Table of Contents

[Algo Mind maps 4](#_Toc144301291)

[Techniques 4](#_Toc144301292)

[Data Structure 5](#_Toc144301293)

[All Algo 6](#_Toc144301294)

[Algorithm Types and Definitions 7](#_Toc144301295)

[Elementary data structure 7](#_Toc144301296)

[Stack and queues 7](#_Toc144301297)

[trees 7](#_Toc144301298)

[heaps and heapsort 7](#_Toc144301299)

[sets and disjoints set union 7](#_Toc144301300)

[graphs 7](#_Toc144301301)

[hashing 7](#_Toc144301302)

[Divide and Conquer 7](#_Toc144301303)

[binary search 8](#_Toc144301304)

[finding the maximum and minimum 8](#_Toc144301305)

[mergesort 8](#_Toc144301306)

[quicksort 8](#_Toc144301307)

[selection 8](#_Toc144301308)

[strsassen's matrix multiplication 8](#_Toc144301309)

[The greedy method 8](#_Toc144301310)

[the general method 8](#_Toc144301311)

[optimal storage on tapes 8](#_Toc144301312)

[knapsack problem 8](#_Toc144301313)

[job sequencing with deadlines 8](#_Toc144301314)

[optimal merge patterns 8](#_Toc144301315)

[minimum spanning trees 8](#_Toc144301316)

[single source shortest paths 8](#_Toc144301317)

[Dynamic programming 9](#_Toc144301318)

[the general method 9](#_Toc144301319)

[multistage graphs 9](#_Toc144301320)

[all pairs shortest paths 9](#_Toc144301321)

[optimal binary search trees 9](#_Toc144301322)

[0/1 knapsack 9](#_Toc144301323)

[reliability design 9](#_Toc144301324)

[the traveling salesperson problem 9](#_Toc144301325)

[flow shop scheduling 9](#_Toc144301326)

[Basic search and traversal techniques 9](#_Toc144301327)

[Backtracking 9](#_Toc144301328)

[the general method 10](#_Toc144301329)

[the 8-queend problem 10](#_Toc144301330)

[sum of subsets 10](#_Toc144301331)

[graphs coloring 10](#_Toc144301332)

[hamiltonian cycles 10](#_Toc144301333)

[knapsack problem 10](#_Toc144301334)

[Branch and Bound 10](#_Toc144301335)

[the method 10](#_Toc144301336)

[0/1 knapsack problem 11](#_Toc144301337)

[traveling salesperson 11](#_Toc144301338)

[efficiency considerations 11](#_Toc144301339)

[Algebraic simplification and transformations 11](#_Toc144301340)

[the general method 11](#_Toc144301341)

[evaluation and interpolation 11](#_Toc144301342)

[the fast fourier transform 11](#_Toc144301343)

[modular arithmetic 11](#_Toc144301344)

[even faster evaluation and interpolation 11](#_Toc144301345)

[Lower bound theory 11](#_Toc144301346)

[np-hard land np-complete problems 11](#_Toc144301347)

[basic concepts 11](#_Toc144301348)

[cook's theorem 11](#_Toc144301349)

[np-hard graph problems 11](#_Toc144301350)

[np-hard scheduling problems 11](#_Toc144301351)

[np-hard code generation problems 11](#_Toc144301352)

[some simplified np-hard problems 11](#_Toc144301353)

[Approximation algorithms for np-hard problems 11](#_Toc144301354)

[introduction 12](#_Toc144301355)

[absolute approximations 12](#_Toc144301356)

[e-approximations 12](#_Toc144301357)

[polynomial time approximation schemes 12](#_Toc144301358)

[fully polynomical time approximation schemes 12](#_Toc144301359)

[Recursion: 12](#_Toc144301360)

[Useful Terms 12](#_Toc144301361)

[State space search 12](#_Toc144301362)

[implicit graph representation 12](#_Toc144301363)

[NP-hardness 12](#_Toc144301364)

[Buble sort 13](#_Toc144301365)

[Insertion 13](#_Toc144301366)

[Merge sort 13](#_Toc144301367)

[Quick sort 13](#_Toc144301368)

[Not a DnQ technique: 13](#_Toc144301369)

[Heap Sort 14](#_Toc144301370)

[Selection Sort 14](#_Toc144301371)

[Binary Sort 14](#_Toc144301372)

[Search Techniques 14](#_Toc144301373)

