



Welcome to the Swift Algorithm Club!

Here you'll find implementations of popular algorithms and data structures in everyone's favorite new language Swift, with detailed explanations of how they work.

If you're a computer science student who needs to learn this stuff for exams – or if you're a self-taught programmer who wants to brush up on the theory behind your craft – you've come to the right place!

The goal of this project is to **explain how algorithms work**. The focus is on clarity and readability of the code, not on making a reusable library that you can drop into your own projects. That said, most of the code should be ready for production use but you may need to tweak it to fit into your own codebase.

Code is compatible with **Xcode 10** and **Swift 4.2**. We'll keep this updated with the latest version of Swift. If you're interested in a GitHub pages version of the repo, check out this.

:heart_eyes: **Suggestions and contributions are welcome!** :heart_eyes:

Important links

What are algorithms and data structures? Pancakes!

Why learn algorithms? Worried this isn't your cup of tea? Then read this.

Big-O notation. We often say things like, "This algorithm is **$O(n)$** ." If you don't know what that means, read this first.

Algorithm design techniques. How do you create your own algorithms?

How to contribute. Report an issue to leave feedback, or submit a pull request.

Where to start?

If you're new to algorithms and data structures, here are a few good ones to start out with:

- Stack
- Queue
- Insertion Sort
- Binary Search and Binary Search Tree
- Merge Sort
- Boyer-Moore string search

The algorithms

Searching

- Linear Search. Find an element in an array.
- Binary Search. Quickly find elements in a sorted array.
- Count Occurrences. Count how often a value appears in an array.
- Select Minimum / Maximum. Find the minimum/maximum value in an array.
- k-th Largest Element. Find the k -th largest element in an array, such as the median.
- Selection Sampling. Randomly choose a bunch of items from a collection.
- Union-Find. Keeps track of disjoint sets and lets you quickly merge them.

String Search

- Brute-Force String Search. A naive method.
- Boyer-Moore. A fast method to search for substrings. It skips ahead based on a look-up table, to avoid looking at every character in the text.
- Knuth-Morris-Pratt. A linear-time string algorithm that returns indexes of all occurrences of a given pattern.
- Rabin-Karp. Faster search by using hashing.
- Longest Common Subsequence. Find the longest sequence of characters that appear in the same order in both strings.
- Z-Algorithm. Finds all instances of a pattern in a String, and returns the indexes of where the pattern starts within the String.

Sorting

It's fun to see how sorting algorithms work, but in practice you'll almost never have to provide your own sorting routines. Swift's own `sort()` is more than up to the job. But if you're curious, read on...

Basic sorts:

- Insertion Sort
- Selection Sort
- Shell Sort

Fast sorts:

- Quicksort
- Merge Sort
- Heap Sort

Hybrid sorts:

- Introsort

Special-purpose sorts:

- Counting Sort
- Radix Sort
- Topological Sort

Bad sorting algorithms (don't use these!):

-
- Bubble Sort
 - Slow Sort

Compression

- Run-Length Encoding (RLE). Store repeated values as a single byte and a count.
- Huffman Coding. Store more common elements using a smaller number of bits.

Miscellaneous

- Shuffle. Randomly rearranges the contents of an array.
- Comb Sort. An improve upon the Bubble Sort algorithm.
- Convex Hull.
- Miller-Rabin Primality Test. Is the number a prime number?
- MinimumCoinChange. A showcase for dynamic programming.
- Genetic. A simple example on how to slowly mutate a value to its ideal form, in the context of biological evolution.
- Myers Difference Algorithm. Finding the longest common subsequence of two sequences. ### Mathematics
- Greatest Common Divisor (GCD). Special bonus: the least common multiple.
- Permutations and Combinations. Get your combinatorics on!
- Shunting Yard Algorithm. Convert infix expressions to postfix.
- Karatsuba Multiplication. Another take on elementary multiplication.
- Haversine Distance. Calculating the distance between 2 points from a sphere.
- Strassen's Multiplication Matrix. Efficient way to handle matrix multiplication.
- CounterClockWise. Determining the area of a simple polygon.

Machine learning

- k-Means Clustering. Unsupervised classifier that partitions data into k clusters.
- k-Nearest Neighbors
- Linear Regression. A technique for creating a model of the relationship between two (or more) variable quantities.

