



Algorithms

To tackle a problem:

- Break the problem down
- Think about which data is important, and consider relevant data structures
- Tackle each part of the problem step by step

Terminal: Indicates the beginning or end of a program 


Line: Shows the direction of the process; connects two blocks

Input/Output: Indicates the process of inputting or outputting data 

Process: Indicates any process, action, or function 

Decision: Indicates a decision point 

Sub Program: Indicates a subroutine 

Connector: Shows a jump from one point in the process flow to another 

Bubble Sort:

- Compare consecutive pairs of items, Swap the items if they are in the wrong order
- Continue completing passes until you complete a pass with no swaps
- Advantages: Good for short lists, easy to code, Space Complexity is Constant
- Disadvantages: Time Complexity is Bad for large lists

Insertion Sort:

- Build up a sorted sublist in the first part of the original list, one item at a time, while the remaining part of the list remains unsorted
- Each item is inserted into the correct position
- Advantages: Good for short lists, easy to code, better than Bubble, Space Complexity is Constant
- Disadvantages: Time Complexity is Bad for large lists

Merge Sort (Divide and Conquer example):

- Split the list in half into two sublists
- Recursively break the sublists in half until each sublist contains only one item
- Merge the sublists back together, **in order**, until the original list is recreated
- Advantages: Time Complexity is low (performance is high), same Time Complexity for all sizes of lists, good for large lists
- Disadvantages: Space Complexity is bad for large lists

Quick Sort:

- Select a pivot value
- Split the list into three parts, a sublist of all values below the pivot, then the pivot, and then a sublist of all values above the pivot
- Split the two sublists recursively, until all sublists contain a single item
- Join the lists together in the same order, the final list will be sorted
- Advantages: High performance, low Space Complexity, good for large lists
- Disadvantages: Recursive, needs lots of memory; if recursion unavailable, implementation is very difficult

Linear Search

- Starts at the first item, checks each item one by one
- Advantages: Simple to implement, list can be unsorted
- Disadvantages: Inefficient for large lists

Binary Search

- Calculate the midpoint position, check the item at that position
- If this item is what you are looking for, the search is complete
- If this item is smaller than what you are looking for, discard all items below and including the midpoint

Algorithms

- If this item is larger than what you are looking for, discard all items above and including the midpoint
- Take the remaining list of items, and repeat
- Advantages: Efficient for large lists, simple to implement (but not as simple as linear)
- Disadvantages: List must be sorted

Binary Search Tree

- Rooted tree with ordered nodes
- Start at the root node's value
- If this value is larger than the item you are looking for, move left
- If this value is smaller than the item you are looking for, move right
- Repeat
- Advantages: Efficient with a large, balanced tree, Generally simple to implement
- Disadvantages: Inefficient with a large, unbalanced tree, recursive, needs lots of memory

Dijkstras's shortest path algorithm: Finds the shortest path between two nodes in a weighted graph; Breadth-first search. Doesn't work for edges with a negative weight value. Time and Space complexities: $O(V^2)$ (V is the number of vertices in the graph)

A* Pathfinding Algorithm differs from Dijkstra's Shortest Path Algorithm in that it stops when the goal node is visited (seeks only the shortest path between the start node and the target node) and uses a heuristic. Best-first search.

Example Heuristics: Manhattan, Euclidian. Manhattan Distance: Equal to the sum of the distances in the x-direction and y-direction. When choosing a heuristic: it should not overestimate costs. The time taken to provide the heuristic must not cancel out any time savings in the process of pathfinding. Dijkstra's algorithm will always find the shortest path, the A* algorithm will not always find the best option, but an option that is good enough. The A* algorithm only stops when the goal vertex has been marked as visited.

Space Complexity: A measure of how much memory the algorithm takes as its input increases

Time Complexity: A measure of how much time it takes to complete an algorithm as its input increases.

Alternative to "Complexity": Efficiency. Often, programmers will need to choose between maximising time Complexity or maximising space Complexity.

Big O Notation / Landau's symbol: A mathematical notation that is used to express the time or space complexity of an algorithm.

Two Rules:

- Remove all terms except the one with the largest factor or exponent
- Remove any constants

$O(1)$; Constant; As the input size increases, the amount of time the algorithm takes to complete stays constant.
; Hashing algorithm

$O(\log(n))$; Logarithmic; As the input size increases, the amount of time the algorithm takes to complete increases in proportion to the logarithmic result of the input size; Binary Search

$O(n)$; Linear; As the input size increases, the amount of time the algorithm takes to complete increases linearly.
; Linear search.

$O(n^2)$; Polynomial; As the input size increases, the amount of time the algorithm takes to complete increases in proportion to the input size raised to a power; Bubble sort.

$O(2^n)$; Exponential; As the input size increases, the amount of time the algorithm takes to complete increases exponentially. If the input size increases by 1, the time taken doubles; Recursive functions with two calls.

When multiple functions are being performed on an input, find the complexities of each function individually. Apply the same Two Rules to find the overall Time Complexity.

No Loop: Constant; Halving Data Set: Logarithmic; FOR/WHILE Loop: Linear; Nested Loop: Polynomial; Recursive: Exponential