Time-space analysis in Programming Languages

Jatinder Manhas*

Abstract

Time and space complexity of a program always remain an issue of greater consideration for programmers. The author in this paper has discussed the time and space complexity of a single program written in different languages. The sample program was one by one executed on a single machine with similar configuration in each case with different compiler versions. It has been observed that in different languages (C, C++, Java, C#) time taken varies. Mostly the research done for language in case of time-space trade-off is usually not bounded with fixed system configuration. In that case we calculate the time and space complexity of a particular algorithm with no system constraints. Such procedure does not give us time and space complexity of program with respect to language. The author here believes to calculate the time and space complexity of a program with respect to language.

While execution of a sample code written in different languages on fixed system configuration, for each run of sample code, CPU time is calculated and compared. It shows how the choices of language affect the time and space resource.

Keywords: Time and space Complexity, Program, Language, Time Complexity, Space Complexity

Introduction

In dealing time and space complexity we come up with the introduction of following elements:

Time and Space Complexity

In computer science, the time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input. The time complexity of an algorithm is commonly expressed using big O notation, which excludes coefficients and lower order terms. When expressed this way, the time complexity is said to be described asymptotically, i.e., as the input size goes to infinity. For example, if the time required by an algorithm on all inputs of size n is at most $5n^3 + 3n$, the asymptotic time complexity is $O(n^3)$.[URL: http://hopl.murdoch.edu.au/]

Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Thus the amount of time taken and the number of elementary operations performed by the algorithm differ by at most a constant factor.[URL: http://hopl.murdoch.edu.au/]

Since an algorithm's performance time may vary with different

**Department of Computer Science and IT (BC), University of Jammu, (J&K), India.*

Authors are grateful to the anonymous referees for comments and suggestions for improving the text and contents of the paper. Authors alone are responsible for any errors/mistakes still remaining in the paper.

inputs of the same size, one commonly uses the worst-case time complexity of an algorithm, denoted as *T(n)*, which is defined as the maximum amount of time taken on any input of size n. Time complexities are classified by the nature of the function *T(n)*. For instance, an algorithm with *T (n)* = O *(n)* is called a linear time algorithm, and an algorithm with T(n) = $O(2^n)$ is said to be an exponential time algorithm.[1]

The author in this paper slightly differ form the general definition of time complexity. Author while researching take the time complexity not as the number of elementary operations performed on any system, but as the CPU time taken by the set of instructions on some specific system. The reason for considering this is that, we need to compare the time and space trade-off of program with respect to different languages, not from different systems.

Reasons for measuring the time complexity

1. If the program is to run with given time constraint we must need to calculate its time complexity.
2. To know in advance how much time is required by program for execution.
3. It is used to calculate the overall efficiency of a program.
4. Calculation of time complexity is needed by the program while writing larger codes.
5. It can be used to estimate the largest problem that the programmer can solve.

Space Complexity

It is the amount of memory which is needed by the algorithm (program) to run to completion. We can measure the space by finding out that how much memory will be consumed by the instructions and by the variables used.

E.g.:-Suppose we want to add two integer numbers. To solve this problem we have following two algorithms:
Both algorithms will produce the same result. But first takes 6

Algorithm 1:	Algorithm 2:
Step 1- Input A.	Step 1- Input A.
Step 2- Input B.	Step 2- Input B.
Step 3- Set C: = A+ B.	Step 3- Write: 'Sum is ', A+B.
Step 4- Write: 'Sum is ', C.	Step 4- Exit.
Step 5- Exit.	

bytes and second takes 4 bytes (2 bytes for each integer). And first has more instructions than the second one. So we will choose the second one as it takes less space than the first one.

While working with different languages space complexity usually does not vary, as the number of instruction executed remain fixed for each language.

Reasons for measuring the space complexity

1. If the program is to be run on multiuser system, then we will have to specify the amount of memory to be allocated to the program.
2. To know in advance whether enough memory is available for the program or not.
3. There can be several solutions for the problem with different space requirements and we have to choose, one which takes least space among them.
4. It can be used to estimate the size of the largest problem that a program can solve.

