Name - Neha Saxena
Section - ML
RollNo - 46 (2014747)

Page No.:
Date:

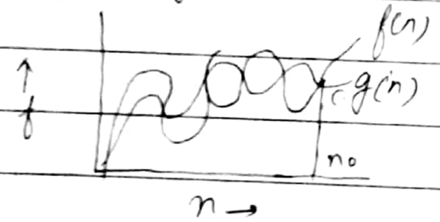
Design and analysis of algorithm

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

① Big On (O)

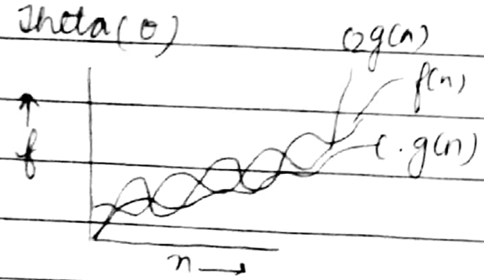
$f(n) = O(g(n))$
if and only if
$f(n) \le c\, g(n)$
$\forall\ n \ge n_0$

② Big Omega (Ω)



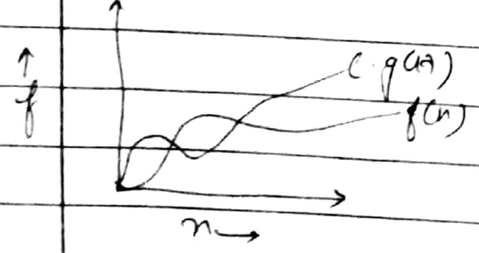
$f(n) = \Omega\, g(n)$
if and only if
$f(n) \ge c\, g(n)$
$\forall\ n > n_0$

③ Theta (θ)



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

④ small - On (o)



$f(n) = o(g(n))$
$f(n) < c.g(n)$
$\forall\ n \ge n_0$

Scanned with CamScanner

Page No.
Date

Ans 2. $i = 1, 2, 4, 8 \cdots n$

$2^0, 2^1, 2^2, 2^3 \cdots 2^k \rightarrow gp$

$a = 1 \quad , \quad r = 2$

$t_k = ar^{k-1}$

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

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

$2^k = 2n \quad \Rightarrow k = \log_2(2n)$

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

$\Rightarrow \log_2(n) + 1$

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

Ans 3. $T(n) = 3T(n-1) \quad — ① \quad n > 0$

$T(1) = 1$

put $n = n-1$ in eq ①

$T(n-1) = 3T(n-2) \quad — ②$

put $T(n-1)$ in eq ①

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

$T(n) = 9T(n-2) \quad — ③$

put $n = n-2$ in eq ①

$T(n-2) = 3T(n-3) \quad — ④$

put $T(n-2)$ in eq ③

$T(n) = 9(3T(n-3))$

$T(n) = 27T(n-3)$

$T(n) = 3^k T(n-k) \quad — ⑤$

$T(1) = 1$

$n - k = 1$

$k = n - 1 \quad — ⑥$

from ⑤ & ⑥

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

$T(n) = \frac{3^n}{3} \times 1 \quad \Rightarrow T.C = O(3^n)$

Scanned with CamScanner

Page No:
Date:

Ans 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)

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

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)

⋮

$T(n) = 2^k[T(n-k)] - 2^{k-1} - 2^{k-2} - 2^{k-3} - \cdots - 2^1 - 2^0$ —(4)

∴ $T(1) = 1$

$n - k = 1$

$k = n - 1$ —(5)

from (4) & (5)

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

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

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

$= \frac{1}{2} \times 1 = \frac{1}{2}$ $T.C = O(1)$

Ans 5. $n = 1, 3, 6 \cdots k \rightarrow A.P$

$T.C = \frac{k(k+1)}{2}$ $O\left(\frac{k^2+k}{2}\right) = O(k^2)$

$\boxed{T.C = O(n^2)}$

Scanned with CamScanner

ans 6. void function(int n) {
int i, count = 0; $\rightarrow 1$
for(int i = 1; i * i <= n; i++)
count++; } $- n$

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

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

$n^2 + 4n + 3$

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

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

ans 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) $\rightarrow O(\log n)$
for (k = 1 ; k <= n ; k = k * 2) $O(\log(n))$
count ++; }

