Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 5, Issue. 1, January 2016, pg.91 – 95

A Comparative Study and Analysis on the Performance of the Algorithms

B.Swathi

New Horizon College of Engineering, India

baswarajuswathi@gmail.com

Abstract: Performance Analysis of algorithms has gained its importance in the computer science. Performance Analysis considers the resources used by the system such as space occupied by the algorithm and the time required to execute the algorithm. Performance Analysis of Algorithm is an important part of a computational theory, which provides theoretical estimates for the resources needed by any algorithm. For considerably large inputs Asymptotic Notations are preferred. This paper focuses on the performance analysis of the algorithms basically considers time complexity of an Algorithm. In this paper we study theoretical analysis of the time complexity and compare the results with the Experimental solution and discuss about the importance of Algorithm Engineering in this context.

Keywords: Performance Analysis, Asymptotic Notation, computational theory, Time Complexity, Algorithm Engineering

I. Introduction

The analysis of algorithms is the estimation of the amount of resources as time taken to execute an algorithm and the memory storage required by an algorithm. Parameters like instruction space in space complexity and parameters like compile time are considered as constant in this context. The running time of an algorithm is defines as a function with the time complexity and the Space complexity. These estimates provide the directions of search for efficient algorithms.

Theoretical analysis of algorithms generally includes estimation of their complexity in the asymptotic notation[1],which deals with the large input. O(big O),theta and omega notations are used. For instance, binary search takes O(log(n)), the logarithmic time. Asymptotic estimates are used because different algorithms which are intended to perform the same functionality may differ in terms of their complexities. Such type of algorithms intended for same functionality are related by a constant factor called *hidden constant*[2].

Exact measures of efficiency can be computed but requires certain considerations that particularly on implementation of the algorithm, called *model of computation*. A model of computation[3] may be defined in terms of an abstract computer. For example the binary search function with n elements with the sorted list requires at most $\log_2 n + 1$ time units.

II. Time –Analysis Theoretical

Time estimation generally deals with an executable step. The analysis corresponds the time required to perform a step. For Example, some count an addition of two numbers as one step. The same assumption may not be considered in certain contexts. For instance, if the numbers involved in a computation may be considerably large, the time considered by a single addition cannot be considered as a constant because it varies with the number. Run-time analysis is a theoretical assumption that estimates the increase in run time of an algorithm with respect to its input values. Depending on the implementation of the algorithm a program can take seconds or minutes to finish its execution.

Consider the following iterative factorial function where n is the number passed:

```
1.  While(n>0) {
2.      fact=fact*n;
3.      n--;}
4.  return fact;}
```

Considering the execution time, which is called the time complexity of an Algorithm, it only considers the program step more specifically called as implementation steps.

In general the steps other than declarative can be considered as implementation steps. Therefore our concern is to count the number of implementation steps in the function.

So, counting the total implementation steps and the number of times they occur,

Step 1:n+1 step 2:n step 3:n step 4:1

The time complexity of the function is given as f(n)=(n+1)+n+n+1=**3n+2 .**

Consider the recursive factorial function

1.if(n==1)

2. return 1;

*3. else return(n*fact(n-1));*

Step 1:n step 2:1 step 3:n-1

The time complexity for the above factorial function is f(n)=(n)+1+(n-1)=**2n**.

Examining the above analysis, the recursive Algorithms performance is highly considerable than iterative algorithms.

III. Empirical Time Analysis

The cost estimation models which can be considered primarily are uniform cost model[4] and logarithmic cost model [5].The first model assigns a constant cost where as second assigns a cost to a machine instruction proportional to number of bits.

Clock() /get time() is the function which returns the current system time. To calculate time taken by a function or an Algorithm , we can use clock() or get Time() function which is available in *time.h[6]* .We can call the get Time() function at the beginning and end of the code for which we measure time, obtain the difference between these values, and then divide by CLOCKS_PER_SEC (to get processor time).

