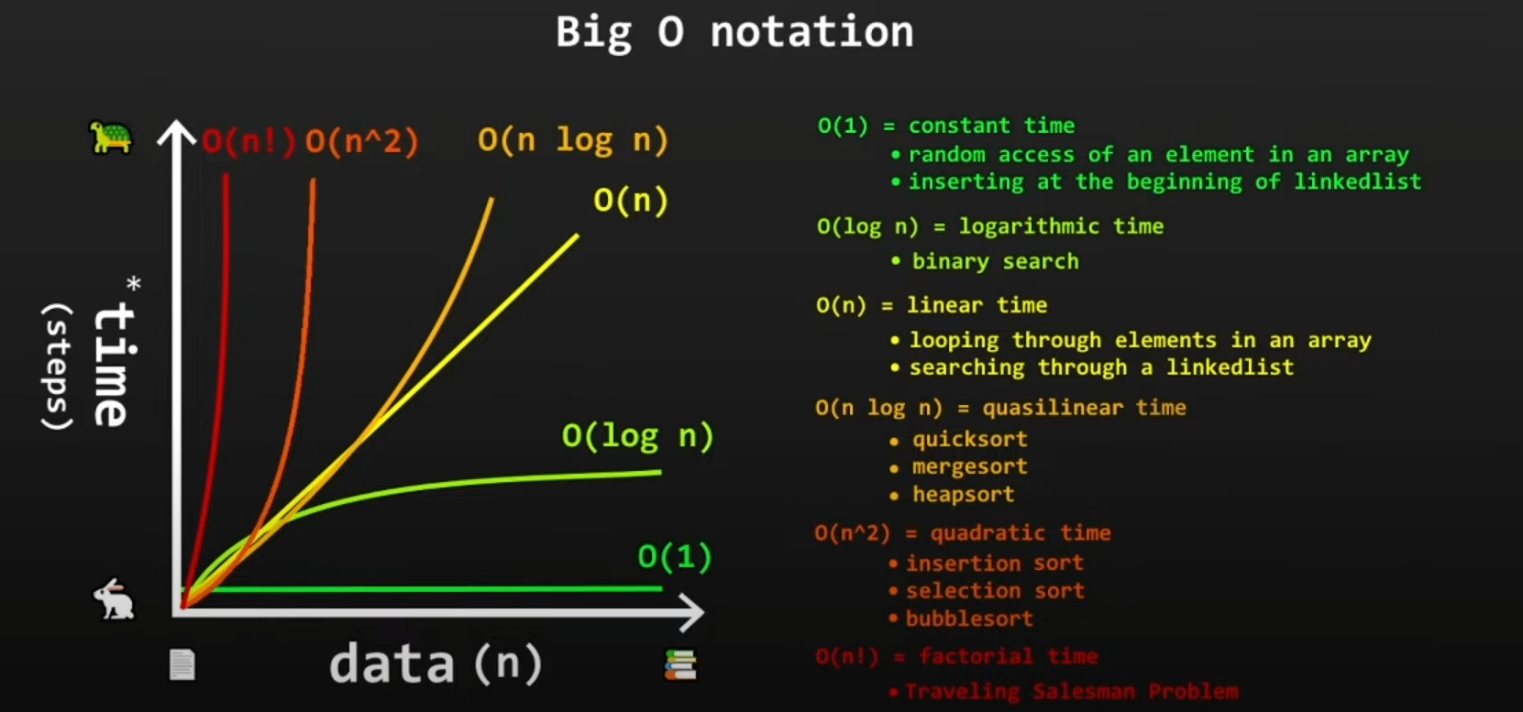
**Algorithm and Data Structure**

**Big O Notation**

1. Describes the performance of an algorithm as the amount of data increases.
2. Machine independent (# of steps to completion).
3. Ignore smaller operations O(n + 1) -> O(n).



