Osheen Langer
Sec - A
Roll No - 30

2014425
classmate
Date
Page

Assignment - 1

1.

Ans - Asymptotic notations are mathematical tools to represent the time complexity of algorithm for asymptotic analysis.

. **O Notation** - it bounds a function from above & below, so it defines exact asymptotic behaviour.

Ex - $3n^3 + 6n^2 + 6000$

$O(n^3)$

Dropping lower order terms is always fine because there will always be no after which $O(n^3)$ has higher values than $O(n^2)$, irrespective of constants involved.

$O(g(n)) = \{f(n)$: there exists +ve constants c_1, c_2 and no such that $0 <= c_1 * g(n) <= f(n) <= c * g(n)$ for all $n >= n_0\}$

- **Big O (Notation)**: it defines an upper bound of an algo. it bounds a function only from above. for ex - consider the case of insertion sort. It takes linear time in best case and quadratic time in worst case.

$O(n^2)$ → worst case
$O(n)$ → best case
$O(n^2)$ is overall case.

$O(g(n)) = \{ f(n)$: there exists positive constants c & n_0 such that $0 <= c * g(n) <= f(n)$ for all $n > n_0 \}$.

- **Ω notation**:
provides lower bound.
(least used)

$\Omega(g(n)) = \{ f(n)$: there exists +ve const c and n_0 such that $0 \leq g(n) \leq f(n)$ for all $n >= n_0 \}$

ex: for fib(n):

fib(n-1) fib(n-2) => $O(2^n)$
=> $O(1.6^n)$
=> $\Omega(1.5^n)$

- O (notion): loose upper bound of f^n

let $f(n)$ and $g(n)$ be functions

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

$f(n) < c\ (g(n))$ — ex - Merge Sort

$\forall\ n > n_0$ &

$\forall\ c > 0$

- Ω notation: Strict lower bound — Small omega

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

$f(n) > c\ (g(n))$

$\forall\ n > n_0$ & $\forall\ c > 0$

ex - Merge Sort

Q.

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

Sol.

i	
1	2^0
2	2^1
⋮	⋮
n	2^K

$(K+1)^{th}$ iteration

$2^k = n$

$\log 2^k = \log n$

$k \log_2 2 = \log n \quad (\log_2 2 = 1)$

$k = \log(n)$

$\boxed{O(k) = O(\log n)}$

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

Sol.

Using Substitution

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

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

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

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

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

$= 3^n(T(0))$

$= 3^n * 1 = 3^n$

$= O(3^n)$

(4) $T(n) = \begin{cases} 2T(n-1)-1 & \text{if } n>0, \text{ else } 1 \end{cases}$

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

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

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

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

$= 2^n(T(n-n)) - 2^{n-1} - 2^{n-2} - 2^{n-3}$

$= 2^n - 2^{n-1} - 2^{n-2} \ldots 2^0$

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

$= 1$

$T(n) = 1 \Rightarrow O(1)$

(5) T.C $\Rightarrow$

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

Sol.

i	s
1	1
2	3
3	6
4	10
5	15

$1 + 2 + 3 + \cdots K$ at K^{th}

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

$$\frac{K^2 + K}{2} >= n$$

$$K^2 >= n$$

$$K >= \sqrt{n}$$

$O(K) \Rightarrow O(\sqrt{n})$

(6)
```
void function (int n)
{
int i, c=0;
for (i=1; i*i<=n; i++)
  c++;
}
```

i

1

2^2

3^3

.

.

.

K^2

$K^2 >= n$

$K >= \sqrt{n}$

$\boxed{O(\sqrt{n})}$

(7)
```
void 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)
        c++;
}
```

i	j	k
$n/2$	$\rightarrow 1$	$\rightarrow 1$

$n/2$ → $2^0, 2^1, 2^2, \ldots, 2^{p-1}, i$ → $\log(n)$

$2^0, 2^1, 2^{p-1}$ → $\log(n)$

$O(n/2 * \log(n) * \log(n))$

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

$\Rightarrow \boxed{O(n \log^2 n)}$

(8)

```
fun ( int n)
{ if (n == 1)
    return;

  for (i = 1 to n)
  {
      for (j = 1 to n)
      { printf("*");
      }
  }
  fun (n-3);
}
```

