Algorithm:

An algorithm is any well-defined computational procedure that takes some values, or set of values, as input and produces some values, or set of values, as output. An algorithm is a sequence of computational steps that transforms the input into output.

Analysis of an Algorithm:

The analysis of an algorithm is a major task in computer science. There are some criteria to measure the efficiency of an algorithm. That criterion is determining the amount of time taken by each step of the algorithm to perform an operation, and the overall space or memory used by the algorithm altogether.

“The time complexity of an algorithm M can be defined as a function f(n) which gives the running time and/or storage requirement of the algorithm in terms of the size n of input data.”

Example: Searching a Text paragraph

Suppose we are given a text paragraph and we have to traverse the paragraph to search for a three-letter word.

If the word to be searched is ‘the’, it has a high probability and so we can find this word at the beginning of the paragraph. It will take less time to search the word, so f(n) will be small.

On the other hand, if the word under consideration is ‘zoo’, there is a chance that we don’t find the word in the complete paragraph. In this case the whole paragraph is to be traversed and so it will take relatively larger time than the previous consideration, f(n) will be large.

Best Case:

The minimum possible value of f(n) or the minimum number of comparisons made in order to search the item is included in the best case.

Worst Case:

Clearly, the worst possible case occurs if the item-to be searched is the last element of the data or is not present in the data. In both cases, we have to make comparisons that are equal to number of n elements present in the data.

Such that; C(n) = n

Average Case:

Here we assume that item does appear in the data, and that is equally likely to occur at any position in the data. Accordingly, the number of comparisons can be of the numbers 1, 2, 3, …, n, and each number occurs with probability p = 1/n. Then;

C(n) = 1. 1/n + 2. 1/n + 3. 1/n + 4. 1/n + … + n. 1/n

= (1 + 2 + 3 + 4 + … + n). 1/n

= [n. (n + 1) / 2]. 1/n

= (n+1) / 2

It says that the average number of comparisons needed to find the item is equal to half the number of elements present in the data given.

The complexity of an average case is much more complex to analyze than of worst case. Whereas, best case show only the minimum possible time complexity. So, in order to measure the time complexity of an algorithm Worst Case time complexity is considered, as it includes all the possible cases and is less complex than average case.

Rate of Growth

Suppose M is an algorithm with n inputs. The time complexity f(n) of M increases as the number if inputs n of M increases. This increase of f(n) with inputs is than characterized as the big O notation.

Standard Functions

Some standard functions are:

Log2 n, n, nlog2 n, n2, n3, 2n

The rate of growth of these functions is given as:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | log2 n | n | nlog2 n | n2 | n3 | 2n |
| 5 | 3 | 5 | 15 | 25 | 125 | 32 |
| 10 | 4 | 10 | 40 | 100 | 103 | 103 |
| 100 | 7 | 100 | 700 | 104 | 106 | 1030 |
| 1000 | 10 | 103 | 104 | 106 | 109 | 10300 |

The rate of growth of these standard function increases as we move from top to bottom in the table. This increase is due to increasing number of inputs. But this increase is immeasurably more if we look at the functions at the right side of the table. This implies that with some function the rate of increase of time complexity f(n) is more than others. The program or algorithm with less rate of increase of f(n) is considered to be the efficient one.

Big O Notation:

The rate on increase in terms of f(n) is written as:

|f(n)| ≤ |Mg(n)|

Also written as;

f(n) = O(g(n))

which is read as “f(n) is of order g(n)”

Well-known Algorithms:

The time complexity of certain well-known searching and sorting algorithms is given as:

* Linear Search: O(n)
* Binary Search: O(log n)
* Bubble Sort: O(n2)
* Merge Sort: O(nlog n)

Link: https://www.linkedin.com/pulse/what-time-complexity-jaweria-b