© April 2020 | IJIRT | Volume 6 Issue 11 | ISSN: 2349-6002

Methods in complexity of algorithms

Mr.Yashwanth Balan[1], Ms.Apoorva R Narendra[2], Ms. Pooja V[3]
[1,2,3] *T.John College, Bengaluru, India*

INTRODUCTION

An algorithm is a step by step instructions designed to perform a specific task. It can be a simple method, for addition, multiplying two numbers, or a complex operation,

The step by step procedure to complete a task with in a finite number of steps is known as an algorithm in general.

Algorithmic complexity is concerned about how fast or slow particular algorithm performs and how much memory it requires .Complexity of algorithm is a function of size of input of a given problem instance which determines how much running time/memory space is needed by the algorithm in order to run for the completion .

Algorithm analysis refers to the task of determining the computing time and storage space requirement of an algorithm. This is known as analysis performance or algorithmic efficiency which enables us to select an efficient algorithm.

When we have a problem to solve, there may be many algorithms to solve, and we will be choosing the best in it. The selection of best algorithm is possible by analysing the algorithms in proper manner.

We can analyse the algorithm in two ways

1. The correctness of the algorithm
2. Checking time complexity and space complexity

To calculate the analysis of algorithm, two types are needed

1. Priori analysis
2. Posteriori analysis

A. Priori Analysis

One of the creative analysis of algorithm. Will be having a function which bounds the algorithm computing time For example, if there a statement in the middle of the program, we have to calculate the total time that statement will spend for execution, given by some initial state of input data. This needs mainly two types of information

a. The statement of frequency count
b. The time taken for an execution

Since the time per execution depends on both, the machine being used and the programming languages used together with its compiler, a priori analysis limits itself to determine the frequency count of each statement.

The notation used in the priori analysis Big-Oh (O), Omega (Ω), theta (θ)

Priori analysis computing time ignores all of the factors, which are machine or programming language dependent and only concentrates on determining the order of the magnitude of the frequency of execution of the statements.

B. Posteriori Analysis

Will be collecting the actual statistics about the algorithm, conjunction of the time and space while executing.

To test is

a. Debugging – It is the process of executing programs on sample data sets that determine whether we get proper results ,if the fault occurs it has to be corrected
b. Profiling- The process of executing a correct program on actual data sets and measuring the time and space it takes to compute the results during execution. The actual time taken by the algorithm to process the data is called profiling.

In order to perform any of the above tasks two kinds of efficiencies are needed

1. Space Efficiency /Space Complexity
2. Time Efficiency/Time Complexity

Space Complexity

The count of required temporary storage required for running the algorithm

The space needed by an algorithm consists of the following components

1. Fixed static part

This part typically includes the instruction space, space for simple variables, space for constants and fixed size components variable

2. Variable dynamic part

It consist of space needed by component variables whose size is dependent on the particular problem instance at runtime being solved

Time Complexity

The total time required to run the program is noted as time complexity. The total time taken by the program is the sum of the compile and run time. The compile time doesn't depend on the characteristics and it can be assumed as a constant factor.

Measuring Input size

The analysis of an algorithm will be focused on input size 'n'. It is to find the logical part of an algorithm efficiency as a function of some criteria indicating 'n' as size of the input.

Units for measuring running time

It is not possible to measure the running time by seconds, milliseconds. There should be a metric that doesn't not depend on these factors.

1. Operation counts
2. Step Counts
3. Asymptotic notation

1. Operation counts

The time and the space are the two measures for the efficiency of the algorithm. In operation counts the time is measured by counting the number of base operations or main operations.

The basic operations are defined that times for the other operations is much less than or almost proportional to the time for the basic operations

The operation count concentrate on important basic operations, multiplications, for loop or while loop where it takes considerably more time than any other operations in an algorithm

2. Step Counts

In step count we attempt to find the time spent in all parts of the program. A step is any computational unit that is independent of the selected characteristics.

There is another method in step count of an algorithms is to build table in which we list the total number of steps contributed by each statement .This is to find the number of steps in each execution of the statement and the total time each statement will be executing.

3. Asymptotic Notations

The asymptotic efficiency of algorithm is to find how the running time of a function will increases with the size of the input in the limit as the size of input increases without bound.

To simplify the asymptotic analysis there are methods.

1. Big –Oh-notation (O)
2. Omega notation (Ω)
3. Theta notation (θ)

Big –Oh-notation (O)

The big oh notation gives an upper bound on function f(n).The upper bound of f(n) indicates that the function f(n) will be worst case I doesn't consume more than this computing time.

Omega notation (Ω)

This is used to find the lower bound of f(n).The lower bound implies that below this time the algorithm cannot perform better .The algorithm will take at least this much time .And the notation is called Omega notation (Ω)

Theta notation (θ)

The theta notation can be used when the function f(n) can be bound both from above and below by the same function f(n). for some function lower bound and upper bound will be the same . And this notation is called theta (θ)

CONCLUSION

Algorithmic complexity is concerned about how fast or slow particular algorithm performs and how much memory it requires .Complexity of algorithm is a function of size of input of a given problem instance which determines how much running time/memory space is needed by the algorithm in order to run for the completion.

And above mentioned methods are to find the complexity in different levels.

ACKNOWLEDGMENT

I thank Dr. Felcy Judith (Head of Computer applications Dept. T John College) for the encouragement and technical support to make this paper successful.