Sol. $T(n) = T(n-3) + n^2$

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

$T(1) = 1$

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

Put $n = n-3$

$T(n-3) = T(n-6) + (n-3)^2$

$T(n-6) = T(n-9) + (n-6)^2$

$T(n) = T(n-6) + (n-3)^2 + n^2$

$= T(n-k) + (n-(k-3))^2 + (n-(k-6))^2 + \ldots\ldots n^2$

put $n-k=1$, $k=n-1$

$$+(1)+(-2)^2+(-5)^2+(-8)^2+\ldots\ldots n^2$$
$$=1+4+25\ldots\ldots n^2$$
$$=1+\Sigma(3n-1)^2$$
$$=\Sigma(3n-1)^2$$
$$=\Sigma(9n^2+1-6n)$$
$$=\frac{9(n)(n+1)(2n+1)}{6}$$
$$\approx n^3$$

$O(n^3)$

⑥
```
void fun (int n)
{
  for (i=1 to n)
  {
  for (j=1; j<=n; j=j+i)
      { printf("*");
  }
```

i	j
1	n-times
2	n/2 -times
:	:
n	1 time

$O(n \log n)$

(10) $n^k = O(a^n)$

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

$c * a^n \leq n^k$

for $k \geq 1$

$a > 1$

let $k = 2$

$a = 2$

$c * a 2^n \leq n^2$

$n = 2$

$c = 1$

let $k = 1$

$a = 2$

$c * 2^n \leq n$

$n = 0, c = 0$

let $k = 2, a = 2$

$f(n) = n^2$

$g(n) = 2^k$

[for $k \geq 2$ & $a \geq 2$

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

$n^2 \geq c * 2^k$

log:

$O(\log n)$

(11)

```
void fun (int (n)
{
  int j=1, i=0;
  while (i<n)
  {
    i = i + j;
  } j++;
```

j	i
1	1
2	3
3	6
4	10

At i^{th} iteration, i is sum of integers till j.