[Linear search 14](#_Toc144301374)

[Binary search 14](#_Toc144301375)

[Heap and Priority Queues 14](#_Toc144301376)

[BigO Notation 15](#_Toc144301377)

[O(n) 15](#_Toc144301378)

[O(1) 15](#_Toc144301379)

[O(n) 15](#_Toc144301380)

[O(log n) 15](#_Toc144301381)

[O(n^2) 15](#_Toc144301382)

[O(nm) 15](#_Toc144301383)

[Trees 15](#_Toc144301384)

[Hashing 17](#_Toc144301385)

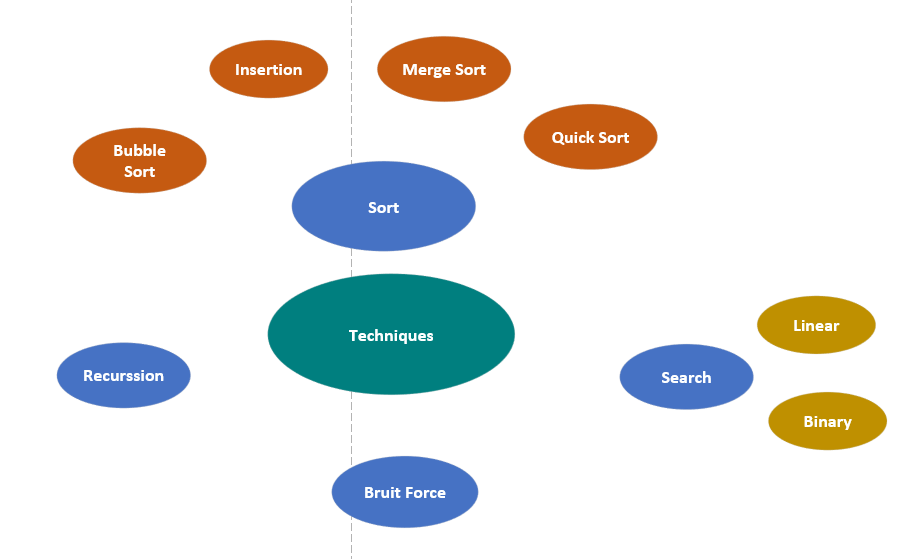
[Set 20](#_Toc144301386)

[Concurrency 20](#_Toc144301387)

[Data Structure 22](#_Toc144301388)

