 5 3 2 4

**1 2** 3 5 4

**1 2 3**  5 4

**1 2 3 4**  5

**1 2 3 4 5**

3 5 4 2 1

3 5 4 2 1

3 5 4 2 1

3 5

# Graph:

- data structure for storing *connected* data

- nodes(vertices) and edges

Vertex: represents entity (locations, people)

degree (number of edges)

edge: relationship between the entities

ordered pairs

outgoing

incoming

V = {1,2,4,6,8,9}

E = { (6,2), (6,1), (6,8), (8,4), (1,4)}

Undirected Graph

Directed Graph

DAG (Directed Acyclic Graph)

MultiGraph

Simple Graph

Weighted and Unweighted

Complete Graph

Connected Graph

Path: sequence of alternating vertices & edges

Cycle: path that starts & ends at the same vertex

Forest: graph without cycles

Tree: connected graph with no cycles

Adjacency Matrix representation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 | 8 |
| 1 | 0 |  | 1 | 1 |  |
| 2 |  | 0 |  | 1 |  |
| 4 |  |  | 0 |  |  |
| 6 | 1 | 1 |  | 0 | 1 |
| 8 |  |  | 1 | 1 | 0 |

Adjacency List

6 🡪 1 2 8

2 🡪 6

1 🡪 6 4

4 🡪 6 8

8 🡪 6 4

9 🡪

## Famous graph Libraries for Java

Guava (Google)

Apache Commons

Sourceforge JUNG

JGraphT