Data Structures, Concepts & Algorithms

Data Structures	Algorithms	Concepts
Linked Lists	Breadth First Search	Bit Manipulation
Binary Trees	Depth First Search	Singleton Design Pattern
Tries	Binary Search	Factory Design Pattern
Stacks	Merge Sort	Memory (Stack vs Heap)
Queues	Quick Sort	Recursion
Vectors / ArrayLists	Tree Insert / Find / etc	Big-O Time
Hash Tables		

Data structures are methods to store information. Data structures can be combined with each other

Major operations include: tracking heads/pointers, insertion, deletion, search, traversal, sort

Comparison of major data structures in Python

Name	*Notation*	*Ordered?*	*Data Types*	*Item duplicates?*	*Length*	*Mutable?*
Vector	[]	Ordered	1	Duplication	Var	Mutable
Set (fast)	()	**Unordered**	1 or many	**Unique**	Var	Mutable
Tuple	**{}**	Ordered	**Many**	Duplication	**Fix**	**Immutable**
List	[]	Ordered	1	Duplication	var	Mutable
ArrayList	[]	Ordered	1 (transforms type when added)	Duplication	var	Mutable

- Array – numbered list of elements
    - Can be accessed quick, but deletion takes a lot of time, because all subsequent elements have to be shifted accordingly
    - Dynamic arrays (or ArrayLists) can change in size (and include pointers), static array have a predefined, fixed size
    - Vectors are one-dimensional, dynamic arrays (either 1 column or 1 row)
    - Array offers direct access to random items (linked lists do not)
    - For insertion or deletion, existing items in an array have to be moved
- String: read-only array of characters
- Lists: Linked List – list/sequence of numbers/nodes that contain a pointer that points to the next element of the sequence
    - In C, a list is a struct (data type)
    - Advantage to arrays – easier to insert new items, no fixed length
    - Position -1 is last item, head is first item, tail is last item
    - Quite slow
    - Singly linked lists can be traversed only in forward direction
    - In a Doubly-linked list, each node has 2 pointers, one to the previous, and one to the next node. Thus, the list can be traversed in either direction
    - Circular linked lists (singly and doubly) have no head or tail (no ends)
- Stack (LIFO)
    - LIFO (last in first out, like a stack of plates)
    - Variables are directly stored in memory, access is very fast but a lot of dependencies to other elements / parts of the program that need to be executed before the stack

    - Thus useful for tasks that are divided into subtasks
    - Push (insertion) and pop (deletion)
    - Implemented via array in Python
- Queu (FIFO)
    - FIFO (first in first out, like cinema queu)
    - Priority queu assigns priority to each element and the elements come out in order of priority Implemented via array in Python
- Hash Table
    - Unique keys are mapped to values
    - Linear probing solves potential conflicts where 2 or more values are assigned the same key
    - Stores and retrieves data quickly. Is not able to order data (thus needs more memory)
- Trie (from retrieved)
    - Tree structure that stores the alphabet, strings can be retrieved from the trie
    - Works with linked nodes and pointers and an empty root node on top
- Binary Tree
    - Tree structure with each node having at most 2 children nodes (left and right)
    - Binary search trees (very common) store items in such a tree structure in memory
    - Binary heaps are binary search trees where the key/value of the parent node is larger (max-heap) or smaller (min-heap) than the individual key/value of any of its children nodes
        - Heaps have memory allocated at run time (slower than stacks), but no dependencies between elements. You can allocate and free a block at any time
    - Breadth First Search (stops when node is found)
        - Analyzes tree from top to bottom, exploring all items of tree layer before moving on to next deepest layer
    - Depth First Search (stops when node is found)
        - Analyzes tree from top to bottom as far as possible, exploring a full tree branch before moving on to subbranches
        - Can be implemented by a Queu (FIFO)
    - Traversal (Search + Operation)
        - Preorder (first node, left child, right childs
        - Inorder (left child, first node, right child)
        - Postorder (left child, right child, first node)
- Sorts
    - Insertion (elementary) sort: gets elements and adds them to a new sorted list
    - Selection (elementary) sort: gets element, searches for smallest element and swaps them
    - Merge sort (divide & conquer): divide array into 2 parts, recursively sort the 2 subarrays and merge them
    - Quick sort (divide & conquer): shuffle array and partition it in 2 parts, then in 4 parts etc.
    - Bubble sort: Comparing and switching (if needed) two elements with each other
    - Bucket sort: Partition into sorted buckets and distribute elements to buckets
- Dictionary
- Graphs
    - undirected and directed graphs (nodes can have multiple parent nodes), trees (each child node only has 1 parent node)
    - Values or weights between nodes are called “edges”
    - graphs model networks, trees model hierarchies

