

---

**Note:**

Please don't take effort to create pull requests for new algorithms or data structures. This is just a curiosity-driven personal hobby and was originally not intended to be a library. Feel free fork and modify to fit your need if that's what you are looking for. You can however open issues or fix bugs with pull requests, I would be happy to take a look when I get time

## Advanced Algorithms

Various important computer science algorithms generically implemented in C#.



Install by nuget

For beta releases on beta branch

```
1 Install-Package Advanced.Algorithms -Pre
```

For stable releases on stable branch

```
1 Install-Package Advanced.Algorithms
```

- API documentation

Supports

- .Net Standard 1.0 or above
- .Net Framework 4.0 or above

## Development environment

### Windows

- Visual Studio Code as IDE for .NET core
- Visual Studio 2017 as IDE for .NET framework/.NET core

### Mac OS

- Visual Studio Code as IDE for .NET core
- Visual Studio 2017 as IDE for Mono

---

## Linux

- Visual Studio Code as IDE for .NET core
- Mono develop as IDE for Mono

## Data structures

### List

- ☒ Array list (dynamic array) (implementation | tests)
- ☒ Skip list (implementation | tests)

### HashSets

- ☒ HashSet (using separate chaining optionally with open address linear probing) (implementation | tests)
- ☒ Ordered HashSet (implementation | tests)

### Dictionaries

- ☒ Dictionary (using separate chaining optionally with open address linear probing) (implementation | tests)
- ☒ Ordered Dictionary (implementation | tests)

### Stack

- ☒ Stack (using dynamic array and optionally using singly linked list) (implementation | tests)

### Queue

- ☒ Queue (using dynamic array and optionally using doubly linked list) (implementation | tests)
- ☒ Priority queue (implementation | tests)

### Linked list

- ☒ Singly linked list (implementation | tests)
- ☒ Doubly linked list (implementation | tests)
- ☒ Circular linked list (implementation | tests)

---

## Heap

- ☒ Binary heap (implementation | tests)
- ☒ d-ary heap (implementation | tests)
- ☒ Binomial heap (implementation | tests)
- ☒ Fibonacci heap (implementation | tests)
- ☒ Pairing heap (implementation | tests)

Note: It is observed that among the implementations here, in practice, with the exclusion of UpdateKey (decrement/increment) operation, regular Binary Heap & d-ary Heap outperforms other in theory superiors. Likely because it doesn't have pointer juggling overhead and heap arrays are faster!

## Tree

- ☒ Tree (implementation | tests)
- ☒ Binary tree (implementation | tests)

## Binary search trees

- ☒ Binary search tree (implementation | tests)
- ☒ AVL tree (implementation | tests)
- ☒ Red black tree (implementation | tests)
- ☒ Splay tree (implementation | tests)
- ☒ Treap tree (implementation | tests)

## B trees (database trees)

- ☒ B-tree (implementation | tests)
- ☒ B+ tree (implementation | tests)

## Queryable trees

- ☒ Segment tree (implementation | tests)
- ☒ Binary indexed tree (Fenwick tree) (implementation | tests)
- ☒ Multi-dimensional interval tree (implementation | tests) using nested red-black tree
- ☒ Multi-dimensional k-d tree (implementation | tests) for range and nearest neighbour queries
- ☒ Multi-dimensional range tree (implementation | tests) using nested red-black tree
- ☒ R-tree (implementation | tests)
- ☒ Quadtree (implementation | tests)

---

TODO: Support multi-dimensional segment tree & binary indexed tree.

### **Lookup trees**

- ☒ Prefix tree (Trie) (implementation | tests)
- ☒ Suffix tree (implementation | tests)
- ☒ Ternary search tree (implementation | tests)

TODO: implement trie compression.

### **Set**

- ☒ Disjoint set (implementation | tests)
- ☒ Sparse set (implementation | tests)
- ☒ Bloom filter (implementation | tests)

### **Graph**

#### **Adjacency list**

- ☒ Graph (implementation | tests)
- ☒ Weighted Graph (implementation | tests)
- ☒ DiGraph (implementation | tests)
- ☒ Weighted DiGraph (implementation | tests)

#### **Adjacency matrix**

- ☒ Graph (implementation | tests)
- ☒ Weighted Graph (implementation | tests)
- ☒ DiGraph (implementation | tests)
- ☒ Weighted DiGraph (implementation | tests)

### **Algorithms**

#### **Graph algorithms**

##### **Articulation points**

- ☒ Tarjan's articulation points finder (implementation | tests)

---

## Bridges

- ☒ Tarjan's bridge finder (implementation | tests)

## Connectivity

- ☒ Kosaraju's strongly connected component finder (implementation | tests)
- ☒ Tarjan's strongly connected component finder (implementation | tests)
- ☒ Tarjan's bi-connected graph tester (implementation | tests)

