== Sliding Window

🡪 pattern used to perform an operation on a specific window size of a given array

or linked list

🡪 starts at 1st element, continues shifting right by one element while adjusting the

length of the window according to the problem being solved

🡪 commonly used in problems where:

- the problem is a linear data structure (arr, string, linked list)

- asked to find longest/shortest substring/subarr/etc.

A diagram of a window and a window

Description automatically generated

== Two Pointers/Iterators

🡪 problem in which two pointers iterate through structure in tandem until one OR both hit a certain condition

🡪 often useful when searching pairs in a sorted array or linked list

* For example, comparing each element of an array to other elements

🡪 two pointers needed in problems such as this because with just 1 pointer you would have to continuously loop back through the array for your answer, which is inefficient

🡪 identifying problems which need 2 pointer method:

* Features problems where you deal with sorted arrays/linked lists to find a set of elements which fulfill constraints
* Set of elements in array is pair/triplet/subarray

A diagram of a number system

Description automatically generated with medium confidence

== Fast and Slow Pointers (Hare & Tortoise Algorithm)

🡪 uses two pointers to move through an array/linked list at different speeds

🡪 very useful strategy when dealing with cyclic linked lists or arrays; by moving at different speeds, the algorithm uses the fact that the pointers are bound to meet. The fast pointer should catch the slow pointer once both are in a cyclic loop

🡪 when to use this versus the 2 pointer method:

* In cases such as a singly linked list in which you cannot move in a backwards direction
* A good example of the necessity of the fast and slow pattern is determining if a linked list is a palindrome

A diagram of a flowchart

Description automatically generated

== Merge Intervals

🡪 useful technique to deal with overlapping intervals; many interval problems require us to find overlapping or merging intervals

🡪 given 2 intervals ‘a’ and ‘b’ there are 6 different ways they can relate (listed below)

🡪 problems which require merge intervals pattern:

* Asked to produce a list of mutually exclusive intervals
* If the problem mentions the term overlapping intervals

A diagram of a number of rectangular objects

Description automatically generated

== Cyclic Sort

🡪 approach to deal with problems involving arrays containing numbers in a given range

🡪 iterates over array one number at a time, and if the current number you are iterating is not at the correct index, swap with correct index

🡪 placing the number in its correct index will create a O(n^2) complexity, hence why we use the cyclic sort pattern

🡪 problems in which we should use this pattern:

* Problems involving a sorted array with numbers in given range
* If the problem asks to find a missing/duplicate/smallest number in a sorted/rotated array

A screenshot of a computer

Description automatically generated

== In-Place Reversal of Linked List

🡪 many problems may ask to reverse links between a set of nodes in a linked list; the common constraint being that you need to do the replacement in-place, ie use the existing node objects without extra memory

🡪 pattern reverses one node at a time starting with a curr variable pointing to the head of the linked list, with a prev variable pointing to the previous processed node

* In a lock-step manner, reverse the current node by pointing it to previous before moving on to next node
* Update previous to always point to the previous node processed

A diagram of a flowchart

Description automatically generated

== Tree Algorithms

🡪 Tree BFS:

* Technique to traverse a tree using a queue to keep track of all the nodes at a given level before jumping to the next one
* Any problem involving traversal of a tree in level-by-level order can be solved efficiently using this approach
* Works by:
  + Pushing root node to the queue
  + Continuously iterating until queue is empty
  + For each iteration, remove node at head of queue and visit that node
  + After removing each node from queue, insert all its children into queue

🡪 Tree DFS:

* Technique to traverse a tree which can use recursion or iterative approach to keep track of all previous (parent) nodes
* Needed when the problem asks to traverse a tree in-order, postorder, or preorder, or if the problem requires searching for something where node is closer to a leaf
* Works by:
  + Starting at root of tree; if node is not a leaf:
  + Decide whether to process current node now (preorder) or between processing two children (inorder) or after processing both children (postorder)
  + Make two recursive calls for both children of current node to process them

== Two Heaps

🡪 in many problems, we are given a set of elements we can divide into two parts; we are usually interested in knowing the smallest element of one part and biggest element of another

🡪 pattern uses 2 heaps: min heap to find smallest element and max heap to find biggest element. Store first half of numbers in max heap, and second half of numbers in min heap. At any time, the median of the current list of numbers can be calculated from the top of element of both heaps.

🡪 useful in problems which include:

* Situations which need priority queue/scheduling
* Problems which require you to find smallest/largest/median elements of a set
* Can be useful in problems featuring binary tree struct

== Subsets

🡪 huge number of coding problems involve dealing with permutations or combinations of a given set of elements. The pattern subsets describes an efficient BFS approach to handle these problems

🡪 pattern goes like:

* Start with empty set
* Add first number to all the existing subsets to create new subsets
* Add second number to all existing subsets
* Add third number to all existing subsets
* And go on

A diagram of a diagram

Description automatically generated

== Modified Binary Search

🡪 when given a sorted array, linked list, or matrix, and we need to find a certain element, best algorithm to use is the binary search

🡪 pattern for an ascending order set:

* Find middle, start, and end
* If key is equal to number at middle index return
* If key is not equal to index middle:
  + Check if key < arr[middle] if it is then reduce search to end = middle – 1
  + If key > arr[middle] then reduce search to end = middle + 1

A diagram of a number system

Description automatically generated with medium confidence

== Top K elements

🡪 any problem which asks us to find top/smallest/most frequent K elements can use this pattern

🡪 best structure to keep track of k elements is a heap:

* Insert k elements into min-heap or max-heap based on problem
* Iterate through remaining numbers and if find one larger than what is in heap, remove that number and insert larger one

A screenshot of a math test

Description automatically generated

== K-way Merge

🡪 helps to solve problems which involve a set of sorted arrays

🡪 when given k sorted arrays, can use heap to perform a sorted traversal of elements of all arrays.

* Insert first element of each array in min heap
* Take out smallest (top) element from heap and add to merged list
* After removing smallest element from heap, insert next element of same list into heap
* Repeat 2nd and 3rd steps to populate merged list in sorted order

🡪 useful when problem features sorted arrays, matrix or asks to merge sorted lists or find smallest element in a sorted list

A screenshot of a computer game

Description automatically generated

== Topological Sort

🡪 used to find a linear ordering of elements which have dependencies on each other

🡪 pattern works like:

* Initialization
  + Store graph in adjacency lists using HasMap
  + Find all sources using HashMap to keep count of in-degrees
* Build graph from input and populate in-degrees hashmap
* Find all sources
  + All vertices with 0 in-degrees are sources and stored in queue
* Sort
  + for each source:
    - add to sorted list
    - get all children from graph
    - decrement in-degree of each child by 1
    - if child’s in-degree becomes 0, add to sources queue
    - repeat until source queue is empty

🡪 used in problems which have:

* graphs with no directed cycles
* asked to update all objects in sorted order
* if we have class of objects which follow a particular order

(diagram on next page)

A screenshot of a computer program

Description automatically generated