$O(n) * O(\log n) * \log(n)$

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

ans 8. function (int n) {
if (n == 1) return ; $\rightarrow 1$
for (i = 1 to n)
for (j = 1 to n)] $n * n = n^2$
printf (" * "); $\rightarrow 1$
}
} function ((n-3)) $\rightarrow$ n2

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

$O(n^3)$

Page No.:
Date:

Ans 9. for (i = 1 to n)
for (j = 1; j <= n; j = j+1)
printf("*"); }
}

i	j	times
1	1→n	$\frac{n+1}{2}$
2	1→n	$n+1/2$
⋮		
n	1→n	$n+1/2$

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

Ans 10.

$$n^k \le c\, a^n$$
$$a^n + n^k \le c a^n - a^n$$
$$a^n + n^k \le a^n(c-1)$$
$$\frac{a^n + n^k}{a^n} \le c(c-1)$$
$$c \geq \frac{1+n^k}{a^{n_0}} + 1$$

$$c \geq 2 + \frac{n_0^{\,k}}{a^n}$$
$$c \geq 2 + \frac{n_0^{\,1}}{1.5^{\,n}}$$
$$n_0 = 1$$
$$c \geq 2 + \frac{1}{1.5}$$
$$c \geq 3.0 + 1 \Rightarrow c \geq 4$$

Ans 11. Time complexity $= O(n)$

The execution of diff code lines here are :-

1. while $(n-1)$
2. $i = i + j = (n)$
3. $j++;\ (n)$

$$T.C = n + n + n - 1$$
$$= 3n - 1$$
$$T.C = O(3n-1)$$
$$= O(n)$$

Ans12. the main working of fibonacci series is

$f(n) = f(n-1) + f(n-2)$

where base cond^n

$f(1) = 1$

n

n-1   n-2

n-2   n-3   n-3   n-4

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

$a = 1, \; r = 2$

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

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

Ans13. (1) $O(n(\log n))$

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

(2) $O(n^3)$

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

Page No.
Date.

$T(n) = T(n/4) + T(n/2) + Cn^2$

Ans 14

Cn^2 — 1

$T(n/4) \quad T(n/2)$ — 2

$T(n/8) \quad T(n/16) \quad T(n/4) \quad T(n/8)$ — 4

$T(n/56) \; T(n/32) \; T(n/32) \; T(n/64) \; T(n/8) \; T(n/16) \; T(n/32) \; T(n/16)$ — 8

n^2

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

ratio $= 5/16 \quad = n^2 / 1 - 5/16 = O(n^2)$

Ans 15

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

i	j	times
1	$1 \to n$	n
2	$1 \to n$	$n/2$
n		n/n

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

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

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

Ans 16

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

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.

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

Page No.:
Date:

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

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

Ans 19.
```
Int fun(int arr[N], key) {
    for (i = 0 to n-1) {
        if (arr[i] = key) {
            return i ; }}
    return -1; }
```

Ans 20. Iterative Insertion sort.
```
void insertionsort (int arr[], int n) {
    int i, temp, j;
    for (int i = 1; i <= n-1 ; j++)
    temp = arr [i];
    j = i - 1;
    while (j >= 0 && arr[j] > temp) {
        arr [j+1] = arr [i];
        i = j - 1; }
    arr [j+1] = temp;   }
```
Recursive insertion sort
```
void insertionsort (int arr [], int n) {
    if (n < 2) return;
    insertion sort (arr, n-1);
    last = arr [n-1], j = n-2;
    while (j >= 0 && arr[j] > temp) {
    arr [j+1] = arr [j] ,
    j = j - 1; }  arr [j+1] = last; }
```

Page No:
Date:

Ans 21.

	algorithm	Best case	avg case	worst case
1	Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
2	Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
3	Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
4	Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
5	Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
6	Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Ans 22.

	algorithm	inplace	stable	online
1	Bubble	✓	✓	x
2	Selection	✓	x	x
3	Insertion	✓	✓	✓
4	Merge	x	✓	x
5	Quick	x	x	x
6	Heap.	✓	x	x

Ans 23. Iterative Binary search

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

Recursive Binary search

```
int binarysearch (int arr[], int l, int r, int x)
```

Page No.:
Date:

```
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		Space complexity
Linear (recursive)	O(n)	O(1)
Binary (recursive)	O(n)	O(log n)
Linear (Iterative)	O(1)	O(1)
Binary (Iterative)	O(1)	O(1)

Ans 24. recursive relation for binary search

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

Scanned with CamScanner