Name : Mangesh Khode
Class : B.Tech CSE (H)
Subject : Design and Analysis of algo.

Q What are the desirable characteristics of an algorithm? Find the GCD of p=144 and q=55 using Euclid's algo.

Characteristics of an algorithm :-

① Well-Defined Inputs :- If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.

② Well-Defined Outputs :- The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.

③ Clear and Unambiguous :- The algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.

④ Finite-ness :- The algorithm must be finite, i.e. it should terminate after a finite time.

⑤ Effectiveness :- An algorithm must be developed by using very basic, simple and feasible operations so that one can trace it out by using just paper and pencil.

⑥ Language Independent :- The algorithm designed must be language independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

Euclid's algorithm :- The Euclid's algorithm is method of calculating the greatest common diviser (GCD) of two integers. The largest number that divides them both without a remainder is (GCD). (Divide the large number by smaller, the smaller by the remainder, the first remainder by the second remainder, and so on until the remainder becomes 0)

GCD of two number using Euclid's algorithm:

```
# include<stdio.h>
int main()
  { int p,q;
    printf("Enter two integers numbers:");
    scanf("%d %d", &p,&q);
    while (q≠0)
     { int r= p%q;
           p=q;
           q=r;
     }
     printf("GCD=%\n",p);
     return 0;
  }
```

$p = 144$ and $q = 55$

Iteration	p	q	$r = p \% q$
1	144	55	34
2	55	34	21
3	34	21	13
4	21	13	8
5	13	8	5
6	8	5	3
7	5	3	2
8	3	2	1
9	2	1	0
10	1 (GCD)	0 (stop)	—

Here, 1 is the greatest common divisor as we get remainder 0 in the last step.

Q Prove that $1 + 2 + 2^2 + \ldots\ldots + 2^n = 2^{n+1} - 1$, for all natural n.

$P(n): 1 + 2 + 2^2 + \ldots\ldots + 2^n = 2^{n+1} - 1$

(i) $P(1): 1 + 2 = 2^{1+1} - 1$

$3 = 4 - 1$

$3 = 3$

P(1) is True.

2) $P(k)$ is true

$P(k): 1+2+2^2+\dots+2^k = 2^{k+1}-1$ —(i)

3) $P(k+1): 1+2+2^2+\dots 2^k+2^{k+1} = 2^{(k+1)+1}-1$

Taking L.H.S.

$1+2+2^2+\dots+2^k+2^{k+1}$

$2^{k+1}-1+2^{k+1}$

$2.2^{k+1}-1$

$2^{(k+1)+1}-1$

$=$ R.H.S.

$P(k+1)$ is true.

by the principle of mathematical induction $P(n)$ is True for all natural Numbers n.

What are Asymptotic Notations in Complexity Analysis of Algorithm

The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine specific constants and don't require algorithms to be implemented and time taken by programs to be compared.

Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

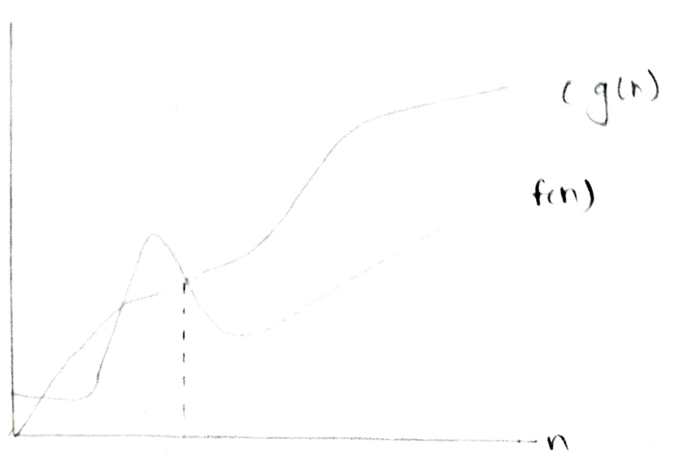
(1) Big-O Notation : Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm.

If f(n) describes the running time of an algorithm, f(n) is O(g(n)) if there exist a positive constant c and no such that,

$0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$

It returns the highest possible output value (big-O) for a given input.

The execution time serves as an upper bound on the algorithm's time complexity.



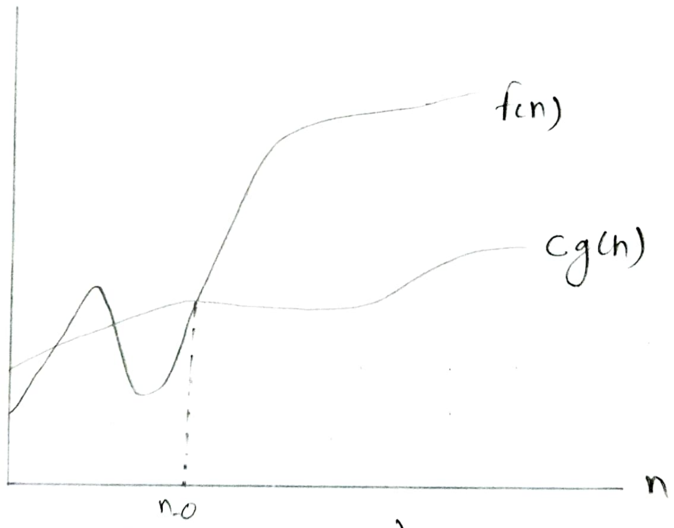
$f(n) = O(g(n))$

② **Omego Notation (Ω)** : Omega notation represent the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

The execution time serves as a both lower bound on the algorithm's time complexity.

