Design And Analysis Of Algorithms

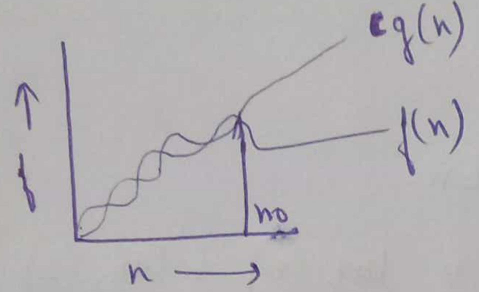
Assignment - 1

Yashika Dongi
53
SE

1. Asymptotic notations to analyse running time identifying its behaviour as the input size for the algorithm increases. These notations are used to tell the complexity of an algorithm when input is very large.

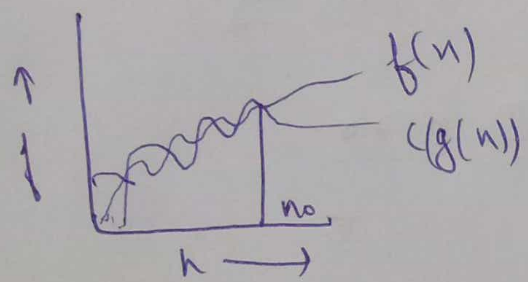
Types

1) Big-On



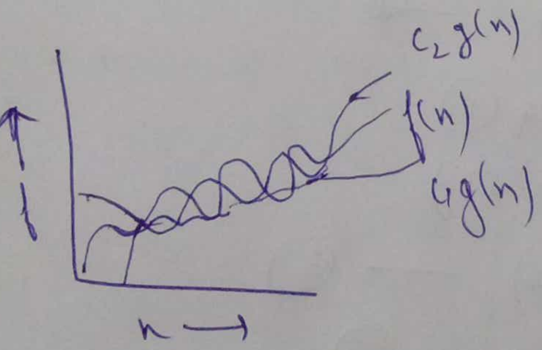
$f(n) = O(g(n))$
if and only if
$f(n) \le c.g(n)$
$\forall n \ge n_0$

2) Big Omega (Ω)



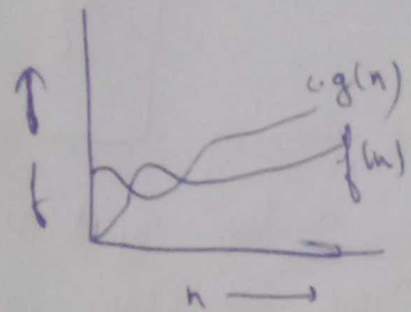
$f(n) = \Omega g(n)$
if and only if
$f(n) \ge c.g(n)$
$\forall n \ge n_0$

3) Theta (θ)



$f(n) = \theta g(n)$
if and only if
$c_1 g(n) \le f(n) \le c_2 g(n)$
$\forall n \ge \max(n_1, n_2)$

4) Small - Oh (o)



$f(n) = o(g(n))$

$f(n) < c.g(n) \; \forall \; n > n_0$

2. $i = 1, 2, 4, 8, \ldots n$

$2^0, 2^1, 2^2 \ldots\ldots 2^k$

$a = 1, \; r = 2$

$t_n = ar^{k-1}$

$= 1 \times 2^{k-1}$

$n = \frac{2^k}{2} \Rightarrow 2^k = 2n$

$k = \log_2(2n)$; $k = \log_2(n) + \log_2(2)$

$= \log_2(n) + 1$

$T.C = O(\log_2(n) + 1) = O(\log n)$

3. $T(n) = 3T(n-1)$ —— ① $n > 0$

$T(1) = 1$

put $n = n-1$ in eq ①

$T(n-1) = 3T(n-2)$ —— ②

put $T(n-1)$ in eq ①

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

$T(n) = 9T(n-2)$ —— ③

put $n = n-2$ in eq ①

$T(n-2) = 3T(n-3)$ —— ④

put $T(n-2)$ in eq ③

$T(n) = 9(3T(n-3))$
$= 27\,T(n-3)$

$T(n) = 3^k T(n-k)$ —— (5)

$T(1) = 1$
$n-k = 1$
$k = n-1$ —— (6)

from (5) & (6)

$T(n) = 3^{n-1}\,T(1)$

$T(n) = \frac{3^n}{3} \times 1$

$TC = O(3^n)$

4. $T(n) = 2T(n-1) - 1$ —— (1)

$T(1) = 1$

put $n = n-1$ in eq (1)

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

put $T(n-1)$ in eq (1)

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

$T(n) = 4T(n-2) - 2 - 1$ —— (2)

put $n = n-2$ in eq (1)

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

put $T(n-2)$ in eq (2)

$T(n) = 4(2T(n-3) - 1) - 2 - 1$

$T(n) = 8T(n-3) - 4 - 2 - 1$ —— (3)

$T(n) = 2^k [T(n-k)] - 2^{k-1} - 2^{k-2} - \dots - 2^1 - 2^0$ —— (4)

$T(1) = 1$

$n - k = 1$

$k = n - 1$ —— (6)

from (4) & (5)

$T(n) = 2^{n-1} [T(n-(n-1))] - 2^{n-2} - 2^{n-3} \dots 2^0$

$= 2^{n-2} - 2^{n-1} - 2^{n-3} - 1$

$= \frac{1}{2} [2^n - (2^n - 1)]$

$= \frac{1}{2} \times 1 = \frac{1}{2}$

$\boxed{TC = O(1)}$

5.

