Effective Teaching of “Data Structures”

(A Module in the CSEDU – Certificate Program in CS Education)

Objective

The aim of this module is to help teachers in colleges/universities (attendees) improve their teaching of the Data Structures course, as per the AICTE syllabus for it (given at the end of this document). I.e. After this module, the attendees will improve their teaching of this course leading to improved learning by their students and more students achieving the learning outcomes of the AICTE course. This module is part of the Certificate Program in CS Education initiative (csedu.iiitd.ac.in). The main learning outcomes of ***this module*** are: (At the end of the module, an attendee will:)

- Have clearer understanding of importance of this course in a CSE program, the desired learning outcomes established in the AICTE course, and the course syllabus
- For each topic in the AICTE syllabus, have a deeper understanding of concepts, what is important, how to teach, what type of assignments to give, etc.

The focus of the module will be to help in delivering the Essential Learning Outcomes of the AICTE course syllabus. Some advanced topics may also be discussed, based on the inputs from the attendees.

Module Syllabus

Each weekly session will discuss one of the modules in the AICTE syllabus, teaching methodology for it, and the type of assignments to give. Slides / Notes that can be used for teaching students will be discussed and provided; sample assignments will also be provided.

The week-wise syllabus for this module is:

Wk	**Module (from AICTE syllabus to be discussed)**	**Approach**	**Self-work for the week**
1	Motivation, role of data structures in algorithm design, discussion of asymptotic complexity	What and Why? (i) Role of data-structures a) in algorithms (e.g., second-max, Dijkstra etc.) b) Data Structures being the core problem themselves (Google, Database queries) (ii) Measuring solutions, asymptotic complexity, max, min, binary search, Table showing running time	Discussion on problems related to the role of data-structures, binary search, etc. Watching videos on related topics.

		and the problem size that can be reasonably solved. (iii) Preview of the Course: (Data Structures for Dictionary, Priority Queue, Disjoint Sets) and a preview of why simple solutions can be expensive.	
2	Data-type vs data-structure, arrays, lists		Programming Assignment on application of Stacks.
3	Trees and Binary trees	Implementation, access, NIL pointers, properties, complete binary tree	Discussion on problems related to trees and representations.
4	Priority Queues	Start with insert and max, gradually get into deletions	Programming assignment related to real life application of priority queues. Problems on priority queues. Visualization videos.
5,6,7	Binary Search Trees	(i) BST as generalization of binary search: show that the search procedure can be “modeled” using a BST, and this also motivates definition of BST (ii) Explain why height is an important parameter and first show search, insert, delete in O(height) time. (iii) Motivate AVL trees and explain the idea behind rotation.	Programming assignments related to applications of AVL trees. Visualization videos.
8	Hash tables		Programming assignment related to real life application of hash tables. Visualization video.
9	Tries and extensions		Programming assignment related to real life application of tries. Visualization video

10	Sorting: mergesort: recurrences, etc.	(i) Explain the idea behind divide and conquer (ii) Write down recurrences for mergesort, and explain the solution using a tree. Also show the recursion tree for mergesort.	Problems on divide-and-conquer technique, assignment related to understanding various sorting algorithms.
11	Quicksort, partitioning, selection, average case ?	(i) Explain the partitioning procedure in quicksort, and how it leads to selection and sorting algorithms (ii) Show worst case behaviour by examples, and explain cases where running time of quicksort would be O(n log n) (e.g., almost equal partition, or almost equal partition happening every alternate recursive step etc.) (iii) Meaning of average case analysis and intuitively explain why it would be O(n log n) (iv) Randomization in quicksort and selection	Visualization videos.
12	Representation and motivation, trees, BFS (Module 7)	Adjacency array, list, what is easier in one, but difficult in other, graphs in real world, trees, properties, BFS through maze traversal etc, time complexity, some properties about tree and non-tree edges	Problems related to applications of graphs, and visualization of BFS.
13	DFS: directed and undirected (Module 7)	What happens inside the recursive dfs procedure. How can it be implemented using stacks. Role of arrival and departure time of nodes in designing algorithms. How are arrival and departure times of head and tails of various edges related?	Visualization videos on DFS
14	BFS and DFS Applications (Module 7)	BFS: Connected components, Girth of a graph and checking if the graph is bipartite. DFS: Topological sort.	Programming assignment on graphs.

15	Advanced topics based on feedback from the participants		

Schedule

The module will meet online once a week. In addition, a weekly help session to clear doubts and to help with the assignment will be provided through TA s. Details about joining these sessions will be provided later.

- **Weekly Session**: <day>, 4-530 pm (or 430 pm to 6 pm)
- **Weekly help Session**: <day>, 4-530 pm (or 430 pm to 6 pm)

Text to be used for the Module

The textbook suggested in the AICTE syllabus will be used as the basis of this module.

Resources to be provided to attendees.

