# DS and Algorithm Master Reference

[DS and Algorithm Master Reference 1](#_Toc14883)

[Commonly used words 2](#_Toc4971)

[Primitive Data Structures 3](#_Toc29080)

[Integer 3](#_Toc31221)

[Float 3](#_Toc5470)

[Character 3](#_Toc3606)

[Boolean 3](#_Toc18886)

[Linear Data Structures 3](#_Toc20760)

[Arrays 3](#_Toc18478)

[Linked Lists 3](#_Toc18233)

[Singly Linked Lists 3](#_Toc15076)

[Doubly Linked Lists 3](#_Toc29805)

[Circular Linked Lists 3](#_Toc5478)

[Stacks 3](#_Toc26668)

[Queues 3](#_Toc13173)

[Priority Queue 3](#_Toc6328)

[Circular Queue 4](#_Toc7625)

[Deque (Double-Ended Queue) 4](#_Toc4473)

[Non-Linear Data Structures 5](#_Toc2416)

[Trees 5](#_Toc21863)

[Binary Trees 5](#_Toc3869)

[Binary Search Trees 5](#_Toc7578)

[AVL Trees 5](#_Toc16881)

[Red-Black Trees 5](#_Toc7540)

[Heap (Min-Heap/Max-Heap) 5](#_Toc18818)

[B-Trees 5](#_Toc260)

[Tries 5](#_Toc303)

[Graphs 5](#_Toc16269)

[Directed Graphs 5](#_Toc28418)

[Undirected Graphs 5](#_Toc27492)

[Weighted Graphs 5](#_Toc13998)

[Unweighted Graphs 5](#_Toc12659)

[File Structures: 6](#_Toc4457)

[Sequential Files 6](#_Toc3624)

[Indexed Files 6](#_Toc22966)

[Direct Files 6](#_Toc10912)

[Hashing: 6](#_Toc29271)

[Hash Function 6](#_Toc11329)

[Hash Table 6](#_Toc18947)

[Hash Map 6](#_Toc21036)

[Handling collision 7](#_Toc2451)

[Resizing 9](#_Toc22311)

[Advanced Data Structures: 10](#_Toc10430)

[Segment Trees 10](#_Toc1009)

[Fenwick Trees (Binary Indexed Trees) 10](#_Toc17826)

[Suffix Trees 10](#_Toc32030)

[K-D Trees 10](#_Toc15788)

[Sparse Tables 10](#_Toc28910)

[Disjoint-Set (Union-Find) 10](#_Toc15001)

[Specialized Data Structures: 10](#_Toc11449)

[Bloom Filters 10](#_Toc15526)

[LRU Cache 10](#_Toc7913)

[Abstract Data Types (ADTs): 10](#_Toc13103)

[List ADT 10](#_Toc12700)

[Stack ADT 10](#_Toc32585)

[Queue ADT 10](#_Toc2904)

[Map ADT 11](#_Toc22550)

[Set ADT 11](#_Toc28293)

#### References

Coursera: <https://www.coursera.org/learn/algorithms-part1?action=enroll>

Github: https://github.com/BalajiBaskaran24/DotnetDev/tree/main/DSAndAlgoSample/DSAndAlgoReference

## Commonly used words

Trivial - Less significant or little value

## Recursion

Function calling itself. Base for dynamic programming. Parameterized recursion, functional recursion are some kind of recursion.

Sample in github (RecursionMethods in DSAndAlgoReference)

## Primitive Data Structures

### Integer

### Float

### Character

### Boolean

## Linear Data Structures

### Arrays

### Linked Lists

### Singly Linked Lists

### Doubly Linked Lists

### Circular Linked Lists

### Stacks

### Queues

### Priority Queue

### Circular Queue

### Deque (Double-Ended Queue)

## Non-Linear Data Structures

### Trees

### Binary Trees

### Binary Search Trees

### AVL Trees

### Red-Black Trees

### Heap (Min-Heap/Max-Heap)

### B-Trees

### Tries

### Graphs

### Directed Graphs

### Undirected Graphs

### Weighted Graphs

### Unweighted Graphs

## File Structures:

### Sequential Files

### Indexed Files

### Direct Files

## Hashing:

Hashing is the process of converting a large, possibly variable-sized amount of data into a small datum, usually a single integer that may serve as an index to an array (called a hash table). The values are used to index a fixed-size table called a hash table. In C# **Dictionary** is implemented using hashing technique