$n = 1, 3, 6, \dots k$ ⇒ AP

$TC = \frac{k(k+1)}{2}$

$O\left(\frac{k^2+k}{2}\right)$

$O(k^2)$

$TC.\ O(n^2)$

6.

```
void function (int n) {
  int i , count = 0          → 1
  for (i=1 ; i*i<=n ; i++)
        1     (n+1)^2    n
     count ++;
  }  n
```

$1 + 1 + (n+1)^2 + n + n$

$2 + n^2 + 2n + 1 + 2n$

$n^2 + 4n + 3$

$O(n^2 + 4n + 3)$

$O(n^2)$

$\boxed{TC = O(n^2)}$

7. void function (int n) {

int i, j, k, count = 0;

for (i = n/2 ; i <= n ; i++) - O(n)

for (j = 1 ; j <= n ; j = j * 2) - log(n)

for (k = 1 ; k <= n ; k = k * 2)

count ++; - log(n)

}

T.C $= \log(n) * \log(n)$

$= \log^2(n)$

$= O[\log^2(n)]$

8. function (int n) {

if (n == 1) return; - 1

for (i = 1 to n)

for (j = 1 to n)] - n * n

printf("*"); - 1

}

}

function (n-3) → $n * n^2$

}

$1 + n^2 + 1 + n^3$

$n^3 + n^2 + 2$

$\boxed{O(n^3)}$

9.

```
for ( i = 1 to n)
for (j=1 ; j<=n ; j = j+1)
        printf("*");
  }
}
```

T.C $= \log n . \frac{(n+1)}{2}$

$= O(\frac{n+1}{2} \log n)$

$= O(n \log n)$

10.

$n^k \leq c a^n$

$a^n + n^k \leq c a^n \rightarrow a^n$

$a^n + n^k \leq a^n (c-1)$

$\frac{a^n + n^k}{a^n} \leq (c-1)$

$c \geq 1 + \frac{n^k}{a^{n_0}} + 1$

$c \geq 2 + \frac{n_0^k}{a^n}$

$c \geq 2 + \frac{n_0}{1.5^n}$

$n_0 = 1$

$c \geq 2 + \frac{1}{1.5}$

$c \geq 3.0 + 1$

$c \geq 4$

11.