**Algorithm:** a collection of steps to solve a problem.

**Data Structure:** a named location that can be used to store and organize data.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data Structure** | **Definition** | **Purpose** | **Space** | **Search** | **Insert** | **Delete** |
| Stack | LIFO: Last-In First-Out   * Stores objects into a sort of “vertical tower” * push() to add to the top * pop() to remove from the top | 1. Undo/redo features in text editors  2. Moving back/forward through browser history  3. Backtracking algorithms (maze, file directories)  4. Calling functions (call stack) |  |  |  |  |
| Queue | FIFO: First-In First-Out   * A collection designed for holding elements prior to processing * Linear data structure * add = enqueue, offer() * remove = dequeue, poll() | 1. Keyboard Buffer  2. Printer Queue  3. Used in LinkedLists, PriorityQueues, Breadth-first search |  |  |  |  |
| Priority Queue | A FIFO data structure that serves elements with the highest priorities first before elements with lower priority. |  |  |  |  |  |
| Linked List | * Stores Nodes in 2 parts (data + address) * Nodes are in non-consecutive memory locations * Elements are linked using pointers   Singly Linked List:  [data | address] -> [data | address]  Doubly Linked List**:**  [address | data | address]  <-> [address | data | address] | 1. Implement Stack/Queues  2. GPS navigation  3. Music playlist |  | O(n) | O(1) | O(1) |
| Dynamic Array | * In Java, known as ArrayList * Dynamic capacity 🡪 array size increases by factor of 3/2 every time you reach over capacity |  |  | O(1) |  |  |
| Hash Table | * A data structure that stores unique keys to values ex. <Integer, String> * Each key/value pair is known as an Entry * FAST insertion, look up, deletion of key/value pairs * Not ideal for small data sets, great for large data sets   Hashing: takes a key and computes an integer (formula will vary based on key & data type)   * In a Hash table, we use the hash % capacity to calculate an index number * key.hashCode() % capacity = index   Bucket: an indexed storage location for one or more Entries   * can store multiple Entries in case of a collision (linked similarly to a Linked list)   Collision: hash function generates the same index for more than one key 🡪 less collisions more efficiency | Run time complexity:  Best: O() ~ no collision  Worst: O() |  |  |  |  |
| Graphs | * Adjacency Matrix 🡪 time complexity: O(), space: O() * Adjacency List 🡪 : O(), space: O() |  |  |  |  |  |
| Adjacency Matrix | * A 2D array to store 1’s/0’s to represent edges * # of rows = # of unique nodes * # of columns = # of unique nodes |  | O() | O() | O() | O() |
| Adjacency List | * An array/arraylist of linkedlists. * Each linkedlist has a unique node at the head * All adjacent neighbours to that node are added to that node’s linkedlist |  | O() | O() | O() | O() |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Definition** | **Advantage** | **Disadvantage** | **Run-Time Complexity** | **Space Complexity** |
| Linear Search | * Iterate through a collection one element at a time | * Fast for searches of small to medium data sets * Does not need to be sorted * Useful for data structures that do not have random access (Linked List) | * Slow for large data sets | O() |  |
| Binary Search | * Search algorithm that finds the position of a target value within a sorted array * Half of the array is eliminated during each “step” |  |  | O(log n) |  |
| Interpolation Search | * Improvement over binary search best used for “uniformly” distributed data * “Guesses” where a value might be based on calculated probe results * If probe is incorrect, search area is narrowed, and a new probe is calculated |  |  | Average Case: O(log(log(n)))  Worst Case: O() [values increase exponentially] |  |
| Bubble Sort | * Pairs of adjacent elements are compared, and the elements swapped if they are not in order. |  |  | O() | O() |
| Selection Sort | * Search through an array and keep track of the minimum value during each iteration. At the end of each iteration, we swap variables. |  |  | O() | O() |
| Insertion Sort | * After comparing elements to the left shift elements to the right to make room to insert a value | * Less steps than bubble sort | * Best case is O() compared to selection sort | O() | O() |
| Recursion | * Repetition of an internal process * Apply the result of procedure to a procedure * A recursive method calls itself, can be a substitute for iteration * Divide a problem into sub-problems of the same type as the original * Commonly used with advanced sorting algorithms and navigating trees | * Easier to read/write * Easier to debug | * Sometime slower * Uses more memory | N/A | N/A |
| Merge Sort | * Recursively divides array in 2, sorts, re-combine |  | * Linear space, other previous sorts use constant space | O() | O() |
| Quick Sort | * Moves smaller elements to left of a pivot and recursively divide array into 2 partitions |  | * Worst case if the array is already sorted * Higher space complexity due to recursion | Best: O()  Avg: O()  Worst: O() | O() |
| Depth First Search | * A search algorithm for traversing a tree or graph data structure   1. Pick a route.  2. Keep going until you reach a dead end, or a previously visited node.  3. Backtrack to last node that has unvisited adjacent neighbours. |  |  |  |  |
|  |  |  |  |  |  |

