Chennai and Dr.MGR University Second International Conference on Sustainable Energy and Intelligent System (SEISCON 2011),
Dr. M.G.R. University, Maduravoyal, Chennai, Tamil Nadu, India. July. 20-22, 2011.

An Abstract to Calculate Big O Factors of Time and Space Complexity of Machine Code

S. Gayathri Devi*, K. Selvam†, Dr. S. P. Rajagopalan†

* Department of Mathematics, Dr.M.G.R.University,Chennai, India
†Department of Computer Applications, Dr.M.G.R.University,Chennai, India
gayatrisomu@gmail.com selwam2000@gmail.com sasirekharaj@yahoo.co.in

Keywords: Time complexity, Space complexity, BigO, f(n), cg(n), O(G(n)).

Abstract

Algorithms are generally written for solving some problems or mechanism through machines, the algorithms may be several in numbers, further to these the efficiency of the produced algorithms for the said issue need to be quantified: the factors which are to be quantified are time complexity, space complexity, administrative cost and faster implementation etc..,..One of the effective methods for studying the efficiency of algorithms is Big-O notations, though the Big-O notation is containing mathematical functions and their comparison step by step, it illustrates the methodology of measuring the complexity facto. The output is always expected as a smooth line or curve with a smaller and static slope.

1 Introduction

Definition: A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Informally, saying some equation f(n) = O(g(n)) means it is less than some constant multiple of g(n). The notation is read, "f of n is big oh of g of n".

Formal Definition: f(n) = O(g(n)) means there are positive constants c and k, such that $0 \le f(n) \le cg(n)$ for all $n \ge k$. The values of c and k must be fixed for the function f and must not depend on n.

2 Mathematical Base

In mathematics, computer science, and related fields, big-O notation is often denoted by a big cryptographic O, describes the limiting behavior of the function when the argument tends towards a particular value or infinity, usually in terms of simpler functions. BigO notation characterizes functions according to their growth rates: different functions with the same growth rate may be represented using the same O notation. this notation is now frequently used in the analysis of algorithms to describe an algorithm's usage of computational resources: the worst case or average case running time or memory usage of an algorithm is often expressed as a function of the length of its input using big O notation. This allows algorithm designers to predict the behavior of their algorithms and to determine which of multiple algorithms to use, in a way that is independent of computer architecture or clock rate. Because bigO notation discards multiplicative constants on the running time, and ignores efficiency for low input sizes, it does not always reveal the fastest algorithm in practice or for practically-sized data sets, but the approach is still very effective for comparing the scalability of various algorithms as input sizes become large. A description of a function in terms of bigO notation usually provides only an upper bound on the growth rate of the function. Associated with bigO notation there are several related notations exist.

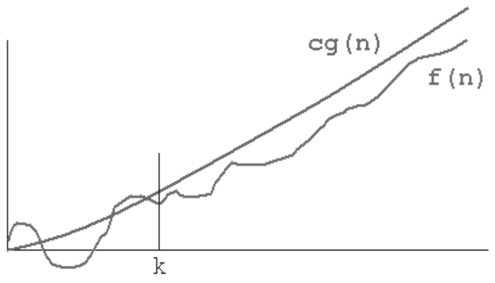

Figure 1: Big-O Notation's output of an Algorithm

Let f(n) and g(n) be two functions defined on some subset of the real numbers. One writes

$$f(n) = O(g(n)) asn \to \infty \tag{1}$$

if and only if, for sufficiently large values of n, f(n) is at most a constant multiplied by g(n) in absolute value. That is, f(n) = O(g(n)) if and only if there exists a positive real number M and a real number n0 such that

$$|f(n)| \le M|g(n)| \quad for\ all\ n \rangle n_o \tag{2}$$

In many contexts, the assumption that we are interested in the growth rate as the variable n goes to infinity is left unstated, and one writes more simply that f(n) = O(g(n)). The notation can also be used to describe the behavior of f near some real number a (often, a = 0): we say

$$f(n) = O(g(n)) \quad as\ n\ \to a \tag{3}$$

if and only if there exist positive numbers δ and M such that

$$|f(n)| \le M|g(n)| \quad for \quad |n-a| \langle \delta \tag{4}$$

If g(n) is non-zero for values of n sufficiently close to a, both of these definitions can be unified using the limit superior:

$$f(n) = O(g(n)) \quad as \quad n \to a \tag{5}$$

if and only if

$$\limsup_{n\to a} \left|\frac{f(n)}{g(n)}\right| \langle \infty \tag{6}$$

In typical usage, the formal definition of O notation is not used directly, the O notation for a function f(n) is derived by the following simplification rules:

- If f(n) is a sum of several terms, the one with the largest growth rate is kept, and all others omitted.
- If f(n) is a product of several factors, any constants (terms in the product that do not depend on n) are omitted.

