# Advanced Data Structures

## B-Tree, Splay, Priority Queue, Complexity, Searching, Sorting, Hashing, Algorithms, Implementation

In this specialist course we examine modern data structures from a mathematical viewpoint.

We explore how to correctly design modern data structures, how to implement the algorithms that interact with them, memory management, traversal, item searching and sorting, merging and splitting, and lots more. We use both imperative and functional programming styles. We also cover the wide range of proven data structure layouts already used in industry - each with its own capabilities, challenges and recommended usage domains.

We are particularly interested in where data structures and mathematics meet. For example, this course explores order theory and complexity theory and brings the richness of certain mathematical structures to common programming libraries (e.g. mathematical set has extra useful features that are not present in common implementations of sets in various runtimes). We also pay attention to correct usage of terminology to avoid misnaming – e.g. we see in C++ STL use of the term "vector" for something that is not a mathematical vector (an STL vector is a mathematical sequence [dynamic array]) - it is now too late to correct that misnaming.

### Target Audience

This course is aimed mathematicians and developers who wish to gain a richer understanding of modern approaches to data structures.

### Prerequisites

Attendees need a good background in mathematics and programming.

### *Contents of One-Day Training Course*

#### Goals of Data Structures

Designing data structure the right way
Important considerations (future evolution, performance, compression, ..)
Accessibility (random vs. sequential)
Visibility (public vs. internal functionality)
Complexity and Big O notation

#### Common Data Structures

Review of common data structures from a mathematical viewpoint
Mathematical sequence = array
Mathematical set = set
Mathematical morphism = map (sometimes called dictionary)
How best to implement

#### Additional Data Structures

Linked list (how to get same performance from singly vs. doubly linked list)
FIFO queue (stack)
LIFO queue
Does each data structure need to be independently created, or can it be layered on top of another (adaptors)?

#### B-Tree

B-Tree is the most important data structure for storage architectures
Performance characteristics of B-Trees
Variations of B-Tree layout

#### Splay Tree

Introduction to binary search tree
A splay tree is a binary search tree
Recently used items quick to re-access
The splaying operation

#### Priority Queue

Priority for an item
Sorting on that

#### Hashing

Hashing plays a particular importance in many searching and sorting algorithms
Examine from first principles how efficient hashing works

#### Common Operations

Searching
Sorting
Merging
Splitting
Parallel access

#### Data Structures For Functional Programming

There are particular considerations to designing data structures for use with functional programming languages
Immutability and versioning

#### Algorithms

Structure independent algorithms
Applying algorithms in a uniform way
Use of iterators to traverse structures

#### Memory

For data structures that grow and contract over time, need good memory management
Memory management and allocators
Slab allocators

#### Project

Creating a custom graph engine to store and interact with large-scale graphs efficiently