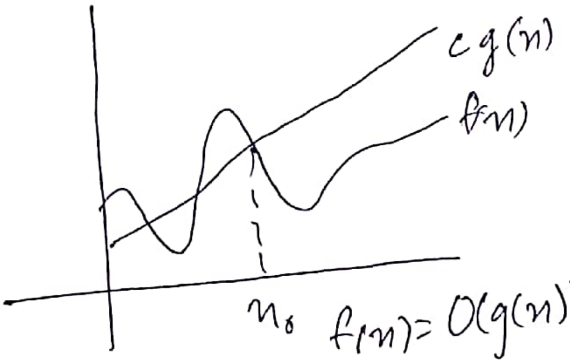
Q1 Asymptotic notations

They are mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value

There are mainly three types

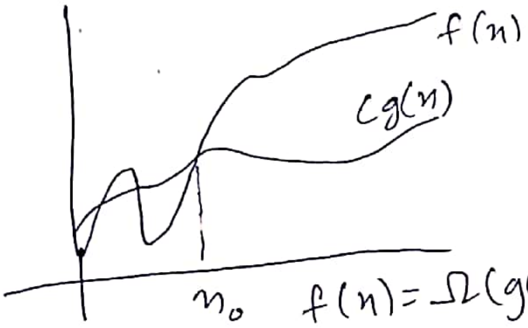
- Big-O notation - It represents the upper bound of the running time of an algorithm, thus gives worst time complexity of an algorithm



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

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$

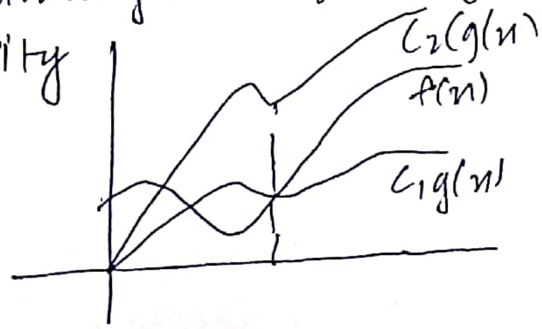
- Omega notation - It represents the lower bound of the running time of an algorithm, thus provides best case complexity.



$f(n) = \Omega(g(n))$

$\Omega(g(n)) = \{f(n)$: there exist positive constants c & n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$

- Theta notation - It represents lower & upper bound of running time of an algo. Thus gives average time complexity



$\Theta(g(n)) = \{f(n)$: there exists positive constants c_1, c_2 & n_0 such that $0 \le c_1g(n) \le f(n) \le c_2g(n)$ for all $n \ge n_0\}$

Q2 for (i=1 to n) { i = i*2 }

i	1	2	4	8 ...	2^k
Val	2^0	2^1	2^2	2^3	n

$$2^k = n$$
$$k \log_2 2 = \log_2 n$$
$$k = \log n$$
$$T.C = O(\log(n))$$

Q3 $T(n) = \begin{cases} 3T(n-1), & n>0 \\ 1 \end{cases}$

By forward

$T(n) = 3T(n-1)$, $T(0) = 1$

$T(1) = 3T(1-1)$
$= 3T(0)$
$= 3$

$T(2) = 3T(2-1)$
$= 3T(1) = 3 \times 3 = 3^2$

$T(3) = 3T(3-1)$
$= 3 \times T(2) = 3 \times 3^2 = 3^3$

$\vdots$

$T(n) = 3^n$

$T.C = O(3^n)$

Q4 $T(n) = \begin{cases} 2T(n-1)-1, & n>0 \\ 1, & n=0 \end{cases}$

$T(0) = 1$

$T(1) = 2T(1-1) - 1$
$= 2T(0) - 1$
$= 2 - 1 = 1$

$T(2) = 2T(2-1) - 1$
$= 2T(1) - 1$
$= 2 - 1 = 1$

$T(3) = 2T(3-1) - 1$
$= 2T(2) - 1$
$= 2(1) - 1 = 1$

$T(n) = 1$

$T.C = O(1)$

Q5

```
int i=1, s=1
while (s<=n)
{ i++;
  s=s+i;
  print("#");
}
```

let

~~see~~

~~3+16+10+15+ ... k = n~~

~~3+6+~~. for k iteration

$S(k) = 1+2+3+\dots+k = \frac{(k+1)*k}{2}$

$\frac{(k+1)k}{2} > n$

$k = O(\sqrt{n})$

$T.C = O(\sqrt{n})$

Q6 fun(int n)

```
{ int i, count=0;
   for (i=1; i*i<=n; i++)
      { c++; }
}
```

For $S(k) = 1^2 + 2^2 + 3^2 \ldots + k^2 \leq n$

$$= \frac{k(k+1)(2k+1)}{6} \leq n$$

$$= 2k^3 + 3k^2 + k \leq 6n$$

$$T(= \sqrt[3]{n}$$

Q7 fun(int n)

```
{ int i, j, k, c=0
   for (i=n/2, i<=n; i++)
     for (j=1; j<=n; j=j*2)
       for (k=1; k<=n; k=k*2)
          count++
```

outer loop run $n/2$ time

second loop runs $\log(n)$ time

third loop runs $\log(n)$ time

T.C $\frac{n}{2} * \log(n) * \log(n)$

$$T.C = O(n(\log_2 n)^2)$$

Q8 fun(int n)
{ if(n==1) return;
for (i=1 to n)
for (j=1 to n)
print("*")

fun(n-3);
}

for 1st loop O. n time
for 2nd loop n time
T.C = $n * n = O(n^2)$

Q9. fun(int n)
for (i=1 to n)
for(j=1; j<=n; j=j+i)
printf("1");

outer loop n times
inner loop logn times
T.C = $n * \log n = O(n \log n)$

Q10. n^k & c^n
$n^k = O(c^n)$