It is defined as the condition, that allows an algorithm to complete statement execution in the shortest amount of time.

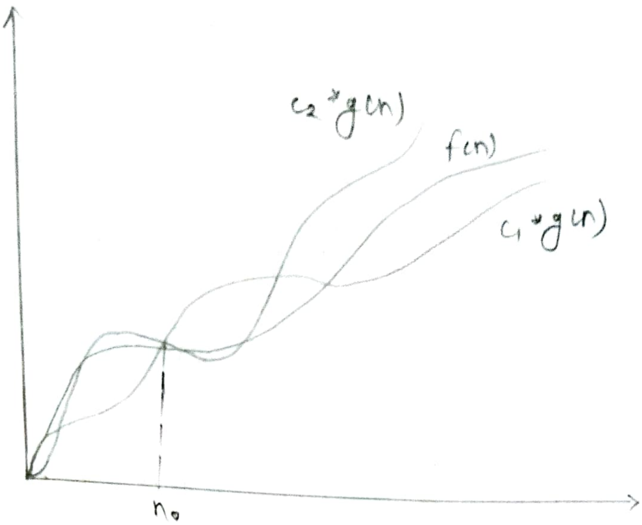
The function $f(n) = \Omega(g(n))$ if there exist positive constants c and n_0 such that $f(n) \geq c * g(n)$ for all n, $n \geq n_0$.



f(n) = Omega(g(n))

③ **Theta Notation (Θ)** : Theta notation enclose the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average case complexity of an algorithm.

The function $f(n) = \Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$.



Little "oh": $f(n) = o(g(n))$ if $f(n) < c\,g(n)$ for all values of $c > 0$ && for all values of $n \geq n_0$.

or

The function $f(n) = o(g(n))$ if,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Little 'omega': The function $f(n) = \omega(g(n))$ if $f(n) > c\,g(n)$ for all value of $c > 0$ && for all value of $n \geq n_0$.

or

~~lim~~

The function $f(n) = \omega(g(n))$ if,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \quad \text{or} \quad \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

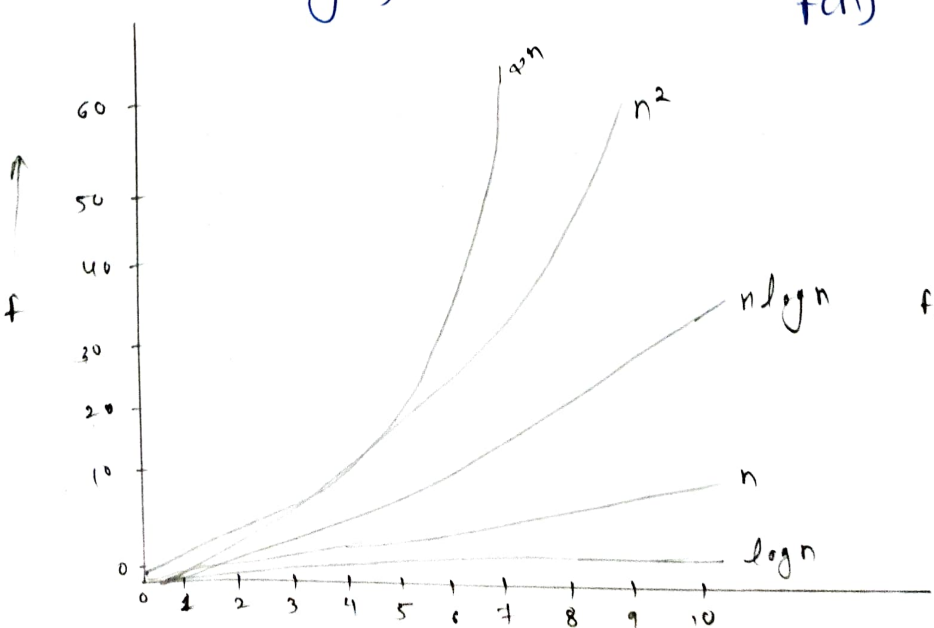


fig: Plot of function values

Examples :-

Big-oh (O) Notation :-

$f(n) = 2n+5$, $g(n) = n$

$f(n) \leq c.g(n)$

$2n+5 \leq c.n$

lets, $c = 3$

$2n+5 \leq 3n$

for $n=1$, $7 \leq 3$ False

for $n=2$, $9 \leq 6$ false

for $n=3$, $11 \leq 9$ False

for $n=4$, $13 \leq 12$ false

for $n=5$, $15 \leq 15$ true

$\therefore$ $f(n) = O(g(n))$

$2n+5 = O(n)$ for $\forall n \geq 5$ and $c=3$

Big Omega (Ω) Notation :-

$f(n) = n$, $g(n) = 3n+2$

$f(n) \geq c.g(n)$

$n \geq c.(3n+2)$

lets $c = \frac{1}{10}$

$10n \geq 3n+2$

for $n=1$ and $c = \frac{1}{10}$ is true.

Theta (Θ) Notation :-

$f(n) = 2n^2+n$, $g(n) = n^2$

check for O Notation :-

$f(n) \leq c.g(n)$

$2n^2+n \leq 3n^2$

This condition is true for $c=3$.

now, check for Ω Notation

$f(n) \geq c.g(n)$

$2n^2+n \geq 2n^2$

This condition is true for $c=2$.

$\therefore$ For theta (Θ) Notation,

$\cancel{2n^2 \leq}$

$c_1 g(n) \leq f(n) \leq c_2.g(n)$

$2n^2 \leq 2n^2+n \leq 3n^2$

where $c_1 = 2$ and $c_2 = 3$.