-
- Logistic Regression
 - Neural Networks
 - PageRank
 - Naive Bayes Classifier
 - Simulated annealing. Probabilistic technique for approximating the global maxima in a (often discrete) large search space.

Data structures

The choice of data structure for a particular task depends on a few things.

First, there is the shape of your data and the kinds of operations that you'll need to perform on it. If you want to look up objects by a key you need some kind of dictionary; if your data is hierarchical in nature you want a tree structure of some sort; if your data is sequential you want a stack or queue.

Second, it matters what particular operations you'll be performing most, as certain data structures are optimized for certain actions. For example, if you often need to find the most important object in a collection, then a heap or priority queue is more optimal than a plain array.

Most of the time using just the built-in `Array`, `Dictionary`, and `Set` types is sufficient, but sometimes you may want something more fancy...

Variations on arrays

- `Array2D`. A two-dimensional array with fixed dimensions. Useful for board games.
- `Bit Set`. A fixed-size sequence of n bits.
- `Fixed Size Array`. When you know beforehand how large your data will be, it might be more efficient to use an old-fashioned array with a fixed size.
- `Ordered Array`. An array that is always sorted.
- `Rootish Array Stack`. A space and time efficient variation on Swift arrays.

Queues

- `Stack`. Last-in, first-out!
- `Queue`. First-in, first-out!
- `Deque`. A double-ended queue.
- `Priority Queue`. A queue where the most important element is always at the front.
- `Ring Buffer`. Also known as a circular buffer. An array of a certain size that conceptually wraps around back to the beginning.

Lists

- **Linked List.** A sequence of data items connected through links. Covers both singly and doubly linked lists.
- **Skip-List.** Skip List is a probabilistic data-structure with same logarithmic time bound and efficiency as AVL/ or Red-Black tree and provides a clever compromise to efficiently support search and update operations.

Trees

- **Tree.** A general-purpose tree structure.
- **Binary Tree.** A tree where each node has at most two children.
- **Binary Search Tree (BST).** A binary tree that orders its nodes in a way that allows for fast queries.
- **Red-Black Tree.** A self balancing binary search tree.
- **Splay Tree.** A self balancing binary search tree that enables fast retrieval of recently updated elements.
- **Threaded Binary Tree.** A binary tree that maintains a few extra variables for cheap and fast in-order traversals.
- **Segment Tree.** Can quickly compute a function over a portion of an array.
 - Lazy Propagation
- **kd-Tree**
- **Sparse Table.** Another take on quickly computing a function over a portion of an array, but this time we'll make it even quicker!.
- **Heap.** A binary tree stored in an array, so it doesn't use pointers. Makes a great priority queue.
- **Fibonacci Heap**
- **Trie.** A special type of tree used to store associative data structures.
- **B-Tree.** A self-balancing search tree, in which nodes can have more than two children.
- **QuadTree.** A tree with 4 children.
- **Octree.** A tree with 8 children.

Hashing

- **Hash Table.** Allows you to store and retrieve objects by a key. This is how the dictionary type is usually implemented.
- **Hash Functions**

Sets

- Bloom Filter. A constant-memory data structure that probabilistically tests whether an element is in a set.
- Hash Set. A set implemented using a hash table.
- Multiset. A set where the number of times an element is added matters. (Also known as a bag.)
- Ordered Set. A set where the order of items matters.

Graphs

- Graph
- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Shortest Path on an unweighted tree
- Single-Source Shortest Paths
- Minimum Spanning Tree on an unweighted tree
- Minimum Spanning Tree
- All-Pairs Shortest Paths
- Dijkstra's shortest path algorithm
- A-Star

