Algorithm ADS	Data Structure ADS
Stack ADS	Queue ADS
Hash table ADS	Bucket Array ADS
Capacity ADS	Open addressing schemes ADS
Separate Chaining ADS	Second-Choice Hashing ADS
Linear Probing ADS	Robin Hood Hashing ADS
Quadratic Probing ADS	Double hashing ADS
Cuckoo Hashing ADS	Tombstone ADS
Backtracking ADS	Degree of a polynomial ADS
Monotonic ADS	Time complexity ADS

A particular way of storing and organising data in a computer so that it can be used efficiently	A method or process followed to solve a problem
A collection of objects that are inserted and removed according to the first-in-first-out (FIFO) principle	A collection of elements that are inserted and removed according to the last-in-first-out (LIFO) principle
An array A of size N where each cell A is thought of as a bucket storing a collection of key-value pairs	Consists of a bucket array and a hash function
Store at most one entry in each bucket	The size of the hash table
Compute two hash functions and store the key value pair in the bucket containing the fewest items	Each bucket A[i] stores a list holding the entries (k,v) such that h(k)=i
Variation of linear probing. If, during probing with a new key, an existing key is found that is "closer to home" than the new key, then the existing key is displaced and replaced by the new key	Try to insert into A[i], then A[(i+1) mod N] and so on until we find and empty bucket
Choose a secondary hash function h' to choose a bucket if the initial hash function leads to a full bucket $A[(i+f(j)) \bmod N],\ \text{for } j = 0, 1, 2, \ldots,\ \text{where } f(j) = j\dot{h}'(k)$	This iteratively tries the buckets: $A[(i + f(j)) mod N], \text{for} j = 0, 1, 2, ..., \text{where} f(j) = j^2$
A marker left in a bucket after something has been deleted	There are two tables with two corresponding hash functions
The highest power in the polynomial	Build up the solution one step at a time, backtracking when unable to continue
Expressed in terms of the number of basic operations used by the algorithm when the input has a particular size	Always going in one direction (either increasing or decreasing)

Big-O ADS	Sum Rule ADS
Product Rule ADS	Big-Omega ADS
Theta ADS	Little-o ADS
Little-Omega ADS	Comparisons needed for a comparison based sorting algorithm ADS
Bucket Sort ADS	Radix sort ADS
Binary Search ADS	Tree ADS
Binary Search Tree ADS	Types of edges for DFS ADS
Kruskal's Algorithm ADS	Prim's Algorithm ADS

The $\mathcal{O}$ of the sum of two functions is the maximum of the $\mathcal{O}$ of the two	$\lvert f(x)\rvert \leqslant C \cdot \lvert g(x)\rvert$ whenever $x \geqslant k$
$\lvert f(x)\rvert \geqslant C \cdot \lvert g(x)\rvert$	The $\mathcal{O}$ of the product of two functions is the product of their $\mathcal{O}$
$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$	$f(x)$ is $\mathcal{O}(g(x))$ and $g(x)$ is $\mathcal{O}(f(x))$
For any comparison based sorting algorithm $\mathcal{A}$ and any $n \in \mathbb{N}$ large enough there exists an input of length n that requires $\mathcal{A}$ to perform $\Omega(n \log n)$ comparisons	$f = \omega(g) \quad \Leftrightarrow \quad g = o(f)$
Like bucket sort but keeps sorting by different levels	Puts elements with key i into the ith bucket, then empties one bucket after another
A connected graph without cycles	Look at the middle of the list, use that to determine which half it is in, recursively call on the sublists
Tree - Edges in the DFS-Forest **Back** - Join a vertex to an ancestor **Forward** - Not in the tree but join a vertex to its descendant **Cross** - All others	A tree in which no node as more than two children. All elements in the left subtree are smaller than v and all in the right are bigger
Let $U = \{u\}$ where u is some vertex chosen arbitrarily Let $A = \varnothing$ Until U contains all vertices: find the least-weight edge e that joins a vertex w not in U and add e to A and w to U	Sort edges by weight Let A=$\varnothing$ Consider the edges in increasing order of weight. For each edge e, add e to A unless it would create a cycle