# Algo Mind maps

## Techniques



## Data Structure

A diagram of data structure

Description automatically generated

## All Algo

A diagram of algorithm

Description automatically generated

Machine Learning

A diagram of machine learning

Description automatically generated

# Algorithm Types and Definitions

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Chapter** | **Content** | **Purpose** |
| 2 | Elementary data structure |  |  |
| 2.1 |  | Stack and queues |  |
| 2.2 |  | trees |  |
| 2.3 |  | heaps and heapsort |  |
| 2.4 |  | sets and disjoints set union |  |
| 2.5 |  | graphs |  |
| 2.6 |  | hashing |  |
| 3 | Divide and Conquer |  |  |
| 3.1 |  | the general method | Splitting the inputs into k distinct subsets, yielding k subproblems and these subproblmes must be solved and method must be found to combine subsoluitons into the solution of the whole. |
| 3.2 |  | binary search |  |
| 3.3 |  | finding the maximum and minimum | |
| 3.4 |  | mergesort |  |
| 3.5 |  | quicksort |  |
| 3.6 |  | selection |  |
| 3.7 |  | strsassen's matrix multiplication |  |
| 4 | The greedy method |  | the most straightforward design technique and can be applied to wide variety of problems.Most though not all of these problems have a n inputs and require us to obtain a subset that satisfies some constraints. Any subset that statisfies this constraint is called feasible solutio. We are required to find a feasible solution that either maximize or minimize a given objective function. A FS that does this is called a optimal solution.It suggests that one can device a algorithm which works in stages, considering one input at at time. At each stage, a decision is made regarding whether or not a particular input is in an optimal solution. This is done by considering the inputs in an order determined by some selection procdure. Input is not added in next stage in case it not constructs Optimal solution. |
| 4.1 |  | the general method |  |
| 4.2 |  | optimal storage on tapes |  |
| 4.3 |  | knapsack problem |  |
| 4.4 |  | job sequencing with deadlines |  |
| 4.5 |  | optimal merge patterns |  |
| 4.6 |  | minimum spanning trees |  |
| 4.7 |  | single source shortest paths |  |
| 5 | Dynamic programming |  | Design method that can be used when the soluton to a problem may be viewed as the result of sequence of decision.In DP an optimal solution sequence of decisions is arrived at by making explicit appeal to the Principle of Optimality. It states that an Optimal sequence of decisions has the property that whatever the initial stage and decision are, the remaining decisions must constitute an optimal decision sequence with regards to the state resulting from the first decision.The greedy method only one decision sequence is ever generated. In DP many decisions sequences may be generated.However, sequence containing suboptimal subsequences cannot be optimal and so will not be generated. |
| 5.1 |  | the general method |  |
| 5.2 |  | multistage graphs |  |
| 5.3 |  | all pairs shortest paths |  |
| 5.4 |  | optimal binary search trees |  |
| 5.6 |  | 0/1 knapsack |  |
| 5.7 |  | reliability design |  |
| 5.8 |  | the traveling salesperson problem | |
| 5.9 |  | flow shop scheduling |  |
| 6 | Basic search and traversal techniques | | When the search necessarily involves the examination of every vertex(node) in the object being searched , it is called a traversal. 3 Categorized or techniques are there i.e. Binary and Binary tree and third is for graphs. These search strategies may not examine all vertices and so are referred to only as search methods. Nodes visited term is used when it search or traverse.Inorder,post-order,preorder. |
| 6.1 |  | the techniques |  |
| 6.2 |  | code optimization |  |
| 6.3 |  | and/or graphs |  |
| 6.4 |  | games trees |  |
| 6.5 |  | biconnected components and depth first search | |
| 7 | Backtracking |  |  |
| 7.1 |  | the general method | Backtracking is a general algorithm for finding solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons a candidate as soon as it determines that the candidate cannot possibly be completed to a valid solutio |
| 7.2 |  | the 8-queend problem | In order to apply the backtrack method, the desired solution must be expressible as an n-tuples, where x is choosed from finite set.Often the problem to be solved call for finding one vector which maximize or minimize or statisfies a criteria function P. the brute force approach would be to form all of these n-tuples and evalute each one with P, saving those which yield the optimum. The backtrack algo has as its virtue the ability to yield the same answer with far fewer than m trials. The basic idea is to build up the same vector one component at a time and to use modified criterion function(some time called bounding fn) to test wheter the vector being formed has any chance of success.. Possible left over vectors can be ignored if they not lead to solution. |
| 7.3 |  | sum of subsets |  |
| 7.4 |  | graphs coloring |  |
| 7.5 |  | hamiltonian cycles |  |
| 7.6 |  | knapsack problem |  |
| 8 | Branch and Bound |  | The algorithm explores *branches* of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated *bounds* on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm. |
| 8.1 |  | the method | It refers to all state space search methods in which all children of the E-node are generated before any other live node can become the E-node. |
| 8.2 |  | 0/1 knapsack problem | Branch and bound is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimizatio |
| 8.3 |  | traveling salesperson |  |
| 8.4 |  | efficiency considerations |  |
| 9 | Algebraic simplification and transformations | | Convert s1 to s2 for easy calculation. Polynomial and coeffiecents are discussed here |
| 9.1 |  | the general method |  |
| 9.2 |  | evaluation and interpolation |  |
| 9.3 |  | the fast fourier transform |  |
| 9.4 |  | modular arithmetic |  |
| 9.5 |  | even faster evaluation and interpolation | |
| 10 | Lower bound theory |  | The **lower bound theory** is the method that has been utilized to establish the given **algorithm** in the most efficient way which is possible. This is done by discovering a function g (n) that is a **lower bound** on the time that any **algorithm** must take to solve the given problem. |
| 10.1 |  | comparison trees for sorting and searching | |
| 10.2 |  | oracles and adversary arguments | |
| 10.3 |  | techniques for algebraic problems | |
| 10.4 |  | some lower bounds on parallel computation | |
| 11 | np-hard land np-complete problems | | In computational complexity theory, **NP-hardness** (non-deterministic polynomial-time hardness) is the defining property of a class of problems that are informally "at least as hard as the hardest problems in NP". A simple example of an NP-hard problem is the subset sum problem. |
| 11.1 |  | basic concepts |  |
| 11.2 |  | cook's theorem |  |
| 11.3 |  | np-hard graph problems |  |
| 11.4 |  | np-hard scheduling problems |  |
| 11.5 |  | np-hard code generation problems | |
| 11.6 |  | some simplified np-hard problems | |
| 12 | Approximation algorithms for np-hard problems | |  |
| 12.1 |  | introduction |  |
|  |  | absolute approximations |  |
|  |  | e-approximations |  |
|  |  | polynomial time approximation schemes | |
|  |  | fully polynomical time approximation schemes | |
|  |  | probabilistically good algorithms Recursion: |  |
| s.no | Topic |  | Description |
| 1 | Recursion |  | It is one of the technique in which function calls itself to perform some repetative tasks. It is good in some cases like divide and conquor technique but has exponential time complexity and it increases by the N value so in case N value is small then its less and it is highly dependent on the algoritm technique that you are using to solve any given problem. |
|  |  |  | We can't apply recursion anywhere and most of the time you use Iterative programming to get your work done. |
| 2 | Iterative design |  | https://www.geeksforgeeks.org/difference-between-recursion-and-iteration/ |