Programming language

According to oxford dictionary, Programming language is "The process of writing computer programs"

History

Programming languages as understood today became possible when the work of Alan Turing and Alonzo Church in the 1930s and 40s provided formal mathematical means to express computational procedures. In a dynamic and evolving field, there is no single standard for classifying programming languages. In fact, dozens of categories exist. One of the most fundamental way by which programming languages are characterized is by programming paradigm. A programming paradigm provides the programmer's view of code execution. The most influential paradigms are examined in the next three sections, in approximate chronological order. Each of these paradigms represents a mature worldview, with enormous amounts of research and effort expended in their development. A given language is not limited to use of a single paradigm. Java, for example, supports elements of both procedural and object-oriented programming, and it can be used in a concurrent, event-driven way. Programming paradigms continue to grow and evolve, as new generations of hardware and software present new opportunities and challenges for software developers. [URL: 2, 3, 4, 5]

Procedural Programming Languages

Procedural programming specifies a list of operations that the program must complete to reach the desired state. This is one of the simple programming paradigms, where a program is represented much like a cook book recipe. Each program has a starting state, a list of operations to complete, and an ending point. This approach is also known as imperative programming. Integral to the idea of procedural programming is the concept of a procedure call. Procedures, also known as functions, subroutines, or methods, are small sections of code that perform a particular function. A procedure is effectively a list of computations to be carried out. Procedural programming can be compared to unstructured programming, where all of the code resides in a single large block. By splitting the programmatic tasks into small pieces, procedural programming allows a section of code to be re-used in the program without making multiple copies. It also makes it easier for programmers to understand and maintain program structure. [URL: 2,3,4,5]

The use of functions to implement the procedural programming greatly affects the time resource as compared to unstructured programming. By dividing the program into modules help the program to shorten the code. The functions in the program reduce the space of code effectively. When functions are used in programming, space complexity of program get reduced but time complexity is not much effected.

Two of the most popular procedural programming languages are FORTRAN and BASIC.

Structured Programming Languages

Structured programming is a special type of procedural programming. It provides additional tools to manage the problems that larger programs were creating. Structured programming requires that programmers break program structure into small pieces of code that are easily understood. It also frowns upon the use of global variables and instead uses variables local to each subroutine. One of the well-known features of structural programming is that it does not allow the use of the GOTO statement. It is often associated with a "top-down" approach to design. [URL: 2, 3, 4, 5]

In structured programming, time and space complexity is effected like as in procedural programming. Structured programming is almost same as procedural programming, in case of time and space trade-off.

The most popular structured programming languages include C, Ada, and Pascal.

Object-Oriented Programming Languages

Object-oriented programming is one the newest and most powerful paradigms. In object-oriented programs, the designer specifies both the data structures and the types of operations that can be applied to those data structures. This pairing of a piece of data with the operations that can be performed on it is known as an object. A program thus becomes a collection of cooperating objects, rather than a list of instructions. Objects can store state information and interact with other objects, but generally each object has a distinct, limited role.

There are several key concepts in object-oriented programming (OOP). A class is a template or prototype from which objects are created, so it describes a collection of variables and methods (which is what functions are called in OOP). New classes can be derived from a parent class. These derived classes inherit the attributes and behaviour of the parentless (inheritance), but they can also be extended with new data structures and methods. The list of available methods of an object represents all the possible interactions it can have with external objects, which means that it is a concise specification of what the object does. One of the key characteristic of OOP is encapsulation, which refers to how the implementation details of a particular class are hidden from all objects outside of the class. Programmers specify what information in an object can be shared with other objects. A final attribute of object oriented programming languages is polymorphism. Polymorphism means that objects of different types can receive the same message and respond in different ways. Polymorphism is often applied to derived classes, which replace the methods of the parent class with different behaviours. Polymorphism and inheritance together make OOP flexible and easy to extend. [2,3,4,5]

Features like polymorphism, inheritance is observed to greatly impact the space complexity. Such features enhance and implement the reusability feature in object oriented programming as compared to previously developed languages.

Polymorphism, when same function is being used to perform different task, it helps programmer to shorten the code length. Inheritance on the other hand also made the program to re-use the code. OOP programming having the features of structured, procedural, and OOP flexibility impact the time and space complexity eventually.

The most popular object-oriented programming languages include Java, Visual Basic, C#, C++, and Python.

Other Paradigms

Concurrent programming provides for multiple computations running at once. This often means support for multiple threads of program execution. For example, one thread might monitor the mouse and keyboard for user input, although another thread performs database accesses.[URL: 2,3,4,5,6] Multiple tasks being done simultaneously reduces the execution time of program eventually. Reduction of execution time of program reduces the time complexity.

Popular concurrent languages include Ada and Java.

Functional programming languages define subroutines and programs as mathematical functions. These languages are most frequently used in various types of mathematical problem solving. Such functions can implement the large set of instruction into one or two lines of code, helping to reduce the time-space complexity of code.