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

$$k^2 >= n$$

$$k >= \sqrt{n}$$

$$\boxed{O(\sqrt{n})}$$

(12) Recurrence Relation

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

$= 2T(n-1) + O(1) + (n-1) \geq T(n-2)$

$= 2 * T(n-1)$

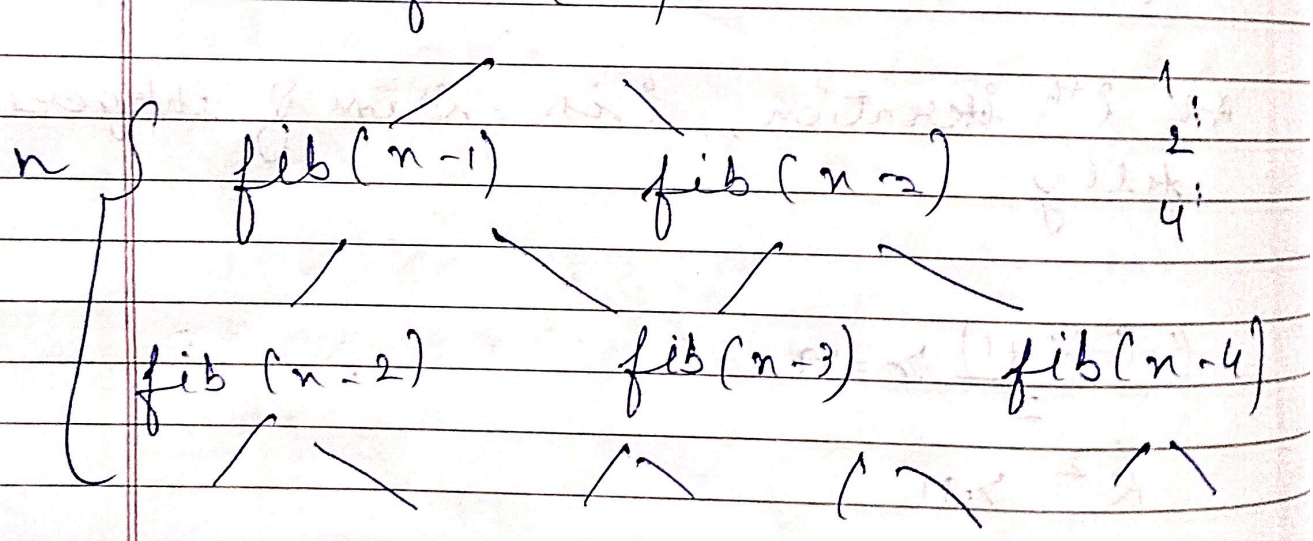
$= 2^2 * T(n-2)$

⋮

$= 2^n * T(n-n)$

$\Rightarrow O(2^n)$

$\Rightarrow O(1.68^n)$



$\therefore C = O(n)$

(3) $n(\log n)$, n^3, $\log(\log(n))$

(1) $n(\log n)$

```
for (int i=1; i<=n; i++)
  {
    for(int j=1; j<=n; j=j*2)
      {
        //O(1) task;
      }
  }
```

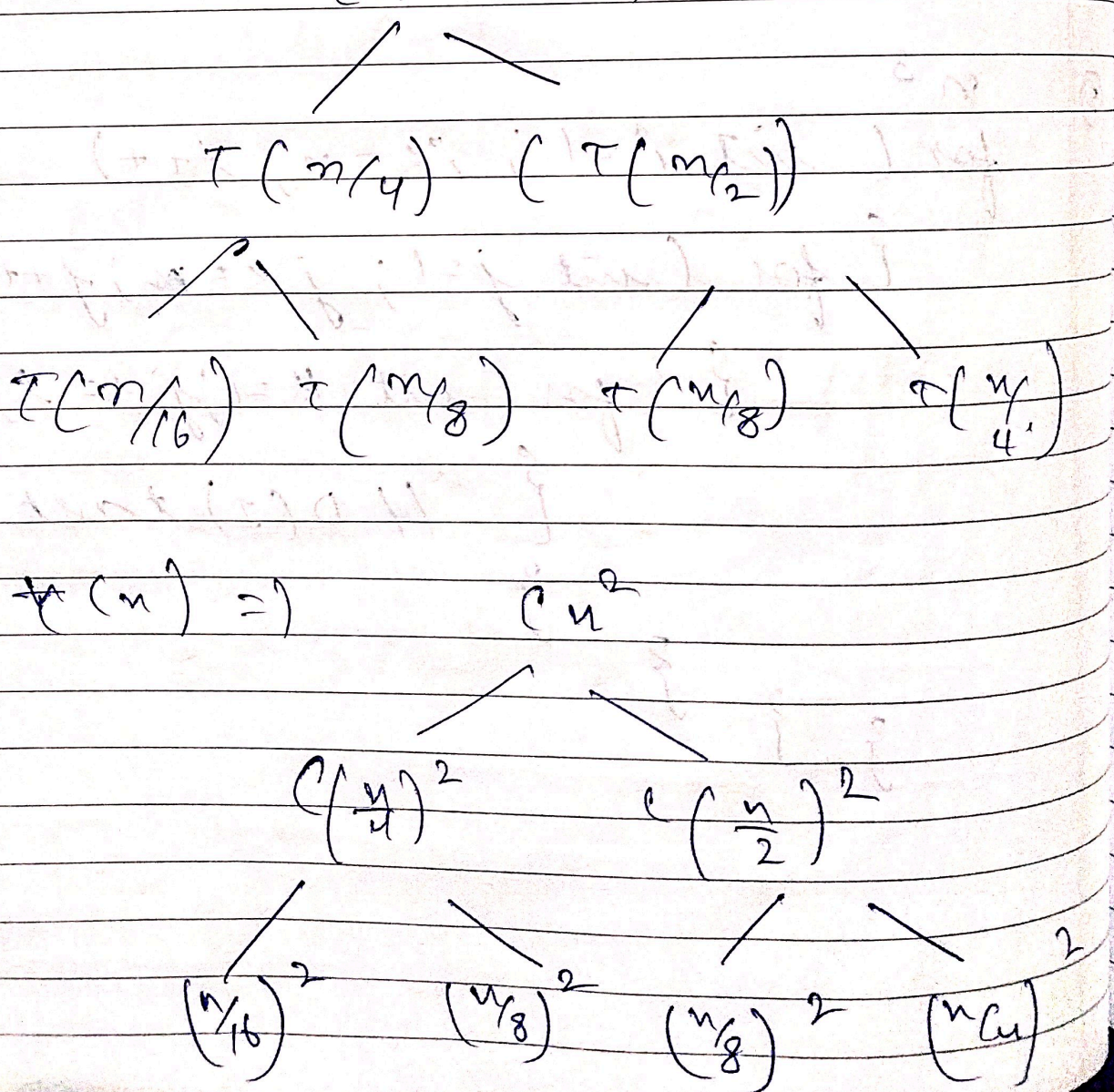
(2) n^3

```
for ( int i=1 ; i<=n ; i++)
  { for (int j=1 ; j<=m ; j++)
      { for (int k=1; j<=l; k++)
          { //O(1) task
          }
      }
  }
```