Algorithms are solving problems:

- Dynamic Connectivity / Union-Find / Disjoint-set
    - Tries to find path to connect 2 objects
    - Dynamic connectivity dynamically maintains information about the connected components of a graph
- Dijkstra's algorithm
    - Also searches a tree, but based on weights of connections
- Binary Search
    - Finds position of a target item within a sorted array
    - Identifies median, compares it to target value and eliminates half of the array, iteratively

Concepts:

- Dynamic Programming (DP) refers to breaking a large problem down in several subproblems which can be optimized individually (and referred back to by "recursion") and its solutions stored and called by "memoization", to optimize the larger problem
- Object-oriented Programming (OOP) – programming based on classes that include attributes and actions that define objects. Main properties:
    - Encapsulation (hiding object data to public)
    - Inheritance (one class can be a more specialized version of another class)
    - Polymorphism (processing objects differently based on their data type or class)
- Divide & Conquer (D&C) is recursively solving subproblems to solve a larger problem (no memoization), while DP solves overlapping problems
- Denormalization is a database optimization technique applied to a database after normalization. Redundant data is added to one or more tables in order to decrease cost of a read/query
- Atomicity: transaction groups commit changes to relational databases in groups which are either all rejected (and the database rolled back to its previous state) or all applied
- Recursion refers to functions calling themselves. It is used in solving a large problem based on solutions of instances of smaller versions of that problem
    - Implemented by a function for a function to call itself
    - If combined with a lookup table and memoization, we can call it dynamic programming
- Big-O Notation relates to how fast the processing time (in terms of number of operations compared to the number of elements n) of an algorithm grows when the input data grows (Big-O refers to worst case scenario, asymptotic running time with large input)
    - O(n) refers to linear time, because the processing time increases linearly with the number of elements (e.g. comparing each value in an array to a max value)
    - O(1) = constant running time is an algorithm that always takes the same time, regardless of the input size
    - PIE p.28
- Memory Footprint Analysis
    - How much memory does an algorithm use as a function of n (number of input elements) when performed?
    - Make sure to know how much memory a specific class requires in general
- Greedy algorithms optimize all subparts independently instead of the whole system at once
    - As e.g. decision tree
- Bit Manipulation
    - Bitwise operators like not, or, and, xor based on 0 and 1
    - Interesting to save memory

- Creational Design Patterns: manage class selection and object creation
    - Singleton Design Pattern
        - Instantiation of a class is restricted to 1 object
        - Interesting i.e. for hardware interface access
    - Builder Pattern: creates objects stepwise without knowing how they are constructed
    - Factory Design Pattern: factory method is added to class
        - Objects that do not have a class will be created using factory methods which are separate predefined operations
        - Subclasses can then override this method if needed
    - Abstract Factory: implementation of factory is separated from usage of factory
- Behavioral Design Patterns: how classes and object interact and communicate
    - Iterator: traversal through data (uni-/bidirectional, with or without manipulation)
    - Observer: object reports state changes to an observer
- Structural Design Patterns
    - organize relationships between classes and objects, providing guidelines for combining and using related objects together to achieve desired behaviors.
    - Decorator/Wrapper: wraps object in another object that implements same interface (called component)
- Testing. Key questions (CtCI):
    - Use cases and outside of use cases (plus performance expectations)
    - What does failing mean
    - Stress conditions on it
- Cryptography
    - A hash function takes arbitrary input (such as a password) and produces a fixed length output = hash, which is stored in a database
        - It is computationally infeasible to compute the original input from the hash
        - SHA-256 often used example
    - Symmetric (shared) key cryptography: 2 people using same key to encrypt and decrypt information
    - Public key cryptography: public key for encryption and private key for decryption