# Data Structures

* A **data structure** is a particular way of organizing data in a computer so that it can be used effectively.
* For example, we can store a list of items having the same data-type using the *array* data structure.



# [**Which algorithms/data structures should I "recognize" and know by name?**](https://softwareengineering.stackexchange.com/questions/155639/which-algorithms-data-structures-should-i-recognize-and-know-by-name)

### **Data Structures**

* Machine Data Representation
  + Ones, Two's Complement, and Related Arithmetic
  + Words, Pointers, Floating Point
  + Bit Access, Shifting, and Manipulation
* Linked Lists
* Hash Tables (maps or dictionaries)
* Arrays
* Trees
* Stacks
* Queues
* Graphs
* Databases

### **Algorithms**

* Sorting:
  + Bubble Sort (to know why it's bad)
  + Insertion Sort
  + Merge Sort
  + Quick Sort
  + Radix style sorts, Counting Sort and Bucket Sort
  + Heap Sort
  + Bogo and Quantum Sort (=
* Searching:
  + Linear Search
  + Binary Search
  + Depth First Search
  + Breadth First Search
* String Manipulation
* Iteration
* Tree Traversal
* List Traversal
* Hashing Functions
* Concrete implementation of a Hash Table, Tree, List, Stack, Queue, Array, and Set or Collection
* Scheduling Algorithms
* File System Traversal and Manipulation (on the inode or equivalent level).

### **Design Patterns**

* Modularization
* Factory
* Builder
* Singleton
* Adapter
* Decorator
* Flyweight
* Observer
* Iterator
* State [Machine]
* Model View Controller
* Threading and Parallel Programming Patterns

### **Paradigms**

* Imperative
* Object Oriented
* Functional
* Declarative
* Static and Dynamic Programming
* Data Markup

### **Complexity Theory**

* Complexity Spaces
* Computability
* Regular, Context Free, and Universal Turing Machine complete Languages
* Regular Expressions
* Counting and Basic Combinatorics

## **Beyond**

To get into what you're asking about later in your question, if you are familiar with the above, you should be easily able to identify the appropriate pattern, algorithm, and data structure for a given scenario. However, you should recognize that there is often no best solution. Sometimes you may be required to pick the lesser of two evils or even simply choose between two equally viable solutions. Because of this, you need the general knowledge to be able to defend your choice against your peers.

Here are some tips for algorithms and data structures:

* Binary Search can only (and should) be used on **sorted data**.
* Radix style sorts are awesome, but only when you have finite classes of things being sorted.
* Trees are good for almost anything as are Hash Tables. The functionality of a Hash Table can be extrapolated and used to solve many problems at the cost of efficiency.
* Arrays can be used to back most higher level data structures. Sometimes a "data structure" is no more than some clever math for accessing locations in an array.
* The choice of language can be the difference between pulling your hair out over, or sailing through, a problem.
* The ASCII table and a 128 element array form an implicit hash table (=
* Regular expressions can solve a lot of problems, but they [can't be used to parse HTML](https://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags).
* Sometimes the data structure is just as important as the algorithm.