For example, let f(n) = 6n4 – 2n3 + 5, and suppose we wish to simplify this function, using O notation, to describe its growth rate as n approaches infinity. This function is the sum of three terms: 6n4, −2n3, and 5. Of these three terms, the one with the highest growth rate is the one with the largest exponent as a function of n, namely 6n4. Now one may apply the second rule: 6n4 is a product of 6 and n4 in which the first factor does not depend on n. Omitting this factor results in the simplified form n4. Thus, we say that f(n) is a big-oh of (n4) or mathematically we can write f(n) = O(n4). One may confirm this calculation using the formal definition: let f(n) = 6n4 – 2n3 + 5 and g(n) = n4. Applying the formal definition from above, the statement that f(n) = O(n4) is equivalent to its expansion,

$$|f(n)| \le M|g(n)| \tag{7}$$

for some suitable choice of x0 and M and for all x > x0. To prove this, let x0 = 1 and M = 13. Then, for all x > x0:

$$\begin{aligned} |6n^4 - 2n^3 + 5| &\le 6n^4 + |2n^3| + 5 \\ &\le 6n^4 + 2n^4 + 5n^4 \\ &\le 13n^4 \quad \le 13|n^4| \end{aligned}$$

So

$$|6n^4 - 2n^3 + 5| \le 13|n^4| \tag{8}$$

BigO notation has two main areas of application. They are in mathematics, and computer science, In computer science it is useful in the analysis of algorithms. In both applications, the function g(n) appearing within the O(...) is typically chosen to be as simple as possible, omitting constant factors and lower order terms. There are two formally close, but noticeably different, usages of this notation. This distinction is only in application and not in principle, however—the formal definition for the "bigO" is the same for both cases, only with different limits for the function argument.

The statement *f(n) = O(n!)* is sometimes weakened to *f(n) = O(n^n)* to derive simpler formulas for asymptotic complexity. For any k > 0 and c > 0, $O(n^c(\log n)^k)$ is a subset of $O(n^{c+\varepsilon})$ for any $\varepsilon > 0$, so may be considered as a polynomial with some bigger order.

3 Application Thoughts

BigO notation is used in Computer Science to describe the performance or complexity of an algorithm. BigO specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

As a programmer and a mathematician (Program analyst) It is found that, the best way to understand BigO thoroughly is to produce some examples in code. So, below are some common orders of growth along with descriptions and examples where possible.

Figure 1: Big-O Notation's output of different 'n' of the Examining Algorithm

O(1)

O(1) describes an algorithm that will always execute in the same time (or space) regardless of the size of the input data set.

```
bool IsFirstElementNull(String[] strings)
  {
      if(strings[0] == null)
        {
            return true;
        }
      return false;
  }
```

O(n)

O(n) describes an algorithm whose performance will grow linearly and in direct proportion to the size of the input data set. The example below also demonstrates how BigO favours the worst-case performance scenario; a matching string could be found during any iteration of the for loop and the function would return early, but BigO notation will always assume the upper limit where the algorithm will perform the maximum number of iterations.

```
bool ContainsValue(String[] strings, String value)
  {
      for(int i = 0; i < strings.Length; i++)
      {
        if(strings[i] == value)
        {
          return true;
        }
      }
      return false;
  }
```

O(n2)

O(n2) represents an algorithm whose performance is directly proportional to the square of the size of the input data set. This is common with algorithms that involve nested iterations over the data set. Deeper nested iterations will result in O(n3), O(n4) etc.

```
bool ContainsDuplicates(String[] strings)
{
  for(int i = 0; i < strings.Length; i++)
  {
    for(int j = 0; j < strings.Length; j++)
    {
      if(i == j) // Don't compare with self
```

```
        {
          continue;
        }
      if(strings[i] == strings[j])
        {
          return true;
        }
      }
    }
  return false;
  }
```

O(2n)

O(2n) denotes an algorithm whose growth will double with each additional element in the input data set. The execution time of an O(2n) function will quickly become very large.