- Lecture Notes / ppt for the different topics in the course – which they can use in their own teaching
- Some sample assignments for each of the major modules
- Some online resources which the attendees can use to revise/update their knowledge, …

Post Module Support

- A mailing list will be created and for a year, once every few months an online session of all attendees will be planned to share experiences and discuss challenges being faced
- …

Appendix: Revised AICTE Syllabus for the course (Tentative)

CS301	**Data Structure & Algorithms**	**3L:0T: 4P**	**5 credits**	**Pre-Reqs: ESC201**

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

1. Understanding abstract specification of data-structures and their implementation.
2. Understanding time and space complexity of programs and data-structures.
3. Knowledge of basic data-structures, their applications and relative merits.
4. Ability to identify the relevant data-structure operations in an algorithm and choose suitable data-structures for these by analyzing the trade-offs involved in terms of time and space complexity.

Desirable/Advanced (<= 3):

1. Amortized Complexity
2. Use of randomization in data-structures

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Introduction and basic terminology (1 week)	(i) Notion of data-structures and algorithms. (ii) $log n, n, 2^n$: understanding growth of these functions, and applications (binary search and extensions to similar problems)	**T1: Chapter 3** (i) Explain the interplay between algorithms and data-structures (ii) Explain the meaning of worst and average case (iii) Examples of $O(\ )$, Motivation behind	(i) Worst/average case analysis for small pseudo-codes (ii) Prove/disprove why a function $f(n)$ is $O(g(n))$ (and similarly for Ω). (iii) Variations on binary search with applications, recursive

	(ii) Worst-case, average-case time/space complexity and their relative merits. (iii) Asymptotic Notation: $O(\ ), \Omega(\ )$	asymptotic analysis (large n and ignoring constants). (iv) Discuss recurrence relation for binary search.	and iterative implementation of binary search with applications to problems.
Module 2: Abstract Data-types, Arrays, Linked Lists, Stacks, Queues Dictionary ADT, Trees, Binary Trees (2.5 weeks)	(i)Abstract data-type (ADTs): arrays and linked list ADTs. (ii) Stacks, Queues: ADTs and implementations using arrays, linked lists. (iii) Doubly linked lists: ADT and implementation (iv) Dictionary ADT: implementation using array, linked lists, binary search. (v) Tree ADT and examples (vi) Implementation of trees and basic traversal algorithms (vii) Binary trees and inorder traversal	**T1: chapter 4, Chapter 6** (i) Explain the difference between specification and implementation of ADTs. (ii) Discuss stack implementation using arrays of fixed size and linked list, and relative merits. (iii) Circular array and linked list implementation of queues, and relative merits. (iv) Discuss some applications of stacks: post-fix notation, matching parentheses etc. (v) Dictionary ADT: discuss why neither arrays nor linked list is a good implementation, but sorted arrays are good for searching. (vi) Discuss situations where we need to store hierarchical data. (vii) Discuss pre-order and post-order ,	(i) Implementation of stacks with application to a problem. (ii) Implementation of queues with application to a problem. (iii) Implementation of trees with applications for storing and accessing hierarchical data. (iv) Show multiple implementatch >>ions of the same ADT: findmin and insert (arrays, linked list, stack).

		traversals and applications in finding height and similar problems.	
Module 3: Priority Queues and Heaps (1 week)	(i) Priority Queue ADT (ii) Definition of heaps (iii) Implementation of Priority Queues using heaps and running time analysis (iv) Implementation of heaps using arrays. (v) Heap-sort	**T1: Chapter 7.1-7.3** (i) Start with FindMin and Insert, and a simple $O(1)$ time and $O(1)$ space algorithm. Buildup towards DeleteMin. (ii) Discuss algorithm for heaps: inserting an element, modifying an element, and deleting the minimum element. (iii) Discuss why array implementation is more efficient than balanced binary trees	(i) Array implementation of heaps and application to a problem. (ii) k-ary heaps: compare with binary heaps (both in theory and practice)
Module 4: Binary Search Trees, AVL Trees, 2-4 trees (3 weeks)	(i) Binary Search Trees: definition and some basic algorithms. (ii) Implementation of Dictionary ADTs using Binary Search trees and running time analysis (iii) AVL trees: height balance condition, rotations, and implementation of dictionary ADT (iv) 2-4 Trees: Multi-way search trees, implementation of dictionary ADT, Informal discussion of extension to B-trees.	**T1: Chapter 9.1, 9.2, 9.4** (i) Discuss algorithms for finding predecessor or successor, and similar problems in Binary Search Trees. (ii) Explain why height of a Binary Search Tree may not remain $O(logn)$. (v) Explain how the height balance condition ensures that the height is $O(logn)$, and how rotation changes the structure of a tree. (vi) Explain why rotations in AVL trees restore height balance condition (vii) Explain why 2-4 trees have $O(logn)$	(i) Implementation of AVL trees with search, insert, delete operations and application to a problem. Comparison with unbalanced Binary Search Trees. (ii) Implementation of 2-4 trees with search, insert, delete operations and application to a problem. (iii) Comparison of the two implementations above.