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

```
for( int i=2; i<=n; i=pow(i,d))
  { // some O(1) task
  }
```

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



Sum of 3 levels:

$$cn^2 + c\left(\frac{n}{4}\right)^2 + c\left(\frac{n}{2}\right)^2 + c\left(\frac{n}{16}\right)^2 + c\left(\frac{n}{8}\right)^2 + c\left(\frac{n}{8}\right)^2 + c\left(\frac{n}{4}\right)^2$$

$$\Rightarrow c\left(n^2 + 5\left(\frac{n^2}{16}\right) + 25\, n^2/256 + \ldots\ldots n\right)$$

→ G.P: $r = 5/16$, $a = cn^2$.

$$Sn = \frac{a}{1 - r}$$

$$= \frac{cn^2}{1 - \frac{5}{16}} \Rightarrow \frac{c16n^2}{11}$$

$$= O(n^2)$$

(5)

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

$i = 1 \rightarrow n$ times

i^{th} iteration $\rightarrow n/i$ times

$$T(n) = O(n(1 + 1/2 + 1/3 + \dots))$$

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

(16)

```
for (int i=2; i<=n; i=pow(i,k))
    { //O(1);
    }
```

$k \rightarrow$ constant

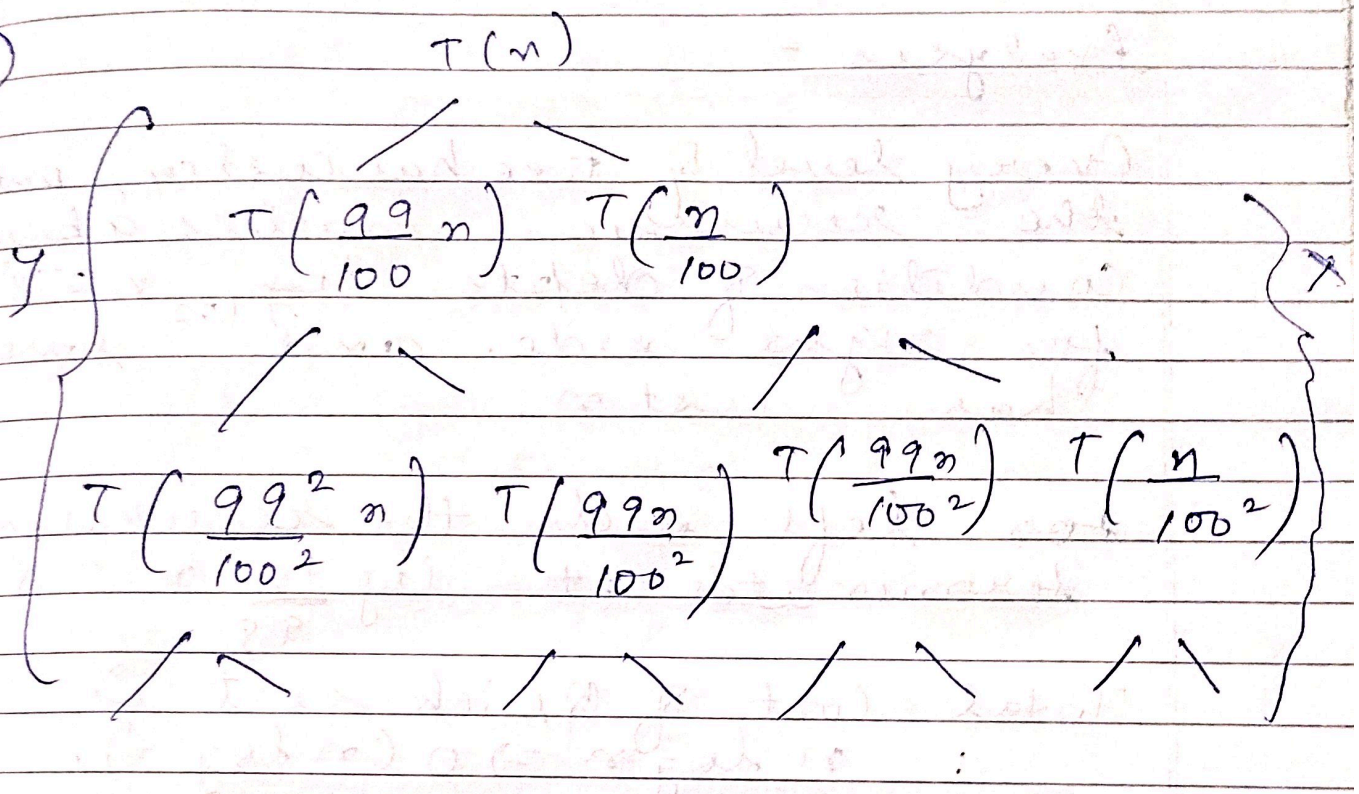
Sol: i takes $2, 2^k, (2^k)^k, \dots$

$2^{*\log k} \log(k)$

$\Rightarrow$ If $\log_2 n = n \rightarrow$ last term

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

(1)



$y = \log_{\frac{100}{99}} n$ → height of left side

$x = \log_{100} n$ → height of right side

Difference :

$$\log_{100} n - \frac{\log_{100} n}{99}$$

$$T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right) + n$$

↳ Recurrence relation

Analysis:-

Every level of tree has cost cn, until the recursion reaches a boundary condition of depth $\log_{100} n$ ≥ $O(\log n)$ for right side. and levels has cost n.

For left side the recursion terminates at $\log_{\frac{100}{98}} n = O(\log n)$

Total cost of Quick sort is
∴ $n \log n \Rightarrow O(n \log n)$.

(18)

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

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

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

