Dijkstra – This is mainly used to find the shortest path.

Adv: Efficient when we have weighted edges

DisAdv: We need weights, doesn’t work for negative weights and works for single source, single destination

Heapq – This is mainly used for priority queues. For it to have a heap property ie, min heap, it will need to have a root node and all left children needs to be smaller and right children larger

Adv: Speed, poping items is almost constant time (n Log n), finding min/max is constant

DisAdv: Need to maintain a heap property

DFS – Traversing or searching tree or graph data structures, selecting a root node and going as far as possible for that node.

Applications:

* Topological sorting
* DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree
* Detecting a cycle in a graph

BFT - Traversing or searching tree or graph data structures, and going level by level. Explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Applications:

* Shortest Path and Minimum Spanning Tree for unweighted graph
* Peer to Peer Networks. In Peer to Peer Networks like BitTorrent, Breadth First Search is used to find all neighbor nodes
* Facebook to find friends
* Crawlers in Search Engines: Crawlers build index using Breadth First. The idea is to start from source page and follow all links from source and keep doing same. Depth First Traversal can also be used for crawlers, but the advantage with Breadth First Traversal is, depth or levels of the built tree can be limited.
* GPS Navigation systems: Breadth First Search is used to find all neighboring locations

Merge\_sort – Divides itself until only one element is left and then forms back up in order.

Adv: It can be applied to files of any size.

DisAdv: Merge sort is less efficient than other sort; Merge Sort requires more space than other sort.

Rabin\_karp - is a string-searching algorithm. For text of length n and p patterns of combined length m, its average and best case running time is O(n+m) in space O(p), but its worst-case time is O(nm). A practical application of the algorithm is detecting plagiarism.

Adv: Rabin-Karp is a great algorithm for one simple reason – it can be used to match against multiple patterns. This makes it perfect to detect plagiarism even for larger phrases.

DisAdv:

* There are lots of string matching algorithms that are faster than O(n+m)
* It’s practically as slow as brute force matching and it requires additional space

BST - Binary search tree

Linked\_lists

Arrays