- As mentioned and published in the research article ‘K.Selvam, B.Poorna,’ “IMAGE EXTRACTION BY QUINE MC CLUSKEY’S TABULAR METHOD”, in proceeding of IET-UK International Conference on Information and Communication Technology in Electrical Sciences (ICTES 2007),PP 754-757,20-22 Dec.2007.
- And also the same above mentioned article was referred by the Authors ‘Hamed Sheidaeian, Behrouz Zolfaghari, Saadat Pour Mozaffari’ in the International conference proceeding titled “SPIDERS: Swift Prime Implicant Derivation through Exhaustive Rotation and Sort”, in IEEE proceeding of 2010 International Conference on Networking and Information Technology $ 978-1-4244-7578-0 / $ 26.00©2010 IEEE.

The prime implicants and their minimization begins at iterative algorithm for implicant derivation which is based on bitwise rotation and sort in each iteration. This idea is inspired from the fact that minterms which differ only in the least significant variable appear in successive location in the minterm table shows this adjacency. Minterm differing in the variable Z (The least significant variable) occupy adjacent entries in the table. It is also seen that each implicant has been derived from an odd-numbered minterm with its successor. Now consider X’YZ and XYZ minterms. These two minterms differ in the value of X. They are not in adjacent entries but they will obviously be placed in successive entries if it is moved on every minterm tow bits to the successive position

If we number n variables from right to left (beginning at zero), every couple of minterms differing only in the ith variable ($0 \le i \le n-1$) will be adjusted after I consecutive recognitions . This property has been introduced in the proposed recognition model of fingerprint and face image recognition methods. The algorithm is also supported by ”Index sorting” algorithm with time complexity of order O(n) which is also taken for BigO Notation.

Let us simplify the algorithm by an example. Consider the following function.

$$F(x,y,z,w) = \sum(0,1,2,4,5,8,9,10,12,13,15) \tag{9}$$

= x’y’z’w’+x’y’z’w+x’y’zw’+x’yz’w’+x’yz’w+xy’z’w’ +xy’z’w+xy’zw’+xyz’w’+xyz’w+xyzw.

Prime implicants = z’+y’zw’+xyzw.

4 Performance Evalution

The analytical model will be derived in order to evaluate the performance of the”fingerprint and face image recognition algorithm”. Let us begin the derivation of the recognition by calculating the total number of implicants with different sizes (3*3 or 4*4 or 5*5 etc…..) This can be selected by the user as m-variable minterms. It is illustrated in experiment part that each implicant discards a number of variable(This can be equal to 0).The number of implicant that discard $0 \le m \le n$ variables can be calculated as follows:

$$NI_m^n = \binom{n}{m}.2^{n-m} \tag{10}$$

In the above equation, $\binom{n}{m}$ gives the total number of combination including m variable which can be selected to be discarded among n variable. 2^{n-m} shows the number of different sequences that can be made by the remaining n-m variables

Now we can calculate the total number of implicants which can be derived from n variable minterms.

$$NI^n = \sum_{m=0}^{n} NI_m^n = \sum_{m=0}^{n} \binom{n}{m}.2^{n-m} \tag{11}$$

The above summation can be simplified using the binomial extension as follows.

$$NI^n = \sum_{m=0}^{n} \binom{n}{m}.1^n.2^{n-m} = (1+2)^n = 3^n \tag{12}$$

The above equation 12 states that there is a total number of 3n implicants with different sizes which can be constructed from n variable minterms. In other words, this equation gives the total number of implicants which can appear in the simplified forms of all possible n variable logic functions.

The probability that the number of minterms in an n variable logic function is equal to m is given by equation 13.

$$PM_m^n = \frac{\binom{n}{m}}{\sum_{i=0}^{n}\binom{n}{i}} = \frac{\frac{n!}{(n-m)!m!}}{2^n} = \frac{n!}{2^n.(n-m)!m!} \tag{13}$$

The equation 14 gives the probability that k variables are discarded by the largest implicant in a logic function including m minterms ($0 \le k \le [\log_2 m]$).

$$PI_k^m = \frac{\binom{n}{k}.2^{n-k}}{\sum_{i=0}^{[Log_2 m]}\binom{n}{i}.2^{n-i}} \tag{14}$$

The equation 15 gives the probability that one of the k variables discarded by the largest implicant is the least significant variable.

$$PI_k^{'m} = \frac{\binom{n}{k}.2^{n-k}}{\sum_{i=0}^{[Log_2 m]}\binom{n}{i}.2^{n-i}} \tag{15}$$

The probability given by equation 15 is equivalent to the probability that an implicant is detected in a given iteration of the proposed recognition algorithm which discards k of n variables.