Popular functional languages include LISP and Mathematica.

Event-driven programming is a paradigm where the program flow is determined by user actions. This is in contrast to batch programming, where flow is determined when the program is written. Event-driven programming is used for interactive programs, such as word processors and spreadsheets. The program usually has an event loop, which repeatedly checks for interesting events, such as a key press or mouse movement. Events cause the execution of trigger functions, which process the events.

Relationship of Time/space complexity with Programming Language

Different languages are designed differently, with certain aim in mind.

Ritchie: The point of C (as distinct from its immediate predecessor B) was to make language that was designed with word-oriented machines in mind and adapt it to the newer hardware that became available, specifically the PDP-11. It did not take long to learn that the kind of things introduced early on would also make it adaptable to much different machines [URL: http://www.gotw.ca/publications/c_family_interview.htm]

Stroustrup: The initial aim for C++ was a language where one could write programs that were as elegant as Simula programs, yet as efficient as C programs. The projects being considered at the time had to do with distributing operating systems functions and applications across a number of computers in a local-area network programmers to consider both the complexity of the total system and the efficiency of the lowest-level facilities. The object-oriented facilities from Simula helped with the former, and the systems-programming facilities of C with the latter.

[URL: http://www.gotw.ca/publications/c_family_interview.htm]

Gosling: Like C, Java's goals were really clear; programmer being able to build relatively specific kinds of software, and designers were really going for very distributed, very reliable, interacting-with-people kinds of software -- ones where things like reliability mattered a huge amount, things like security mattered a huge amount, and they all mattered enough that there was a willingness to take a performance hit for them. [URL: http://www.gotw.ca/ publications/c_family_interview.htm]

For example, one of the differences between Java and C is that Java has real arrays and the arrays get bounds-checked, and that turns out to be important for both reliability and security. If you look at the history of security breaches on the Internet, some really large fraction of them are exploiting buffer overflows, statically allocated arrays that people just go over the end of. If you look at logs of bug-fixing that people go through, it ends up being that memory integrity issues are a huge fraction of where the problems come from.

Variation in language design and goal lead to different complexity for each language. That means same set of instruction are executed differently in different languages. Development in programming languages also enhances its efficiency in various aspects. Newly developed languages are much more efficient than previous structures. Every language now created is developed, keeping in mind the previous limitations. The aspects of complexity are becoming a major issue in developing new languages. It has been the major goal of developers to decrease the time and space complexity of language as much as possible.

Methodology

The author in this paper took four different languages(C, C++, Java & C#) and executed the sample code on each platform by selecting a common machine with same configuration in each case.

Testing Environment Specification
Processor:
Intel(R) Core(TM) i3 CPU M 330 @2.13GHz 2.13 GHz
Installed Memory: 3.00GB (2.87GB Usable)
System type: 32-bit Operating System.

Sample Code

C/C++Code

```
beg=clock();
for(c=0;c<100;c++)

{
for(d=0;d<100;d++)

{
for(k=0;k<100;k++)
{
sum=sum+(a[c][k]*b[k][d]);
}
mul[c][d]=sum;
sum=0;
}
}
end=clock();
time=(end-beg)/CLOCKS_PER_SEC;
```

Java code

```
beg=System.nanoTime();
for ( c = 0 ; c < 100 ; c++ )
{
for ( d = 0 ; d < 100 ; d++ )
{
for ( k = 0 ; k < 100 ; k++ )
{
sum = sum + first[c][k]*second[k][d];
}


multiply[c][d] = sum;
sum = 0;
}
}

end=System.nanoTime();
time=end-beg;
System.out.println(time);
```

C# Code

```
Stopwatch sw = new Stopwatch();
sw.Start();

Console.WriteLine("Multiplication of two matrix are:");

for (int i = 0; i < 100; i++)
{
for (int j = 0; j < 100; j++)
{

for (int k = 0; k < 100; k++)
{
sum =sum+matrix1[i, k] * matrix2[k, j];
}
multi[i, j] = sum;
sum = 0;

}
}

sw.Stop();

TimeSpan ts = sw.Elapsed;
string elast =
String.Format("{0:00}:{1:00}",ts.Seconds,ts.Milliseconds);

Console.WriteLine("eecution Time"+elast);
```

Analysis

Above sample code represents matrix multiplication in C, C++, java and C#. The size of array is varied for each execution to calculate the time complexity i.e. how much CPU time it took on a system. The author changes the space complexity of program and analyse the variation of execution time for each language. In all the above code there is a core piece of code, where the multiplication of matrix takes place. This piece of code is almost same for all the languages. We set the timer at the start and end for that loop and calculate how much time it takes to execute it. The same procedure is repeated for all of the above languages and is executed on system with similar specifications. The above code is executed and the results were calculated as per the requirement given in the table Table_1 with different condition like array size being 100*100, 80*80, and 60*60.

