**Challenges** | Search and Sort

An overall approach for all might be to create separate module files (Python), or class files (Java) for each of the grouped challenges, and access these from a central file/method. Your solution[s] should be dynamic, in that they must be capable of taking any size matrix and use simple constructs that are common to all programming languages.

# Search

For each of the following challenges create a function/method that takes a one-dimensional array and the item you are trying to locate and returns the index of that value. If the item is not located the algorithm will output a message to the console and return a null value.

## Binary Vs Linear Search Animated Gifs

## Linear/sequential Search:

A simple search algorithm that iterates through each element in a list until it finds the target value or reaches the end of the list.

## Binary Search:

An efficient search algorithm **for sorted lists**, which repeatedly divides the search range in half.

## A quick explanation of DFS & BFS (Depth First Search & Breadth-First Search) | by Sebastian De Lima | Analytics Vidhya | Medium

## Depth-first Search (DFS):

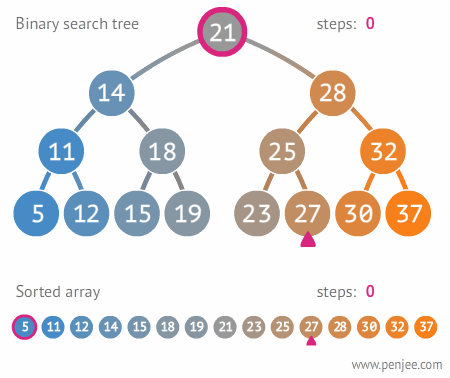
Explores **tree or graph** structures by going as deep as possible along each branch before backtracking.

## Breadth-first Search (BFS):

Explores **tree or graph** structures by visiting all vertices at the current depth before moving on to the next level.

## Binary Tree Search:

The search in a binary tree involves comparing the target value with the values in the nodes and navigating left or right based on the comparison until the target is found or the search reaches a leaf node.



# Sort

Comparison-based Sorting Algorithms:

### Bubble Sort:

Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. Simple but not efficient for large datasets.

### Selection Sort:

Divides the list into a sorted and an unsorted region, repeatedly selects the smallest element from the unsorted region, and swaps it with the first element of the unsorted region. Simple but not efficient for large datasets.

### Insertion Sort:

Builds the sorted list one element at a time, repeatedly taking the next element and inserting it into its proper place. Efficient for small datasets or partially sorted lists.

### Merge Sort:

Divides the list into smaller sub-lists, recursively sorts them, and then merges the sorted sub-lists to produce the final sorted list. Efficient for larger datasets and always has a consistent time complexity.

### Quick Sort:

Selects a "pivot" element and partitions the other elements into two sub-lists according to whether they are less than or greater than the pivot. Recursively sorts the sub-lists. Efficient for large datasets, but worst-case time complexity can be an issue.

## Non-Comparison-based Sorting Algorithm:

### Counting Sort:

Suitable for integers within a specific range. Creates a count array to store the count of each element and then modifies the count array to obtain the sorted list.

## Row-wise traversal (row by row)

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a number game  Description automatically generated | A screenshot of a number game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a number game  Description automatically generated |
| Top-left to bottom-right | Top-right to bottom-left | Bottom-right to top-left | Bottom-left to top-right |

## Column-wise traversal (column by column)

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a number game  Description automatically generated | A screenshot of a number game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a number game  Description automatically generated |
| Top-left to bottom-right | Top-right to bottom-left | Bottom-right to top-left | Bottom-left to top-right |

## Diagonal traversal (corner to corner)

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated |
| Top-left to bottom-right | Top-right to bottom-left | Bottom-right to top-left | Bottom-left to top-right |

## Horizontal snake traversal (row by row)

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a number game  Description automatically generated | A screenshot of a number game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated |
| Top-left to bottom-left | Top-right to bottom-right | Bottom-right to top-right | Bottom-left to top-left |

## Vertical snake traversal (column by column)

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a game  Description automatically generated | A screenshot of a number game  Description automatically generated | A screenshot of a number game  Description automatically generated | A screenshot of a game  Description automatically generated |
| Top-left to top-right | Top-left to top-right | Bottom-right to bottom-left | Bottom-left to top-left |

## Boundary traversal

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated |
| Top | Right | Bottom | Left |
| A screenshot of a game  Description automatically generated | A screenshot of a number game  Description automatically generated |  |  |
| Clockwise | Anticlockwise |  |  |

## Spiral traversal (column by column)

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated |
| Outside in clockwise | Outside in anticlockwise | Inside out clockwise | Inside out anticlockwise |

## Zigzag traversal

|  |  |  |  |
| --- | --- | --- | --- |
| A screenshot of a number game  Description automatically generated | A screenshot of a number game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a number game  Description automatically generated |
| Top-left to bottom-right, column first | Top-right to bottom-left, column first | Bottom-right to top-left, column first | Bottom-left to top-right, column first |
| A screenshot of a number game  Description automatically generated | A screenshot of a number game  Description automatically generated | A screenshot of a game  Description automatically generated | A screenshot of a game  Description automatically generated |
| Top-left to bottom-right,  row first | Top-right to bottom-left,  row first | Bottom-right to top-left,  row first | Bottom-left to top-right,  row first |

## Additional challenges (courtesy of ChatGPT)

* **Islands**: Given a matrix that represents the distribution of land and water in a given area, count the number of islands (connected land cells).
* **Adjacent submatrix**: Locate given element[s] in the matrix, and its adjacent elements.
* **Connected submatrix**: Locate given element[s] in the matrix, and elements deemed connected, for example odd or even.
* **Knight's Tour**: Given a chessboard matrix, show the path a given knight would take to visit every square on the board exactly once.
* **Knight's shortest path**: Extend the Knight's Tour problem by finding the shortest path from a starting position to an ending position on the chessboard.
* **Path**: Given matrix that represents a maze, find a path from the top-left to the bottom-right corner.
* **Unique paths**: Given matrix that represents a maze, find the number of unique paths from the top-left to the bottom-right corner, moving only right or down.
* **Magic square**: Determine if a given square matrix is a magic square, meaning the sum of each row, column, and diagonal is the same.
* **Multiplication**: Multiply two matrices.
* **Longest increasing path**: Given a matrix of numbers, find the longest path where a path is a sequence of cells with increasing values.
* **Sparse matrix compression**: Compress a sparse matrix (containing a sizeable number of zeros), converting it to a more memory-efficient representation.
* **Binary connectivity**: Given a matrix that represents a binary image, find the number of connected components of 1s.
* **Rotation**: Rotate a given matrix by 90 degrees clockwise or counterclockwise.