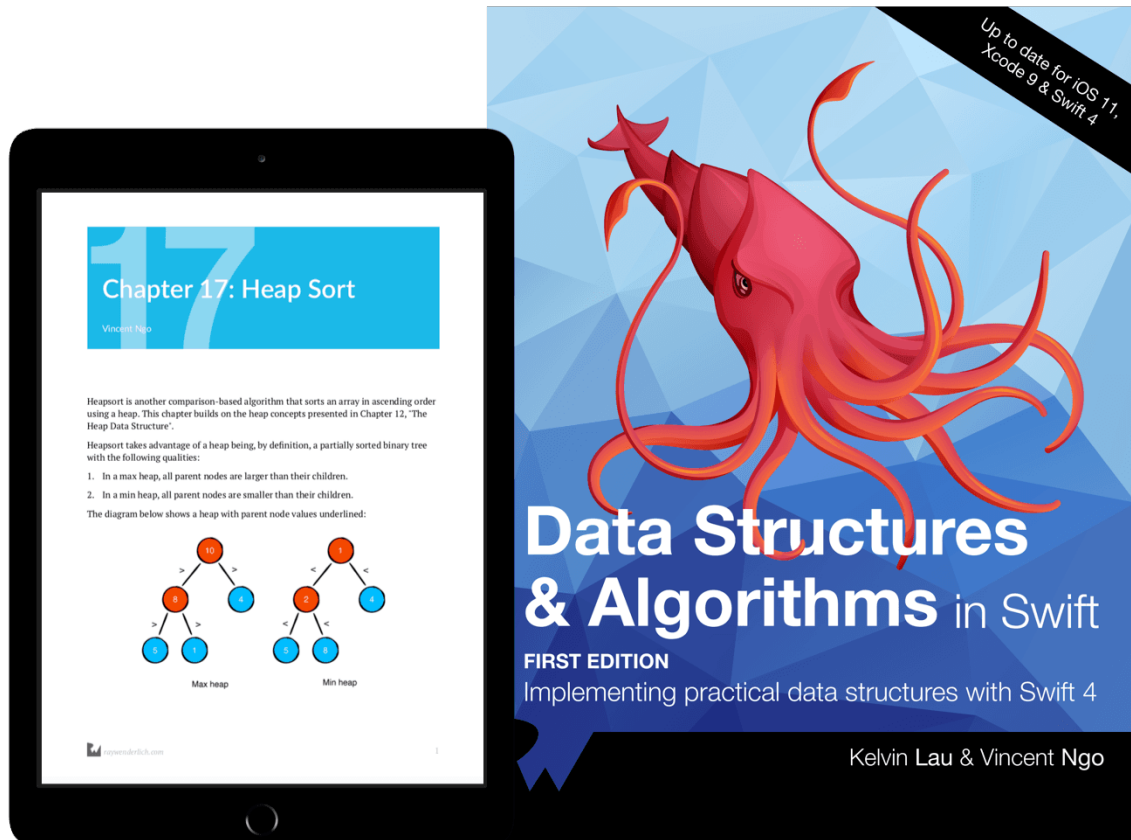
Puzzles

A lot of software developer interview questions consist of algorithmic puzzles. Here is a small selection of fun ones. For more puzzles (with answers), see [here](#) and [here](#).

- Two-Sum Problem
- Three-Sum/Four-Sum Problem
- Fizz Buzz
- Monty Hall Problem
- Finding Palindromes
- Dining Philosophers
- Egg Drop Problem
- Encoding and Decoding Binary Tree
- Closest Pair

Learn more!

Like what you see? Check out *Data Structures & Algorithms in Swift*, the official book by the Swift Algorithm Club team!



You'll start with the fundamental structures of linked lists, queues and stacks, and see how to implement them in a highly Swift-like way. Move on to working with various types of trees, including general purpose trees, binary trees, AVL trees, binary search trees, and tries.

Go beyond bubble and insertion sort with better-performing algorithms, including mergesort, radix sort, heap sort, and quicksort. Learn how to construct directed, non-directed and weighted graphs to represent many real-world models, and traverse graphs and trees efficiently with breadth-first, depth-first, Dijkstra's and Prim's algorithms to solve problems such as finding the shortest path or lowest cost in a network.

By the end of this book, you'll have hands-on experience solving common issues with data structures and algorithms — and you'll be well on your way to developing your own efficient and useful implementations!

You can find the book on the raywenderlich.com store.

Credits

The Swift Algorithm Club was originally created by Matthijs Hollemans.

It is now maintained by Vincent Ngo, Kelvin Lau, and Richard Ash.

The Swift Algorithm Club is a collaborative effort from the most algorithmic members of the raywenderlich.com community. We're always looking for help - why not join the club? :]

License

All content is licensed under the terms of the MIT open source license.

By posting here, or by submitting any pull request through this forum, you agree that all content you submit or create, both code and text, is subject to this license. Razeware, LLC, and others will have all the rights described in the license regarding this content. The precise terms of this license may be found [here](#).

build unknown