T.C $= O(n)$

1) while $\Rightarrow (n-1)$

2) $i = i + j \Rightarrow (n)$

3) $j{+}{+} \Rightarrow (n)$

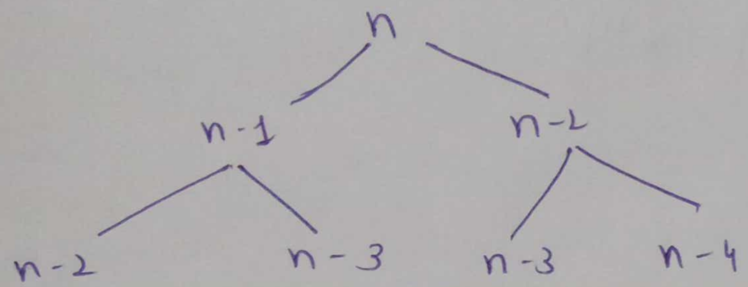
$T.C = n + n + n - 1$

$= 3n - 1$

$TC = O(3n-1)$

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

12 Main working ; $f(n) = f(n-1) + f(n-2)$



$T(n) = 1 + 2 + 4 + \cdots\ 2^n$

$a = 1, \quad r = 2$

$\frac{a(r-1)}{r-1} = \frac{1(2^{n+1}-1)}{2-1} = 2^{n+1}$

$T(n) = O(2^{n+1}) = O(2^n * 2^1) = O(2^n)$

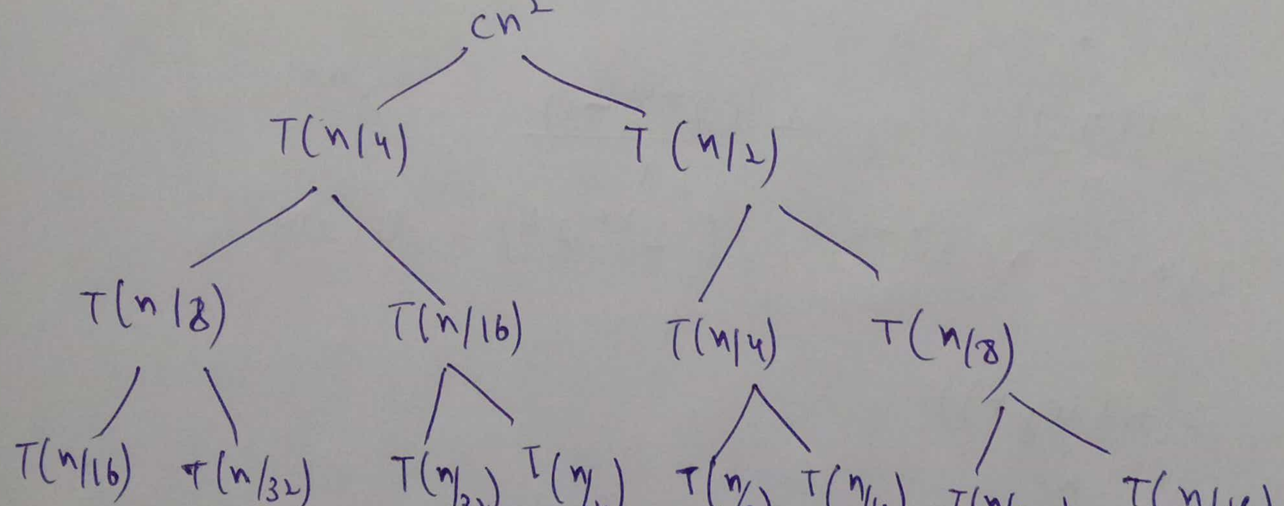
13 $O(n(\log n))$

→
```
int n;
for (int i = 0 ; i<n ; i++) {
 for (int j = n ; j>0 ; j/=2 ) {
    printf("*");
  }
}
```

$O(n^3)$

```
int i ,j,k;
for ( i=1; i<=n ; i++)
{
    for (j=1; j<=n ; j++) {
        for (k=1; k<=n ; k++)
        {
            printf ("*");
        }
    }
}
```

14 $T(n) = T(n/4) + T(n/2) + cn^2$



$$T(n) = c\left(n^2 + \frac{5(n^2)}{16} + 25\left(\frac{n^2}{256}\right) + \cdots\right.$$

ratio $= 5/16$

$= \dfrac{n^2}{1 - 5/16}$

$= O(n^2)$

15

```
int fun (int n) {
for (int i=1 ; i<=n ; i++) {
    for (int j=1 ; j<=n ; j+=i) {
        a[1] ; }
    }
```

$T(n) = n + \frac{n}{2} + \frac{n}{3} + \cdots \frac{n}{n}$

$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$

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

16 $i = 2, 2^c, 2^{c^2}, 2^{c^3} \cdots\cdots 2^{c^{\log_c \log(n)}}$

The last term has to be $<= n$

$2^{c^{\log_c(\log(n))}} = 2^{\log n} = n$

There are in total $\log_c(\log(n))$ iterations each take a constant amount of time to run.

$T.C = O(\log(\log n))$

18. a) $100 < \log\log n < \log n < \sqrt{n} < n < n\log n = \log(n!)$
$< n^2 < 2^n < 2^{2^n} < 4^n < n!$

b) $1 < \log\log(n) < \sqrt{\log(n)} < \log n < 2n < 4n < 2(2^n)$
$< \log(2N) < 2\log(n) < n < n\log n = \log(n!) < n < n^2$

c) $96 < \log_2(n) = \log_8(n) < n\log_6(n) = n\log_2(n) = \log(n!)$
$< 5n < 8n^2 < 7n^3 < 8^{2n}$

~~int fun(int arr[N] , key) {~~

19.

```
int fun ( int arr [N], key ) {
  for ( i=0 to n-1) {
     if ( Arr [i] = key ) {
        return i; }
  }
  return -1;
}
```

21.

Algorithm	Best Case	Average Case	Worst case
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

22.

Algorithm	In-place	Stable	Online
Bubble	✓	✓	X
Selection	✓	X	X
Insertion	✓	✓	✓
Merge	X	✓	X
Quick	X	X	X
Heap	✓	X	X

23

```
int BinarySearch (int arr [], int l, int r, int n)
while (l <= r) {
    int m = (l+r) / 2;
    if ( arr [m] = n;
       return m;
    else if (arr[m] < n)
       l = m+1;
    else r = m-1; }
    return -1;
}
```

RECURSIVE BINARY SEARCH

```
int Binarysearch (int arr []; int l , int r, int n)
   if ( l > r)
      return -1;
   int m = (l+r) / 2;
   if ( arr[m] = n;
      return m;
   else if ( arr[m] < n)
   return Binary Search (arr, m+1 , r, n);
     else
   return Binary Search (arr, l, m-1 , n);
}
```

Time complexity

Linear (Recursive) - $O(n)$

Binary (Recursive) - $O(n)$

Linear (Iterative) - $O(1)$

Binary (Iterative) - $O(1)$

Space Complexity

Linear (Recursive) - $O(1)$

Binary (Recursive) - $O(\log n)$

Linear (Iterative) - $O(1)$

Binary (Iterative) - $O(1)$

24. Recurrence relation for binary search

$$T(n) = T(n/2) + 1.$$