

Overview of important Algorithms

- Searching

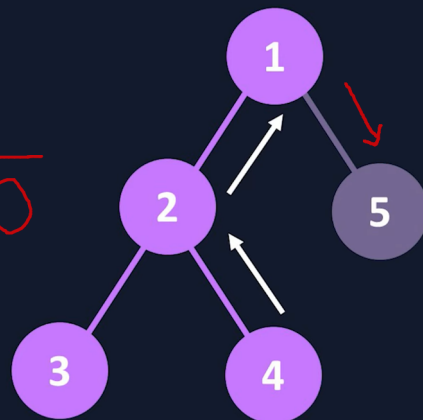
- Binary Search

- Depth First Search for trees and graphs

- start from the top of a tree and go as deep as possible along the same branch
 - once you are at the bottom then go to nearest unvisited node usually a sibling of the deepest node
 - this process is called **Backtracking**
 - used to solve a maze
 - $O(\text{number of nodes} + \text{number of branches})$

DFS: Visualized

visited = [1, 2, 3, 4]. 5



- Breadth First Search for trees and graphs

- you don't go to deepest point like DFS
 - instead you make sure that the sibling node has been visited
 - once you are on a node look at its children and add them to a queue and then you visit the node in the queue and add them to visited array and remove them from sibling queue
 - if the node in the queues has more children then add them to queue when marking it visited
 - used in chess
 - $O(\text{number of nodes} + \text{number of branches})$

- Sorting

- **Insertion Sort**
 - compares the n th element with $(n+1)$ th element and swaps them if n th element is larger
 - best case $O(n)$ if everything is already sorted
 - worst case $O(n^2)$ when nothing is sorted beforehand
- **Merge Sort**
 - divide and conquer and conquer by divide and conquer and so on
 - recursion
 - splits array in half till we have pairs of 2
 - then all pairs of 2 are sorted and then 2 pairs of 2 are merged and sorted till the array is completely merged back again
 - best and worst case are same $O(n \log n)$
- **Quick Sort**
 - recursive like merge sort so divides and conquers
 - we choose a pivot element of the array which is closest to the median of the array elements
 - then we split the lists into 2 such that one list has elements less than the pivot element and one where all elements are greater than the pivot element
 - we repeat the same on these 2 lists
 - we move the pivot element to the end of the list
 - we place 2 pointers one on the 0th index and the 2nd on the 2nd last element and compare the two if the 0th one is larger we swap
 - keep doing it till the 2 pointers meet
 - when they meet replace that element with the last one
 - we now have 2 lists like we wanted and we can do the same thing on them individually
 - best case $O(n \log n)$
 - worst case $O(n^2)$
 - still can be 2 to 3 times faster than merge sort by reducing the chances of worst case
 - needs less memory $O(\log n)$ than merge sort $O(n)$
- **Greedy Algorithm**
 - It makes the best possible decision at every local step
 - when not to be greedy
 - not meant for efficiency

- when to be greedy
 - when you don't want to find the most efficient way out of millions of permutations then greedy might be a good enough solution
 - when optimal solution not possible and brute force is not acceptable become greedy