# Useful Terms

|  |  |  |
| --- | --- | --- |
| 1 | State space search | is a process used in the field of computer science, including artificial intelligence (AI), in which successive configurations or *states* of an instance are considered, with the intention of finding a *goal state* with a desired property. |
| 2 | implicit graph representation | In the study of graph algorithms, an implicit graph representation (or more simply implicit graph) is a graph whose vertices or edges are not represented as explicit objects in a computer's memory, but rather are determined algorithmically from some other input, for example a computable function. |
|  |  | Used in search algorithms and like unordered adjacent node/list |
|  |  |  |
| 3 | NP-hardness | In computational complexity theory, **NP-hardness** (non-deterministic polynomial-time hardness) is the defining property of a class of problems that are informally "at least as hard as the hardest problems in NP". A simple example of an NP-hard problem is the subset sum problem. |
|  |  | |  | | --- | | The **subset sum problem** (SSP) is a decision problem in computer science. In its most general formulation, there is a multiset S {\displaystyle S} of integers and a target-sum T {\displaystyle T} , and the question is to decide whether any subset of the integers sum to precisely T {\displaystyle T} | |

Sorting and Searching:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no** | **Algo Type** | **Description** | **Big O** |
| 1 | Buble sort | Iteration with each other and swap at same time | Bubble sort best O(n) , avg O(n^2) , worst O(n^2) |
| 2 | Insertion | Iteration only once and just create placeholder where it finds a issue | Insert sort best O(n) , avg O(n^2) , worst O(n^2) |
|  |  | **Stable** is a property to store all timeseries data in proper order |  |
|  |  | Insertion is well suited for low memory condtions and best suited for condition when most of the data is already sorted and only require O(1) memory and even sorting can be done on streaming data. |  |
|  |  | Insertion into binary search tree O(log n) for average case and O(n) for worst case | O(log n) for average case and O(n) for worst case |
| 3 | Merge sort | divide and conquer approach | Merge sort require Best O(n log n), avg O(n log n) and worst O(n log n) |
|  |  | Memory requirement is O(n) and it is stable as equal items retain their relative position |  |
| 4 | Quick sort | divide and conquer approach | Quick sort require Best O(n log n), avg O(n log n) and worst O(n^2) |
|  |  | Require O(n^2) operations in the pre-sorted case and in condition when you pick first value of array |  |
|  |  | First **choose Pivot point and then partition the elements** around pivot point and repeat for each partition |  |
|  |  | reduce the problem down to the most basic form, Require only O(log n) additional memory, there exists many optimization to improve performance |  |
|  |  | Quick sort has O(n^2) as worst case complexity where as in best case it has O(log n) |  |
|  | Not a DnQ technique: |  |  |
| 1 | Heap Sort | Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the minimum element and place the minimum element at the beginning. We repeat the same process for the remaining elements. |  |
| 2 | Selection Sort | The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays |  |
| 3 | Binary Sort | The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n). |  |
|  | Search Techniques |  |  |
| 5 | Linear search | Linear searching must examine each array item until the value is found giving it O(n) algorith complexity |  |
| 6 | Binary search | Binary search is like a sorted dictonary, This technique uses the middle value on sorted array | Binary search is having O(log n) performance |
|  |  | Sorting is more expensive |  |

