Shri Vile Parle Kelavani Mandal’s

Institute Of Technology, Dhule

Department of Information Technology

Subject : Design and Analysis of Algorithm lab

Assignment Title : BIG O NOTATION

Student Name	**Roll No**	**PRN**	**Class**
Bagul Virendra Kashinath	05	2254491246005	SY

Title : BIG O NOTATION

A) Define Big O notation? What is total time complexity of following code?

```
int a, b, c, d, i;
{
    for (i = 0; i < 11; i++)
         {
              a = a + b;
         }
    d = c + a;
}
```

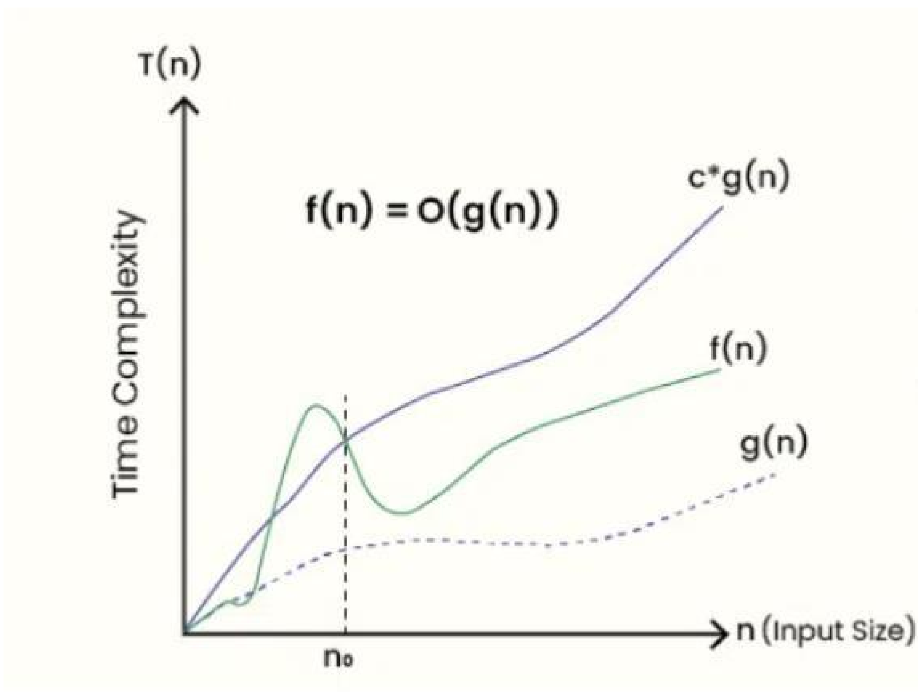
Ans: It is the most commonly used notation to measured the performance of any algorithm by defining its order growth. Big O notation is a powerful tool used to described the time complexity and space complexity.

If f(n) and g(n) are the two functions defines for the positive integers, then f(n) is O(g(n)) oh of g(n) or f(n) is on the order of g(n) if there exist constants c such that f(n)<= c*g(n).

These implies that f(n) does not grow faster than g(n), or g(n) is an upper bound on the function f(n).

Big O notation is a mathematical notation used to described the worst-case time complexity or efficiency of an algorithm. It provides a way to compare the performance of different algorithm and to predict how they will behave as the input size increases.

It is commonly referred to as “order of” is a way to express the upper bound of an algorithm’s time complexity, since it analyses the worst-case situation of algorithm. It provides an upper limit on the time taken by an algorithm in terms of the input. It’s denoted as O(f(n)) where f(n) is a function that represents time complexity of an algorithm of given input size n.



Total time complexity of following code?

```
int a, b, c, d, i;
{       for (i = 0; i < 11; i++)
        {
                a = a + b;
        }
d = c + a;
 }
```

1) Initializing variable a, b, c, d, and i has a constant time complexity which we denotes as O(1).

