BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

The Ministry of Education of the Azerbaijan Republic
The State Oil Company of the Azerbaijan Republic
Baku Higher Oil School

Information Technology Department

Data Structures and Algorithms

Bachelor's Degree in Computer Engineering
Syllabus, Spring 2025

Instructor: Azar Aliyev
Course Code: CExxx
Course Credit: 4
Instruction Language: English
Email: azar.aliyev@bhos.edu.az

Schedule:

- CE23.1/CE23.2: Lecture /Laboratory – Monday, Wednesday, Thursday, Friday

Office Hours:

- Tuesday / Wednesday 11:00 – 15:00

Course Description

This course introduces fundamental concepts of data structures and algorithms, providing a solid foundation for designing efficient and scalable software solutions. Students will learn various data structures, algorithms, and their applications in solving real-world problems. Emphasis will be placed on understanding the principles behind these structures and algorithms, as well as analyzing their time and space complexity.

Objectives

- To develop a deep understanding of fundamental data structures such as arrays, linked lists, stacks, queues, trees, and graphs.
- To explore various algorithms for searching, sorting, and manipulating data.
- To analyze the time and space complexity of algorithms and make informed decisions about algorithmic choices.
- To apply data structures and algorithms to solve complex computational problems.
- To enhance problem-solving skills and algorithmic thinking through hands-on coding exercises and projects.

Course Outcomes

By the end of the course, students should be able to:

- Design and implement basic and advanced data structures.
- Choose appropriate data structures and algorithms for solving specific problems.
- Analyze the time and space complexity of algorithms.
- Apply algorithmic thinking to devise efficient solutions to computational problems.
- Implement and test algorithms using a programming language of choice.
- Demonstrate proficiency in problem-solving through practical projects.

Prerequisites

Students are expected to have a strong foundation in programming, preferably in a language such as Python, Java, or C++. Prior exposure to basic computer science concepts, including loops, conditionals, functions, and recursion, is essential. A solid understanding of mathematical concepts such as set theory and basic discrete mathematics will be beneficial. Familiarity with basic data types and their manipulations is also recommended.

Grading

Type	**Grade**
Activity Point	10%
Homework	20%
*Quiz 1	15%
*Quiz 2	15%
Final exam	40%

**The dates for the quizzes will be announced one week prior.*

Syllabus

Week	Topic	Description
1	Time and Memory Complexity, Recursion, GCD, LCM, Fibonacci	This module introduces fundamental concepts of algorithm efficiency, focusing on **time and memory complexity**, including asymptotic notations and analysis. Students will learn **recursion**, a powerful problem-solving technique, and its application in solving complex problems efficiently. The topic also covers **GCD (Greatest Common Divisor)** and **LCM (Least Common Multiple)**, explaining their importance in computational problems and their efficient computation using Euclid's algorithm. Additionally, the module explores the **Fibonacci sequence**, showcasing both recursive and iterative approaches, emphasizing optimization techniques such as memoization.
2	Stack & Queue	This module delves into two fundamental linear data structures: **Stack** and **Queue**. Students will explore the **LIFO (Last In, First Out)** principle of stacks, covering operations like push, pop, and peek, along with their applications in scenarios such as expression evaluation, backtracking, and undo functionality. For queues, the focus will be on the **FIFO (First In, First Out)** principle, with operations like enqueue and dequeue, as well as their use in task scheduling, buffering, and breadth-first search. Variants such as **circular queues**, **priority queues**, and **double-ended queues (deques)** will also be introduced.