		height and the running time of insert/delete operations. (viii) Discuss how balanced binary tree data-structures can implement a priority queue	
Module 5: Hash tables, tries (2 weeks)	(i) Map ADT (ii) Hash Tables and implementation of Map using Hash Tables (iii) Design of hash functions (iv) Collision resolution schemes: chaining, open addressing schemes like linear probing, quadratic probing, double hashing. (v) Applications of Hashing: finding duplicates, set intersection, etc. (vi) Tries: implementation of Map ADT using tries. (vii) Compressed tries and suffix tries.	**T1: Chapter 8.1-8.3, Chapter 11.3** (i) Explain the difference between Map and Dictionary ADT. (ii) Discuss how hash functions can have non-numeric keys as input. (iii) Discuss the relative merits of hash tables and balanced binary search trees. (iv) Discuss how hashing can be a substitute for sorting in many cases. (v) Explain why tries can be better than balanced binary search trees in some settings. (vi) Explain how compressed tries save space (vii) Discuss real-life applications of tries.	(i) Implementation of hash tables with applications to a problem. (ii) Implementation of tries and applications to a problem.
Module 6: Sorting, Selection (1.5 weeks)	(i) bubble sort, insertion sort, selection sort. (ii) Merge sort and divide and conquer paradigm (iii) Quick sort: average and worst case analysis,	**T1: Chapter 10.1, 10.2, 10.4, 10.5, 10.7**	Implementation of sorting algorithms and comparison of running times on large data-sets.

	randomized quicksort (intuitive explanation) (iv) Selection based on partitioning ideas used in QuickSort.	(i) Discuss why $O\left(n^2\right)$ time algorithms can be useful sometimes (small data, data nearly sorted etc.) (ii) Only the recurrence for merge sort and mention that divide-and-conquer paradigm will be explored more in algorithms course. (ii) Discuss the randomized splitting algorithm for quicksort and selection and explain intuitively the expected running time.	
Module 7: Graphs, representations and traversal algorithms, applications of BFS, DFS (2.5 weeks)	(i) Graph ADTs and applications (ii) Adjacency list and adjacency matrix representations and relative merits (iii) Basic graph definitions: paths, cycles, trees, spanning trees, connected components, Euler's formula. (iv) Depth First Search Traversal algorithm for directed graphs: classification of edges into forward, back and cross edges. Applications to cycle finding, topological sort in directed acyclic graphs, finding connected components. Running time analysis. (v) Breadth first search algorithm: implementation using queues, shortest path	**T1: Chapter 12.1-12.4** (i) Discuss the wide applicability of graphs including social networks, internet. (ii) Discuss time and space complexity of basic operations using adjacency list and adjacency matrix. (iii) Discuss why trees have $n - 1$edges. (iv) Discuss how DFS can be thought of as exploration with backtracking. Explain the role of stack in DFS. (v) Explain how BFS can be thought of as traversal along shortest paths and implementation using queues.	(i) Graph implementation using adjacency list and DFS/BFS traversal with applications.

	tree property. Running time analysis	(vi) Formal proof of why BFS yields a shortest path tree.	

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Amortized Complexity (1 week)	(i) Binary counter (ii) Binomial Heaps (iii) Extendible arrays	**T1: Chapter 5.1.3,** **R2: Chapter 17.1** (i) Explain the motivation behind amortized analysis. (ii) Analyze amortized complexity by explicit calculation of total number of operations after $n$steps.	(i) Exercises on amortized time complexity (e.g., a queue using two stacks etc.) (ii) Implementation of extendible arrays.
Module 2: Randomization in data-structures (1 week)	(i) Skip Lists (ii) randomized quick-sort	**T1: Chapter 8.4, 10.2** (i)Explain the idea of expectation of a random variable. (ii) How does expected running time translate to actual running time (iii)Recurrence for expected running time randomized quicksort	(i)Exercises on calculation of expectation of a random variable. (ii) Implementation of skip lists and comparison with AVL trees.

Suggested text books / Online lectures or tutorials:

1. “Data Structures and Algorithms in Java”, by Michael T. Goodrich and Roberto Tamassia, John Wiley & Sons; 3rd Edition.

2. “Data Structures and Algorithms in Python”, by Michael T. Goodrich and Roberto Tamassia, Wiley, 1st Edition.

Suggested reference books / Online resources:

1. NPTEL video series, Data-structures and Algorithms, Instructor: Naveen Garg.

2. Introduction to Algorithms, 4TH Edition, Thomas H Cormen, Charles E Lieserson, Ronald L Rivest and Clifford Stein, MIT Press/McGraw-Hill.

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: G1, G2

Other discipline GAs to which this course may contribute somewhat: CS1, CS2, CS4