**C# classes**

**Hash Table:** Hashtable class. - Recommeded to use **Dictionary**

**Hash Map:** Dictionary<TKey, TValue> class.

**Hash Set:** HashSet<T> class. - Can be thought of as a Dictionary<TKey,TValue> collection without values. Set is a collection without duplicate elements

### Hash Function

A hash function takes a key and returns an integer, known as the hash code. A good hash function distributes keys uniformly across the hash table to avoid collisions as much as possible.

### Hash Table

A hash table is an array of fixed size, and it stores values. The index where a value is stored is determined by the hash code of its key, often modulo the size of the array.

### Hash Map

A hash map is a data structure that provides a way to store key-value pairs and allows you to efficiently insert, delete, and retrieve values based on the key. It's also known as a hash table or dictionary in various programming languages.

### Handling collision

A collision in hashing is when two or more keys are hashed to the same index in the hash table. This situation needs to be handled, or else the new key would overwrite the value that was already at that index in the hash table. Collisions can occur due to various reasons, such as limited size of the table, a poor hash function, etc.

#### Separate Chaining:

##### Linked List Chaining:

Each cell in the hash table contains a linked list of all the keys that hash to that index.

Code in github

##### Binary Search Tree Chaining:

A variation of chaining that uses a balanced binary search tree instead of a linked list.

#### Open Addressing:

Open addressing stores all entries directly in the array itself. When a collision occurs, the algorithm searches through the array to find the next open spot.

##### Linear Probing:

After a collision, check the next cell in the array, then the next, etc., until an empty cell is found.

##### Quadratic Probing:

Similar to linear probing but looks at cells that are a quadratic number of cells away from the original hash.

##### Double Hashing:

Uses a second hash function to decide how many cells to skip before checking again.

#### Rehashing:

When the table becomes too full, a common strategy is to create a new table of a larger size and rehash all existing keys.

#### Cuckoo Hashing:

Uses two or more hash functions. If a key is hashed to a place that's already occupied, the key that's already there is rehashed with its second hash function, making room for the new key.

#### Hopscotch Hashing:

Allows a new key to displace a key at an existing hash, as long as it doesn't move it too far from its original hash.

#### Robin Hood Hashing:

Works with open addressing and tries to ensure that each key is as close to its original hash as possible. It "steals" from rich cells (those with keys that are close to their original hashes) and gives to poor cells (those with keys far from their original hashes).

#### 2-Choice Hashing (or d-Choice Hashing):

Picks d random hash functions and places the new key in the bucket that is the least full among the d choices.

#### Coalesced Hashing:

Combines open addressing with a linked list, effectively merging cells together as they fill up.

#### Dynamic Perfect Hashing:

A two-level technique where the first level divides the keys into buckets, and the second level uses a perfect hash function for the keys in each bucket.

#### Bloom Filters (Probabilistic Hashing):

A probabilistic data structure that can tell you if a key is definitely not in the set or may be in the set. It's not a traditional collision resolution strategy but is related to hashing.

### Resizing

If the hash table becomes too full, it may need to be resized to maintain efficiency. This often involves creating a new, larger array and rehashing all existing keys.

## Advanced Data Structures:

### Segment Trees

### Fenwick Trees (Binary Indexed Trees)

### Suffix Trees

### K-D Trees

### Sparse Tables

### Disjoint-Set (Union-Find)

## Specialized Data Structures:

### Bloom Filters

### LRU Cache

## Abstract Data Types (ADTs):

### List ADT

### Stack ADT

### Queue ADT

### Map ADT

### Set ADT

## Dynamic Programming