# Heap and Priority Queues

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Main** | **SubTopics** |  | **Description** |
|  | Heap |  |  | A tree based container type that provides O(1) access to the minimum (min-heap) or maximum (max-heap) value while satisfying the heap property |
|  |  | Heap Property |  | the value in the current tree node is greater than, or equal to, its children (max-heap) |
|  |  | Complete tree |  | a tree where every level if filled out from left-to-righ be starting the next leve. |
|  |  | Trees as Array |  | Complete binary tree can be compactly stored as arrays eliminating all structural overhead and providing O(1) data access. |
|  |  | Rule of heap |  | Each element/node is smaller than its children |
|  |  |  |  | elements are added once at a time from left to right |
|  |  |  |  | Behind the scene Trees are stored in dynamic array |
|  |  |  |  | how it is stored: K -> 2K -> 2k+1 |
|  |  |  |  |  |
|  |  |  |  | **Insertion** is done in 2 steps in heaps i..e added at the bottom of the tree on left most and then as second step heap is reorganized to satisfy its rule. |
|  |  |  |  | Insert : O(log N) and reallocation is O(N) and amortized is O(log N) |
|  |  |  |  |  |
|  |  |  |  | **Removal** is 3 step procedure: top min value is removed and bottom values is replaced on top and then heap rule gets applied |
|  |  |  |  | Big Os is same as insertion |
|  |  |  |  |  |
|  | Priority queue | |  | a queue that pops item in priority, non FIFO, order |
|  |  |  |  | Heap is ideal data sructure to accommodate urgent queue conditions. |

# BigO Notation

|  |  |  |
| --- | --- | --- |
| S.No | Big O | Meaning |
| 1 | O(n) | = O(2n), O(n+1) |
| 2 | O(1) | the cost of the algo is unchanged by the input size. |
| 3 | O(n) | These algo increase linearly.Iterating over constant value of data is also categorized under this. |
| 4 | O(log n) | A function whose cost scales logarithmaically with the input size. |
| 5 | O(n^2) | A function that exhibits quadratic growth relative to the input size. |
| 6 | O(nm) | A function which has two inputs that contribute to the growth |
|  |  | amortized complixity - it is a combined complexity instead of single entityt |

# Trees

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | **Traversal Technique:** |  | Description |
|  | NOTE: The order in which we have to do traversal is important as that leads to performance. Like Vist left child, right child and process the current value |  |  |
| 1 | **Pre Order Traversal** |  | The node is visited before its children and left is visited and then in last right and we do this recursively. |
|  |  |  | Usage: Pre order tree is used to copy functions like file copy , structurally identical to original tree as it copies things as it is. |
|  |  |  |  |
| 2 | **In Order** |  | The left child is visited before the node, then the right child |
|  |  |  | Usage: it leads to sort order and priting sorted list |
|  |  |  |  |
| 3 | **Post order** |  | The left and right childrents are visited before the node. |
|  |  |  | Usage: delete every node in the tree as first you have to delete the leaves. Example deleting a directory where first you have to clean all files or subdirectories |
|  |  |  | Traversal complexity - Avg O(log n) and worst is O(n) |
|  |  |  | Removal complexity - Avg O(log n) and worst is O(n) |
|  |  |  | visiting a node in tree is having O(n) complexity, all 3 are having O(n) for both avg and worst |
|  |  |  |  |
|  | **Binary search Tree** |  | A sorted data structure where each node can have 0-2 children and each node , except the root, has exactly one parent. |
|  |  |  | Binary tree insert, remove, and search operations are O(log n) average case |
|  |  |  |  |
|  |  |  | Unbalanced tree whose left and right children have uneven heights. |
|  |  |  | A fully unbalanced binary tree is just a linked list with O(n) algo complexity |
|  |  |  | Balanced tree - a binary search tree whose maximum height is minimized. |
|  |  |  | Balance factor - the difference between the height of the left and right sub-trees |
|  |  |  | Self balancing - the tree is balanced as nodes are added or removed from the tree |
|  |  |  | self balancing algorithm - left rotation, right rotation, left-right rotation and right left rotation. |
|  | **B-Tree** |  |  |
|  | A sorted, balanced , tree structure typically used to access data on slow mediums such as disk or tape drives |  |  |
|  | Nodes will always have one more child than values |  |  |
|  | Minimal degree |  | the minimum number of children that every non-root node mush have. Represented as te variable T |
|  | full degree |  |  |
|  | height |  |  |
|  | Searchin B-Tree |  | compare each value in the current node with the sought value, and then recursively check every child's value. |
|  |  |  | Searching nodes values can be performed using a binary search |
|  |  |  | splitting the root is the only wy that the b-tree height increases |
|  | node splitting |  |  |
|  |  |  |  |
|  |  |  | binary tree are efficient in storing and accessing data |
|  |  |  | Adding items in linked list is just O(1) |
|  |  |  | Sorted is double link list |
|  |  |  | Adds a specified item to a linked list in the sort order of the item type complexity is O(n) because in sorted we have to first find the list and then enter and that s the reason it is having this complexity. |
|  |  |  | Tree is speical type of graph without edges and don't have any cycles and is undirected |
|  |  |  |  |
|  |  |  | balanced and unbalanced trees and now its O(n) |
|  |  |  | Big represnts both best and Worst where as 2 more notation exist that small 'o' and thita |