## Coloring

- ☒ M-coloring (implementation | tests)

## Cover

- ☒ Min vertex cover (implementation | tests)

## Maximum flow

- ☒ Ford-Fulkerson algorithm (implementation | tests)
- ☒ Edmonds Karp's improvement (implementation | tests) on Ford-Fulkerson algorithm
- ☒ Push relabel algorithm (implementation | tests)

## Shortest path

- ☒ Bellman-Ford algorithm (implementation | tests)
- ☒ Dijkstra's algorithm (implementation | tests) using Fibonacci heap.
- ☒ Floyd-Warshall algorithm (implementation | tests)
- ☒ Johnson's algorithm (implementation | tests)
- ☒ Travelling salesman path (implementation | tests)
- ☒ A\* search algorithm (implementation | tests) using Fibonacci heap.

## Matching

- ☒ Max bipartite matching (implementation | tests) using Edmonds Karp's improved Ford Fulkerson max flow algorithm
- ☒ Max bipartite matching (implementation | tests) using Hopcroft Karp algorithm

---

## **Cut**

- ☒ Minimum cut (implementation | tests) using Edmonds Karp's improved Ford Fulkerson max flow algorithm

## **Cycle**

- ☒ Cycle detection (implementation | tests)

## **Search**

- ☒ Depth first (implementation | tests)
- ☒ Breadth first (implementation | tests)
- ☒ Bi-directional (implementation | tests)

## **Topological sort**

- ☒ Depth first method (implementation | tests)
- ☒ Kahn's algorithm (implementation | tests)

## **Minimum spanning tree**

- ☒ Kruskal's algorithm (implementation | tests) using merge sort and disjoint set
- ☒ Prim's algorithm (implementation | tests)

## **String**

- ☒ Manacher's algorithm for linear time longest palindrome (implementation | tests)

## **Pattern matching**

- ☒ Rabin-Karp string search (implementation | tests)
- ☒ Knuth–Morris–Pratt (KMP) string search (implementation | tests)
- ☒ Z algorithm for string search (implementation | tests)

## **Compression**

- ☒ Huffman coding (implementation | tests)

---

## Sorting and searching

- ☒ Binary search (implementation | tests)
- ☒ Quick select for kth smallest/largest in unordered collection using median of medians (implementation | tests)
- ☒ Majority element using Boyer-Moore voting algorithm (implementation | tests)

## Sorting algorithms

- ☒ Bubble sort (implementation | tests)
- ☒ Insertion sort (implementation | tests)
- ☒ Selection sort (implementation | tests)
- ☒ Shell sort (implementation | tests)
- ☒ Tree sort (implementation | tests)
- ☒ Quick sort (implementation | tests)
- ☒ Heap sort (implementation | tests)
- ☒ Merge sort (implementation | tests)
- ☒ Bucket sort (implementation | tests)
- ☒ Radix sort (implementation | tests)
- ☒ Counting sort (implementation | tests)

Note: On a decent desktop, in given implementations here for +ive random input integers, the clear winner is counting sort (~0.1 seconds to sort 1 million integers) followed by Radix Sort (~0.4 seconds). Merge Sort, Heap Sort, Quick Sort & Bucket Sort are all winners for +ive & -ive random integer inputs. Tree sort has pointer juggling overhead on backing Red-Black Tree, so performs 8 times slower than Merge Sort in practice. Bubble Sort, Insertion Sort, Selection Sort & Shell Sort are among the worst for random input as observed from results.

## Combinatorics

- ☒ Permutations (implementation | tests)
- ☒ Combinations (implementation | tests)
- ☒ Subsets (implementation | tests)

## Distributed Systems

- ☒ Circular queue (ring buffer) (implementation | tests)
- ☒ Consistent hash (implementation | tests)

- 
- ☒ LRU cache (implementation | tests)
  - ☒ Asynchronous producer-consumer queue (implementation | tests)

### **Numerical methods**

- ☒ Check primality (implementation | tests)
- ☒ Generate primes using sieve of Eratosthenes (implementation | tests)
- ☒ Fast exponentiation (implementation | tests)

### **Geometry (in 2D)**

- ☒ Convex hull using gift wrapping algorithm (implementation | tests)
- ☒ Line intersection (implementation | tests)
- ☒ Closest point pair (implementation | tests)
- ☒ Check if given point inside polygon (implementation | tests)
- ☒ Rectangle intersection (implementation | tests)
- ☒ Point rotation (implementation | tests)
- ☒ Line intersections with Bentley-Ottmann sweep line algorithm using red-black tree and binary minimum heap (implementation | tests)

### **Bit manipulation**

- ☒ Base conversion (implementation | tests)
- ☒ Calculate logarithm (base 2 & 10) (implementation | tests)
- ☒ GCD (implementation | tests)