Figure 1 and Figure 2 gives the graphical representation of execution time variation of sample code on different languages. From the figures it is clear that the C# took less time than C/C++ and Java.

Table 1

Sr. No.	Language	Name of compiler version	Time Taken in sec		
			[100][100]	[80][80]	[60][60]
1	C	Borland	0.054945	0.054945	0.054945
2	C++	Borland	0.054945	0.054945	0.054945
3	java	Jdk1.7.0	0.005780386	0.0207	0.0084
4	C#		0.00059375	0.00037625	0.00013

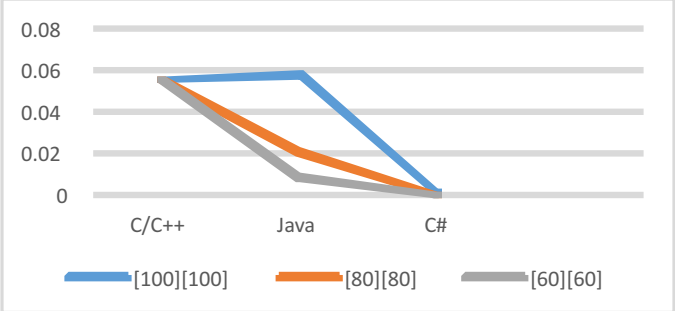

Figure 1: Execution time variation of sample code

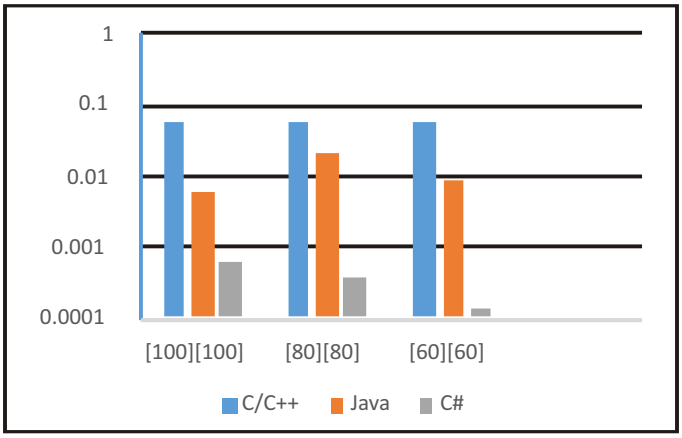

Figure 2: Execution time variation of sample code

Result and discussion

The above analysis gives us the brief knowledge about how time complexity vary for different languages. The time given in the table 1 is CPU usage time. From the table we conclude that C# (visual studio 2010) seems to be most time efficient language. It has been observed by the author that in future if author has to choose to write the code with time and space constraint, C# will be the best option to complete the task given. The above experiment shows that C# is the most time efficient language out of C, C++ and java. The above result gives us the detailed analysis of time complexity with respect to languages. Making algorithm time and space efficient will not solve the entire problem, we should also think about the time efficiency of languages to.

Future Scope

More and More languages are developed every day. It is possible in future that we develop such languages that are extremely efficient, so we need not to worry about algorithmic efficiency. As we seen in our results how time efficiency vary from C/C++ to C#.

Conclusion

Our experiment of running the same sample code written in four different languages (C, C++, Java, C#) on different compiler help the people in academic, research and software development to choose the best language out of the available when the parameter like time and space are to be taken into consideration.

References

1. *https://en.wikipedia.org/wiki/time_complexity*
2. *David G, Suresh J. (1990), Programming Linguistics, The MIT Press.*
3. *Shriram K, Programming Languages: Application and Interpretation. Available online at: http://www.cs.brown.edu/~sk/Publications/Books/ProgLangs/websites*
4. *Computer Languages History graphical chart, http://www.levenez.com/lang/history.html*
5. *History of Computer Languages, http://hopl.murdoch.edu.au/*
6. *An Introduction to Programming Languages, http://www.acooke.org/andrew/writing/lang.html*
7. *http://www.gotw.ca/publications/c_family_interview.htm*