# Hashing

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hashing | hashing is used for a variety of purposes including cryptography, data indexing, and data compression |  |
|  | Hash table | An associative array container that provides O(1) insert, delete and search operations. |  |
|  | Hash Collision | When multiple distinct keys would be inserted at the same hash table index. |  |
|  | Separate Chaining solves above problem - | Collisions in a hash table are chained together into a linked list whose root node is the hash table array entry. |  |
|  |  | The has table handles collisions by linking all the values with the same table index into a linked list of entries. |  |
|  | **there are 2 ways to solve has collision:** |  |  |
|  |  | Fill Factor | the percentage of capacity representing the maximum number of entries before the table will grow. Eg 0.80 |
|  |  | Growth factor | The multiple to increase the capcity of the hash table when the fill factor has been exceeded. Eg 1.50 |
|  |  |  |  |
|  |  | set, dict are heaps |  |
|  |  |  |  |
|  |  | All strings -> fixed length strings |  |
|  |  | md5 is 1028 bits and it is in hexadecimal units |  |
|  |  | All strings -> 32bit integers |  |
|  |  | collision solution is to do overflow and second technique is to override existing value or final solution is to dynamic array by doing double size and move existing items. |  |
|  |  | Re organization is called rehashing. |  |
|  |  | hash table don’t gurantee ordering and don't support element sequenceing |  |
|  |  | add element in hash is O(1) , worst case O(N) and amortized is O(1) |  |
|  |  | Mostly such overflow doen't happen in hast table and in case it is happening then it is sign of poor hash function. |  |
|  |  | in Array and in link list search is in O(N) and in hash its O(1) and that is much faster and in worst case scenerio of overflow chain then therotically worst case is O(N) |  |
|  |  | In hash table we stored plain values but there is another variant i.e. Dictonary or maps are hash and is kind of solution |  |
|  |  | has table are ideal for caching as it can store in key value pair as it can speed up search |  |
|  |  | Array offer ordering where as hash doesn't offer this and that is one of the feature in comparison. |  |
|  |  |  |  |
|  |  | <https://app.pluralsight.com/course-player?clipId=73d108fe-db03-4d34-82f5-d3d10ead9b53> |  |
|  |  |  |  |
|  |  | good link about usage of hashing | https://freemanlaw.com/hash-collisions-explained/ |
|  |  |  |  |
|  | Hashing Algorithms |  | Although the **MD5** hashing algorithm was once one of the most popular hashing algorithms out there,  it is now a compromised algorithm that readily generates collisions. Nowadays, the Secure Hash Algorithm (SHA-2 and SHA-3) family of hashing algorithms is considered the most secure. |
|  |  |  |  |
|  | Solution of hash algo problem |  | One way to always have a perfect hash function is to increase the size of the hash table so that each possible value in the item range can be accommodated. This guarantees that each item will have a unique slot. |
|  |  |  |  |
|  | Goal |  | Our goal is to create a hash function that minimizes the number of collisions, is easy to compute, and evenly distributes the items in the hash table. There are a number of common ways to extend the simple remainder method. We will consider a few of them here |
|  |  |  |  |
|  | Collision Resolution: |  |  |
|  |  |  | a. Open addressing technique called linear probing.(It leads to clustering problem) |
|  |  |  | b. Quadratic probing |

