4th Semester :→ B1

Basic C | Basics C++ | OOPs

30 hrs :→ 5 days x 6 hrs

Data Structures & Algorithms Road Map :→

↳ C / C++ or Java → DMA m, c, r, f
(C++ → oops, Java → oops) DMA new & delete

Arrays → 1D, 2D, MD / Strings (character arrays)

DSA

↓ Algorithms | ↓ Data Structures (Implemented) | ↓ Data Structures (Built-in) STL Collections

↳ Searching :→
1. Linear
2. Binary
3. Recursive Binary xxx
4. Jump Search
5. Interpolation Search

Time Complexity
Big O Notation

Sorting Algos :→
* Bubble Sort
* Selection Sort
* Insertion Sort } $O(n^2)$ → swap, shift, S(c)
* Merge Sort — n log n → O(n)
* Quick Sort — n log n → O(1)
* Heap Sort
* Count Sort
* Radix Sort } Non comparison Algos → Bucket Sort
* Wave Sort
* Shell Sort

Special Algos :→
1. Kadane's Algo
2. Sieve Of Eratosthenes (Prime Numbers)
3. Euclidean Algo → HCF or GCD
4. Pythagorean Triplets
5. Rabin Karp Algo

Data structure :→
Linear :→
* Arrays
* Strings
* Stacks
* Queues
* Linked Lists

Standard Template Library #include<stack>
#include <bits/stdc++.h>
* stack → LIFO
* queue → FIFO
* priority_queue → Max H, Min H — java.util
* deque — O(1) → Complete Binary Tree
* map <k, v> → ordered → asc; unordered → hashing
* set → ordered → asc; unordered → hashing
* list → forward_list → SLL; list → DLL
* Vector → dynamic array.

Non-linear Data structures :→
* Trees
→ Normal Tree
→ Binary Tree
→ Traversals (DFS) → Pre, In, Post
→ BFS (Queue) → Level Order Traversal
→ Binary Search Tree + Disadvantages } Search Trees → code, coder, coding
→ AVL Tree
→ Red Black Tree
→ K-d Tree
→ Segment Tree
→ Fenwick Tree / Binary Index Tree
→ B / B+ Trees
→ Trie → Phone Book | Dictionary | LCP

Graphs :→
* Types
* Representation → Adj Mat / Adj List
* Traversals → DFS / BFS
* Cycle Detection → DFS / BFS
* Topological Sort → Linear Ordering [DAG]
* Kahn's Algorithm → Cycle Detection (Indegree)
* Shortest Distance Algos:
Relaxation of edges {
↳ Dijkstra's → (-ve) weights
↳ Bellman Ford
↳ Floyd Warshall's } (-ve) weights
↳ MST → Minimum Spanning Tree ↓ (Disjoint Set) union → rank, size; findparent
↳ Prim's Algo
↳ Kruskal's
↳ Strongly Connected Components
↳ Kosaraju's Algorithm

* Greedy Algos
* Recursion → Back Tracking; → Dynamic Programming
1. Recursion 2. Memoisation 3. Tabulation 4. Space Optimization

* Bit Manipulations / Bit Masking
CN / OSI / ADA / File Handling

* Duck Typing
* Operator Overloading
* List Comprehension
* Map, Filter, Reduce
Python

Stream API
Generics
Annotations
Lambda Expressions
Java

C → malloc → (size) [garbage] 40 bytes
calloc → (n, size) [10 blocks 4 bytes] 40 bytes
int → 4 bytes ↳ ↓ 0

new → default
int → 0
float → 0.0
String → null
bool → false
ch → null

int * arr = new int [size]; 1D

2D Array → Square Matrix
n x n — n → rows, n → cols
int ** twoD = new int * [n]; creates n rows
{ (i = 0; i < n; i++)
twoD[i] = new int [n]; ↓ cols
}
row

2D Array Non-Square Matrix
n x m — n → rows, m → cols
int ** twoD = new int * [n]; creates n rows
i → 0 to n
twoD[i] = new int [m] ↓ cols
(DRY)