3	Linked List	This module focuses on **Linked Lists**, a dynamic data structure that allows efficient memory utilization. Students will learn about **singly linked lists**, **doubly linked lists**, and **circular linked lists**, understanding their structures, advantages, and limitations compared to arrays. Key operations such as **insertion**, **deletion**, **traversal**, and **searching** will be covered in detail. Techniques, like finding cycle, reversing a list will also be discussed.
4	Sorting and Searching	This module introduces essential techniques for organizing and retrieving data efficiently. Students will explore sorting algorithms, including bubble sort, selection sort, insertion sort, merge sort, and quick sort, focusing on their time and space complexities. The module also covers searching techniques such as linear search and binary search, emphasizing their implementation, performance analysis, and applications. Advanced topics like binary search on sorted arrays, searching in rotated arrays, and the importance of stability in sorting algorithms will be discussed.
5	Greedy Methods **(Quiz #1 will be conducted this week)**	This module explores the **greedy algorithmic paradigm**, which solves problems by making the locally optimal choice at each step with the hope of finding a global optimum. Students will learn the foundational principles of greedy methods, including problem identification and proof of correctness. Emphasis will be placed on understanding when greedy methods are applicable and analyzing their efficiency and limitations.
6	Dynamic Programming	This module focuses on **dynamic programming (DP)**, a powerful problem-solving approach used to solve problems with overlapping subproblems and optimal substructure properties. Students will learn the key principles of DP, such as **memoization** (top-down approach) and **tabulation** (bottom-up approach). The module covers classic DP problems, including **0/1 knapsack**, **longest common subsequence**, **longest increasing subsequence**. Emphasis will be placed on problem formulation, state transitions, and efficiency analysis, along with practical applications in optimization and real-world scenarios.
7	Binary Trees	This module introduces **binary trees**, a fundamental hierarchical data structure widely used in computer science. Students will learn about the structure and properties of binary trees, mainly including **binary search trees (BSTs)**. Key operations such as **insertion**, **deletion**, **traversal** (inorder, preorder, postorder, and level-order), and **searching** will be covered. Advanced topics like **tree height**, **depth**, and **recursive vs. iterative tree operations** will be introduced.
8	Introduction to Graphs	This module provides an introduction to **graphs**, a versatile data structure used to model relationships between entities. Students will learn about the basic components of a graph, including **vertices (nodes)** and **edges**, along with various graph types such as **directed**, **undirected**, **weighted**, and **unweighted graphs**. Key representations like **adjacency matrix** and **adjacency list** will be covered. The module emphasizes understanding graph properties, applications in real-world scenarios (e.g., social networks, maps, and communication networks), and basic operations such as **adding/removing edges and nodes**, **degree calculation**, and **connectivity checks**. Depth-first search (DFS) and breadth-first search (BFS) are excluded in this introduction.
9	Heaps and Union Find	This module introduces two fundamental data structures: **heaps** and **union-find**.

		1. **Heaps**: Students will learn about the **binary heap**, a complete binary tree used for efficient priority queue operations. The focus will be on understanding **min-heaps** and **max-heaps**, their structure, and key operations such as **insertion**, **deletion**, and **heapify**. Applications like **Heap Sort** and **priority scheduling** will also be introduced, showcasing the usefulness of heaps in real-world problems. 2. **Union-Find**: This section covers the **union-find (disjoint-set)** data structure, which efficiently handles dynamic connectivity queries. Students will learn about **union** and **find** operations, with an introduction to optimization techniques like **path compression** and **union by rank**. The module will highlight the applications of union-find in solving problems like **detecting cycles** and **Kruskal's algorithm** for finding minimum spanning trees.
10	Graphs Traversal Techniques – DFS **(Quiz #2 will be conducted this week)**	This module focuses on **Depth-First Search (DFS)**, a fundamental graph traversal algorithm. Students will learn how DFS explores a graph by starting at a source node and visiting as far as possible along each branch before backtracking. The algorithm will be studied in both **recursive** and **iterative** forms, with an emphasis on understanding its implementation and behavior in different types of graphs (directed, undirected, cyclic, and acyclic). Applications of DFS will be covered, including **topological sorting**, **cycle detection**, and **connected components** in graphs. The module will also explore the **space complexity** of DFS and how it can be optimized for large graphs using techniques such as **path compression** and **iteration over recursion**.
11	Graphs Traversal Techniques - BFS	This module covers **Breadth-First Search (BFS)**, a fundamental graph traversal algorithm that explores all nodes at the present depth level before moving on to nodes at the next depth level. Students will learn how BFS systematically visits all reachable nodes starting from a given source node, using a **queue** to manage the nodes to be explored. The algorithm will be implemented both **iteratively** and through **queue-based** structures. BFS is commonly used for tasks like **shortest path finding** in unweighted graphs, **level-order traversal** in trees, and **connected components** detection. The module will also highlight the time and space complexities of BFS, its applications in problems such as **finding the shortest path in a maze** or **broadcasting in networks**, and its comparison with DFS in different graph structures.
12	Graph Connectivity, Graph Algorithms	This module explores **graph connectivity** and introduces several important **graph algorithms**. Students will learn about different types of connectivity in graphs, including **strongly connected components** in directed graphs and **connected components** in undirected graphs. The module covers methods for identifying and analyzing graph connectivity, using algorithms such as **DFS**, **BFS**, and **Union-Find**. In addition to connectivity, students will be introduced to key **graph algorithms** for solving common graph problems. These include **Dijkstra's algorithm** for finding the shortest path in

		weighted graphs, **Kruskal's and Prim's algorithms**for minimum spanning trees, and **Bellman-Ford** for handling graphs with negative weights. The module emphasizes the real-world applications of these algorithms, including **network routing**, **social networks**, and **resource optimization**. The focus will be on understanding the efficiency and practical implementation of these algorithms, including their time and space complexities.