# Set

|  |  |  |
| --- | --- | --- |
| **Algo** | **Function type** | Description |
| Set Algorithms | union | creates an output set that contains the distinct items form both input sets. It answers OR questions. set operations are O(Log n) |
|  | intersection | the of items that exists within both input sets. It answers AND questions |
|  | difference | the set of items which exists in one set which do not exist in the other. "but not" questions |
|  | symmetric difference | the set of items which exists in either of the two input sets, but which are not in their intersection. answers "OR" and BUT NOT both questions |
|  |  |  |
|  |  | Set- Is a data structure that stores unique values in an underminded order |

# Concurrency

|  |  |  |
| --- | --- | --- |
| concurrency | | two or more operations executing at the same time |
|  |  | concurrent updates to non-concurrency-safe collections can lead to unexpected behavior and data loss |

Brute Force Algo

|  |  |  |
| --- | --- | --- |
| **S.no** | **Algo type** | **Description** |
| 1 | Brute force Approach | Unlocking problem of password |
| 2 | Greedy Approach | There is caveat so make sure you pick correct low point to start.  Shortest path problem can be solved using this but still it is not the only right way and even coin problem is used to teach this concept so please check in OneNote. |
| 3 | Divide and conquer | Apply recursively |
|  |  | **There are 3 things to consider:** |
|  |  | a. divide into sub problem |
|  |  | b. solve micro problem |
|  |  | c. assemble parent solution |
|  | **Examples** |  |
|  |  | Example used for DnQ is **sorting alphabets** using quick sort option |
|  |  | Merge sort can be used to sort much data like 5TB of data in just 64GB of memory space. |
|  |  | *Merge sort has good property to store large amount of data using small footprint of memory.* |
|  |  | DnQ can be used to **multiply larger numbers** |
|  |  | Binary search is another example of DnC technique. |
|  |  | DnQ is used to **find two close points** in a plane |

Dynamic Programming

|  |  |
| --- | --- |
| **Problem statement that DP solves:** | In DnQ there are situation where two problems has some part of shared problem that each independent gets solved and in large data set It is duplicate efforts |
| How DP solves abobe problem: | In DP such overlapping part is only solved once and saved in cache. |
|  | Fibonaci with recursive approach takes lot of time but as soon as you adds cache using dynanamic programming approach  then big fibonaci got given results in millisecond like try fb(10) and fb(110) using both recursive and dynamic apprach |
|  |  |
|  |  |
| top down and bottom up aproach to solve fibonaci solution, | **Top down** - call top level function, and build cache and solution along the way |
|  | **Bottom up** :construct cache first, deduce solution next. |
|  | Tabulation: Bottom Up  Memoization: Top Down |
| **Example:** | Pick most valuable VM's for a server within its capacity |

# Data Structure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| s.no | Data Structure | Python lib | Pros | Cons | BigO | Other |
| 1 | Linked List |  | Appends two lists in O(1) time. Simplifies some operations. Eg queues. | No Preallocation: can fill storage Increased element access time and pointer overhead may be large. | O(1) for removal and O(1) for head and tail where as O(n) for insertion, search etc as then it has to travese whole list |  |
|  |  |  |  |  |  |  |
| 2 | Priority queue | heapq in python | Data structure is organized into heaps | |  | Min, Max, Interval, Min-Max and Interval heap |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 3 | Hash | set, dict | Hash tables to the rescue when there is increase in execution time in linked list. |  |  |  |
|  |  |  | Has function property is to reduce big data to smaller data and has property to change output in case if there is any change in original data. |  |  |  |
|  |  |  | There is possibility of collision | |  |  |
|  |  |  |  |  |  |  |

def definition():

print("list = [] - list stores a series of items in a particular order. You access items using an index, or within a loop.\n \ you can have different data types in list, sublist")

print("dict() = {} - Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair")

print("tuple() = () - Tuples are similar to lists, but the items in a tuple can't be modified. \n But we if element is mutable \ then we can modify elemement. \n Tuples can be reassigned")

print("set() = {} - The order of elements in a set is undefined. You can add and delete elements of a set,\n You can iterate the \ elements of the set, you can perform standard operations on sets (union, intersection, difference).\n Set contains unique \

values where as list don't\n Elements are immutable but set itself is mutable")

print(" - A set is a collection which is both unordered and unindexed \n Sets are used to store multiple items in a single variable\ \n Set items are unordered, unchangeable, and do not allow duplicate values.")