Pseudo code for Recursive Factorial Function:

```
#include<time.h>
Clock_t start,stop;
Start=getTime();
fact(number);
Stop=getTime();

fact(number)
if(number==1)
return 1;
else
return (number*fact(number-1))
```


TABLE 1

Results

Number=4	**RUN -1**	**RUN -2**	**RUN -3**
Start time(in nano seconds)	**11265**	**11284**	**11284**
Stop time(in nano seconds)	**11288**	**11288**	**11288**
duration=(stop-start)/CLOCKS_PER_SEC	**1.000000**	**0.100000**	**0.100000**

Pseudo code for Iterative Factorial Function*:*

```
#include<time.h>                    fact(number)
Clock_t start,stop;                 while(number>0){
Start=getTime();                    fact=fact*number;
Fact(number);                       number-- }
Stop=getTime();                     return fact;
```

TABLE II

Results

Number=4	**Run-1**	**Run-2**	**Run-3**
Start time(in nano seconds)	**11265**	**11265**	**11265**
Stop time(in nano seconds)	**11290**	**11290**	**11290**
Duration=(stop-start)/CLOCK_PER_SEC	**1.000000**	**1.000000**	**1.000000**

The above results are obtained under the windows operating Environment and using the c language considering the variable number as 4.same can be observed for different values of number.

IV. Challenges

There are considerable disadvantages using an empirical approach to estimate the performance of a given set of algorithms, as Algorithms are platform independent in the sense, an Algorithm or a function can be implemented in an any language on an any computer running on an arbitrary operating system. The same is observed in the above theoretical and empirical analysis. As far the theoretical analysis is considered recursive algorithms performance is high compared to the iterative algorithms but by the practical measurement it is evident that not much deviation exists or results may be contradictory.

V. Future Enhancements

As it is observed the theoretical assumptions and the practical considerations are not comparable, definitely there is a need to analyze much about the algorithms. In this context, the Algorithm Engineering can be considered.

Algorithm Engineering [7] concerns to bridge the cross over between the theory and practical applications of algorithms. The Algorithm Engineering is mainly applied:

- The algorithm depends on the hardware parameters like instruction latencies [8],data localization where generally these cannot be applied.
- The practical algorithms are less complaint to theory.
- Theoretical considers bounds which may not be considered in practical.

Conclusion

Performance Analysis of the Algorithms is important as the unintentional use of an inefficient algorithm can impact the whole system which we build. In real-time systems, the applications considering inefficient algorithms takes too long time to run may have the chance to be outdated. The same thing can be considered in terms of storage where the applications end up to be outmoded and needs to be reorganized for use of inefficient algorithm

References

[1] Asymptotic sense available: http://mathworld.wolfram/AsymptoticNotation/
https://en.wikibooks.org/wiki/Asymptotic_Notation
[2]hidden constant available http://stackoverflow.com/questions/6242403/why-to-ignore-the-constants-in-computing-the-running-time-complexity-of-an-algorithm
[3] Model of computation available " ://cs.brown.edu/ModelofComputation"
[4] Cost estimation models available http://www.isixsigma/dictionary/costmodel/
[5] Logarthmic cost model " http://www.cse.uconn/~dqg/turing04"
[6] time.h available http://en.cppreference/w/c/chrono/time
[7] Algorithm Engineering available
http://i11www.iti.unikarlsruhe.de/_media/members/dorothea_wagner/itit.2011.9072
[8] instruction latencies available" https://gmplib.org/x86-timing"
[9] (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No.2, February-2011:'An Algorithm to Reduce the Time Complexity of Earliest Deadline First Scheduling Algorithm in Real-Time System'- Johannes Schneider, Roger Wattenhofer Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich.
[10] "Trading Bit, Message, and Time Complexity of Distributed Algorithms"
[11] "Algorithm Engineering for Parallel Computation"- David A. Bader, Bernard M. E. Moret, and Peter Sanders Departments of Electrical and Computer Engineering, and Computer Science.
[12] 'Algorithm Engineering': Matthias Müller-Hannemann, 'Bridging the Gap Between Algorithm Theory and Practice'