2) The loop iterates from 0 to 10 (i<11) and perform the constant time operation inside (a = a + b). Since a loop iterate a constant number of times (11 times). The time complexity of loop is O(1).

3) Assignment outside the loop: After the loop there is an assignment 'd = c + a', which is a constant time operation denoted as O(1).

Combining these steps, we get the total time complexity of code as:

Initialize(O(1)) + Loop(O(11)) + Assignment(O(1))

= O(1) + O(11) + O(1) = O(13)

Therefore, the total time complexity of the given code is O(13), which simplifies to O(1) in terms of big O notation. This means that the time complexity of the code is constant, regardless of the input size.

B] Define algorithm. What is the need of an algorithm analysis? Which factors affects the runtime of algorithm?

Ans: An algorithm is a step-by-step procedure or set of instructions designed to perform a specific task or solve a problem. In computer science, algorithms are fundamental to computer programs because they define how data is manipulated and processed.

Algorithm Analysis:

Algorithm analysis is the process of evaluating and understanding the efficiency and performance characteristics of an algorithm. It involves studying how an algorithm performs in terms of its time complexity (how long it takes to run) and space complexity (how much memory it uses).

The need for algorithm analysis arises from several important factors:

1) Efficiency: Efficient algorithms can save computational resources such as time and memory. By analyzing algorithms, we can identify which ones are more efficient for solving a particular problem.

2) Performance Prediction: Algorithm analysis helps predict how an algorithm will behave as the input size increases. This is crucial for understanding scalability and ensuring that an algorithm can handle large datasets efficiently.

3) Comparison: Algorithm analysis allows us to compare different algorithms for the same problem and determine which one is more suitable based on its performance characteristics.

4) Optimization: By understanding the performance bottlenecks of an algorithm, we can optimize it to improve its efficiency, reduce resource usage, and enhance overall system performance.

Several factors affect the runtime of an algorithm:

1) Input Size: The size of the input data greatly influences the runtime of an algorithm. Generally, algorithms with larger input sizes require more time to process.

2) Algorithm Design: The design of the algorithm itself plays a crucial role. Some algorithms are inherently more efficient for certain types of problems than others. For example, a well-designed sorting algorithm can have significantly better performance than a less efficient one for sorting large datasets.

3) Data Structures: The choice of data structures used by an algorithm can impact its runtime. Different data structures have different time complexities for operations such as insertion, deletion, and search.

4) Resource Constraints: Factors such as available memory, processor speed, and system architecture can also affect the runtime of an algorithm. An algorithm that requires a lot of memory may run slower on a system with limited RAM compared to a system with ample memory.

C] Solve the following recurrence relation using master method . T(n) = 2T(n/2) + cn

Ans: Given recurrence relation: T(n)=2T(n/2) + cn

T(n) = aT(n/b) + f(n) ,   where a >= 1, b > 1

a = 2 , b= 2 , f(n) = cn

T(n) = 2T(n/2) + cn

Since ( ) = is asymptotically equal to n^logba=n,
we fall into case 2 of the Master Theorem.

$T(n)=\Theta(n$^log$ba\log n)$

$T(n)=\Theta(n\log n)$

So, in this case, the solution to the recurrence relation $T(n)=2T(n/2)+cn$ using Master Theorem Case 2 is $T(n)=\Theta(n\log n)$. This means that the time complexity of the algorithm described by this recurrence relation is bounded asymptotically by $\Theta(n\log n)$, indicating that the algorithm's time complexity grows at the rate of $n\log n$ as n increases.

- *Let's verify the solution (T(n) = Theta(n log n)) using the substitution method.*

We assume that (T(n) = an log n + bn) and then substitute it back into the original recurrence relation.

*1. **Assumption:***
We assume that (T(n) = an log n + bn).