Now we can calculate the probability that r such implicant are detected in a certain iteration of the algorithm. This probability is given by the following equation.

$$PI_{r,k}^n = \prod_{i=0}^{r-1} PI_k^{'m-ik} \tag{16}$$

Equation 16 gives the probability that r.2k minterms are omitted from the list in each iteration of the said algorithm. The omission of these minterm causes the next iteration to run faster and this is the main advantage of the algorithm.

The number of store operations (required by the index sorting) in each iteration of this algorithm is equal to the number of move on in each iteration. Therefore in order to determine the time complexity of the proposed algorithm, we can optionally count the number of store operation or the number of move on operations. We as the user choose to calculate the number of move on operation. Each implicant which discards V variable, takes V+1 steps in which a list of minterms are moved on. The length of the list is 2n in the first step, 2n-1 in the second step and so on. Thus the number of move required by such an implicant can be obtained from the following equation.

$$R_v^n = \sum_{j=0}^{v} 2^{n-j} = \frac{2^{n-v}(2^{v+1}-1)}{2-1} = 2^{n+1} - 2^{n-v} \qquad (17)$$

5 Results and Discussion

The worst case for the proposed algorithm occurs when the identical numbers of variables are discarded by all implicants, especially when each implicant consists of a single minterm. On the other hand, there cannot be more than 2^{n-1} minterm in a function of n variables which cannot construct any implicant and discard any variables. Thus the worst case for algorithm occurs when the function to be minimized includes 2^{n-1} minterm each requiring one operation. This means that the worst case time complexity of this algorithm is of order $O(2^{n-1})$. The worst case time complexity of the traditional Quine Mc Cluskeys algorithm is of order $O((n-1).2^{n-1})$ because for each single minterm, the existence of n-1 other minterm which are different with the considered minterm in a single variable must be checked.

The above calculations show that the Fingerprint and face image recognition algorithm reduces the time complexity of the logic function minimization from $O((n-1).2^{n-1})$ to $O(2^{n-1})$ The worst case time complexity of the algorithm is of the order $(n.2^n)$.

6 Conclusion

The method of implementation is found effective in the estimation of the efficiency of an algorithm and helps to fine tune the properties of the algorithm for future enhancement. The knowledge of bigO notation reduces the administrative cost and can be considered as a metric for software projects management and implementation. This method will support when testing any software or algorithm, for measuring their performance and work load, which can effectively be handled by the machine code or applications. This study assists the project development team to produce any personal purposed products carrying "n" subroutines of a routine. This is one of the newer approaches in automation of applications.

References

1. Selvam.K, Poorna.B,' "IMAGE EXTRACTION BY QUINE MC CLUSKEY'S TABULAR METHOD", in proceeding of IET-UK International Conference on Information and Communication Technology in Electrical Sciences (ICTES 2007), PP 754-757, 20-22 Dec.2007
2. Hamed Sheidaeian, Behrouz Zolfaghari, Saadat Pour Mozaffari' in "SPIDERS: Swift Prime Implicant Derivation through Exhaustive Rotation and Sort", in IEEE proceeding of 2010 International Conference on Networking and Information Technology $ 978-1-4244-7578-0 / $ 26.00©2010 IEEE
3. Davide Maltoni, Davio Maie,A.K.Jain and Salil Prabakar,2003.Hand Book of Fingerprint Recognition. Springer Professional Computing,pp:176-213
4. Earl Gose,Richard Johnsonbaugh and steveJost,2000.Pattern recognition and Image Analysis,pp:231-247.
5. Gonzalez,C.and R.E. Woods,2002.Digital Image Processing (2nd edn.), Prentice-Hall, Englewood Cliffs,NJ,pp:36-113.
6. Jain, A.K. (1989) Fundamentals of digital image processing. Prentice-Hall International Inc.
7. Moris Mano,2002.Digital Computer Fundamentals.Prentice Hall,pp:72-112.
8. Nalini Ratha and Ruud Bolle, 2003. Automatic Fingerprint Recognition System Springer, pp:67-112.
9. Woods, R.E. and Gonzalez, R.C.(1981)Real-time digital Image enhancement. Proc. IEEE Vol.69,No.5, pp: 643-654.
10. Wilson R. and Spann,M(1988).Image segmentation and uncertainty. Research studies Press Ltd, John Wiley & Sons Inc.
11. Weszka, J.S (1978). A survey of threshold selection techniques. Computing Vision Graphics Image Process, Vol.7, pp: 259-265.
12. Jain,A.K.,R.Duin and J.Mao,2000. Statistical Pattern Analysis and Machine Intelligence,22:4-37.