Data structure and Algorithm

What is an algorithm?

Step by step instructions to solve a problem.

There might be many ways to solve a problem, its about finding the best solution to solve a problem.

An algorithm must give the same answer for the same set of input in a same problem.

Guidelines for algorithm

1. An algorithm must have a clear problem statement
2. An algorithm definition must a specific set of instructions In a specific order
3. Each step must not be complex ones, steps must not be able to be further broken down.
4. An algorithm must have an end result
5. The algorithm must complete and should not be infinite

LINEAR SEARCH

Searching from start to end one by one.

BINARY SEARCH

Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.

EFFICIENCY OF AN ALGORITHM – for efficiency we always check the worst case scenario

Two measures if efficiency – Time and space. Efficiency measured by time, time complexity is the measure of how long it takes to complete the job

Space complexity – amount of memory that an algorithm takes up.

TIME COMPLEXITY IN DETAIL

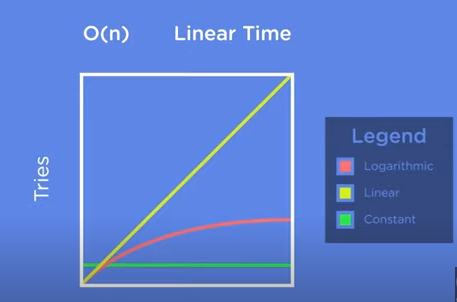
Big O – definition of complexity of an algorithm as a function of the size.it measures how the algorithm works in the worst case scenario.

An example of complexity written in terms of Big O looks like -> O(n)

O - Order of magnitude of complexity.

(n) – A function of the size.

* The big O notation for a **constant time** algorithm (example: reading a value in the list) is -> **O(1).** In this regardless of the size of ‘n’ the run time is constant for a single time.
* Big O notation for logarithmic time algorithm: the number of tries taken in binary search to find the worst case scenario for a given value N is log2 of n +1. Since this pattern is overall a logarithmic pattern we say that the runtime of such algorithm is logarithmic (also called sublinear). In Big O notation we represent a algorithm with **logarithmic run time** as -> **O(log n).** Inlogarithmic run time, as the n grows really large the number of tries increases very slowly and eventually flattens out.
* Big O notation for the **linear time algorithm**: as the number of operation to determine the result in the worst case scenario is almost the same as n, we say that the algorithm runs in linear time. The corresponding Big O notation is **-> O(n).**

****

Quadratic time complexity: In this, for any given value if n, we carry out n **²** number of operations. In Big O notation -> O(n **²**).

Cubic runtime algorithm : In this, for any given value if n, we carry out n ³number of operations. In Big O notation -> O(n³).

Quasilinear Runtimes : Big O notation -> O(n log n) for every value of n, we execute a log n number of operations. Sorting algorithm is one of the places that has this runtime. Ex merge sort.

Polynomial runtime: Big O notation -> O(n^k), where k is any number. If n = 2 -> quadratic runtime. If k = 3 -> cubic runtime

EXPONENTIAL RUNTIMES: an algorithm which has the big O value to be some number raise to the nTH power. Burt force algorithm has exponential runtime. These are really expensive or inefficient.

Recursion: it is something to be noted and it may come up more than one time while implementing algorithms in other languages.

Space Complexity : similar to time complexity space complexity is also an important aspect to check the efficiency of the algorithm are working on.

The space complexity of the iterative version of the binary search is a constant space, while the space complexity of the recursive version of the binary search logarithmic.