*2. **Substitution:***

[T(n) = 2T(n/2) + cn]
[an log n + bn = 2[a(n/2) log(n/2) + b(n/2)] + cn]

3. **Simplify:**
[an log n + bn = 2[a(n/2) log(n/2) + b(n/2)] + cn]
[an log n + bn = 2[a(n/2) log n - alog 2 + b(n/2)] + cn]
[an log n + bn = 2[a(n/2) log n - alog 2 + bn/2] + cn]
[an log n + bn = a(n log n - alog 2 + n/2 log n) + bn - alog 2 + cn]

4. **Match Coefficients:**
- Coefficients of (n log n): (a = 2a), so (a = 2).
- Coefficients of (n): (b = b/2 - alog 2 + c), so (b/2 - alog 2 + c = b), and (c = b2 - alog 2).

5. **Verify Conditions:**
- We need to verify that (c) is positive to ensure that our assumption of(T(n)) is correct.

c = b/2 – alog 2 = b/2 - 2log 2
Since b is a constant and log 2 is a constant, c will be positive if b is sufficiently large.

6. **Conclusion:**
Since we found values for a , b , and c that satisfy the recurrence relation, and c is positive for sufficiently large b , our assumption that (T(n) = an log n + bn is correct.

Therefore, by the substitution method, we have verified that (T(n) = Theta(n log n)) is indeed the solution to the recurrence relation (T(n) = 2T(n/2) + cn).

c) Solve the following recurrence relation using characteristic polynomial.

$$t_n = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ t_{n-1} + t_{n-2}, & \text{otherwise} \end{cases}$$

→ To solve the recurrence relation

$t_n = t_{n-1} + t_{n-2}$

we can used the characteristic polynomial method.

we can write recurrence relation in the form

$$t_n - t_{n-1} - t_{n-2} = 0$$

The characteristic polynomial for this recurrence relation is obtained by replacing t_n with x^n in the homogenous equation.

$x^n - x^{n-1} - x^{n-2} = 0.$

Divide both sides x^{n-2} to simplify

$x^2 - x - 1 = 0.$

Now, we solve the characteristic polynomial $x^2 - x - 1 = 0$ to find its roots.

Using the quadoatic formula

$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

where

$a = 1$, $b = -1$, and $c = -1$

$x = \frac{1 \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2(1)}$

$x = \frac{1 \pm \sqrt{1+4}}{2}$

$x = \frac{1 + \sqrt{5}}{2}$

So the roots of characteristic polynomial are $x = \frac{1+\sqrt{5}}{2}$ and $x = \frac{1-\sqrt{5}}{2}$

since the roots are distinct and real, the general sol^n of the recurrence relation is given by.

$t_n = A\left(\frac{1+\sqrt{5}}{2}\right)^n + B\left(\frac{1-\sqrt{5}}{2}\right)^n$

where A and B are constants

Given that $t_0 = 0$ and $t_1 = 1$, we substitute these values into the general solution to find values of A and B.

$t_0 = A\left(\frac{1+\sqrt{5}}{2}\right)^0 + B\left(\frac{1-\sqrt{5}}{2}\right)^0$

$= A + B = 0$

$$t_1 = A\left(\frac{1+\sqrt{5}}{2}\right)^1 + B\left(\frac{1-\sqrt{5}}{2}\right)^1$$

$$= \frac{A(1+\sqrt{5})}{2} + \frac{B(1-\sqrt{5})}{2}$$

$$= 1$$

solving these two equations simul-taneously gives us the values of A and B.

$$A + B = 0$$
$$A = -B$$

$$\frac{A(1+\sqrt{5})}{2} + \frac{B(1-\sqrt{5})}{2} = 1$$

$$-\frac{B(1+\sqrt{5})}{2} + \frac{B(1-\sqrt{5})}{2} = 1$$

$$-B + \frac{\cancel{2}B}{\cancel{2}}$$

$$= 1$$

$$B = -1$$

$$A = -B = 1$$

Substitute $A = 1$ and $B = -1$ back into the general solution.

$$t_n = \left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n$$

This is the solution to the given recurrence relation using the characteristic polynomial method.