(19)

```
linear-search (a[], item, pos, n)
pos = -1
for (i=0; a[i] != item && pos=-1
      && i<n; i++)
{ if a[i] == item)
{ pos = i
}
return (pos)
```

(20) Iterative insertion sort

```
void insertion sort (int arr[], int n)
{
  int i, key, j;
for (i=1; i<n; i++)
{ key = arr[i];
  j = i-1;
  while (j>=0 && arr[j] <key)
{ arr[j+1] = arr[j];
  j = j-1;
}arr[j+1]= key;
}
}
```

Recursive insertion Sort:

```
void recursivesort (int arr [], int n)
{
  if (n <= 1)
    return;
  recursivesort (arr, n-1);
  int i = arr [n-1];
  int j = n-2;
  while (j >= 0 && arr [j] > i)
  {
    arr[j+1] = arr[j];
    j--;
  }
    arr [j+1] = i;
}
```

* Insertion Sort is online Sorting Algo:

Because online algos are that can process it input piece by piece in serial fuction, i.e the in order that the input is fed to the algorithm without having entire input available from beginning & insertion does not know whole input.

Offline Algorithm	Online
Bubble sort, mergesort, selection sort, quick sort	Insertion sort

(21) Complexity: (Sorting Algo)

	Best	Avg	Worst
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n \log(n))$	$\Theta(n \log n)$	$O(n^2)$
Merge Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$

(22) Sorting Algorithm — Inplace — Stable — Online

Selection Sort	✓	✗	✗
Bubble "	✓	✓	✗
Insertion "	✓	✓	✗
Quick Sort	✓	✗	✓
Merge "	✗	✓	✗

(23) Recursive Binary Search:

```
int binSer(int arr[], int l, int r, int key)
{
  if (r >= l)
  {
    int mid = l + (r - l)/2;
    if (arr[mid] == key)
      return mid;
    if (arr[mid] > key)
      return bin(arr, l, mid-1, key);
    else
      return bin(arr, mid+1, r, key);
  }
  return -1;
}
```

T.C → O(log(n))

S.C → O(log(n))

(23) Iterative Binary Search:

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

T.C → O(log n)
S.C → O(1)

(24) R.R for Binary Search

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