

## **Data Structures and Algorithms**

(adapted from the Final Report of the Computing Curricula 2001 project,  
a joint undertaking of the IEEE-CS and the ACM)

Data structures and algorithms are fundamental to computer science and software engineering. The real-world performance of software is strongly influenced by:

- the choice of appropriate data structures for storing the software's data in order to effectively store, manipulate, and retrieve the data values;
- the use of algorithms that are appropriate and efficiency across the various layers of implementation.

In addition, the study of data structures and algorithms can provide insight into the intrinsic nature of a problem as well as possible solution techniques for the problem that are independent of programming language, programming paradigm, computer hardware, or any other implementation aspect.

An important part of computing is the ability to use both data structures algorithms that are appropriate to particular purposes and to apply them, recognizing their strengths and weaknesses, and their suitability in particular contexts. Efficiency is a pervasive theme throughout this area.

### ***Topics:***

1. Binary search trees
2. Representations of graphs (adjacency list, adjacency matrix)
3. Depth- and breadth-first traversals
4. Shortest-path algorithms (Dijkstra's and Floyd's algorithms)
5. Transitive closure (Floyd's algorithm)
6. Minimum spanning tree (Prim's and Kruskal's algorithms)
7. Topological sort
8. Hash tables, including collision-avoidance strategies
9. Asymptotic analysis of upper and average complexity bounds
10. Identifying differences among best, average, and worst case behaviors
11. Big O
12. Standard complexity classes
13.  $O(N \log N)$  sorting algorithms (Quicksort, heapsort, mergesort)
14. Brute-force algorithms
15. Greedy algorithms
16. Divide-and-conquer
17. Backtracking
18. Branch-and-bound
19. Heuristics
20. Pattern matching and string/text algorithms

### ***Learning objectives:***

1. Solve problems involving binary search trees, especially preorder, inorder, and postorder traversals; heaps, and balanced binary search trees.

2. Solve problems using the fundamental graph algorithms, including depth-first and breadth-first search, single-source and all-pairs shortest paths, transitive closure, topological sort, and at least one minimum spanning tree algorithm.
3. Implement the most common quadratic and  $O(N \log N)$  sorting algorithms.
4. Design and implement an appropriate hashing function for an application.
5. Design and implement a collision-resolution algorithm for a hash table.
6. Choose the appropriate data structure for modeling a given problem.
7. Explain the use of big O notation to describe the amount of work done by an algorithm.
8. Use big O to determine the time and space complexity of simple algorithms.
9. Discuss the computational efficiency of the principal algorithms for sorting, searching, and hashing.
10. Deduce recurrence relations that describe the time complexity of recursively defined algorithms.