ASSIGNMENT-1

Name- Aastha saigal
section- D
University Roll No- 2014355
subject- DAA

① Sol :- Asymptotic notations are used to tell the complexity of an algorithm when input is very large.

The types of asymptotic notations are:-

a) Big-O Notation.

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

b) Big-Ω Notation.

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

c) Theta Notation

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

d) Small-o Notation

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

e) small-omega (ω) notation

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

② complexity of :-

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

Sol^n

$$\Rightarrow \boxed{T(n) = O(\log n)}$$

③ Sol^n

$$T(n) = 3T(n-1) \quad \text{if } n>0$$

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

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

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

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

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

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

$\vdots$

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

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

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

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

④ Sol^n

$$T(n) = 2T(n-1)-1 \quad \text{if } n>0$$

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

$$T(n\cancel{-1}) = 2(2T(n-2)-1)-1$$

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

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

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

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

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

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

$$T(n) = 2^3 T(n-4)-(1+2+4+8)$$

$$T(n) = 2^n \, T(n-n) - (1+2+4+8+\cdots) \quad [a=1, r=2]$$

$$T(n) = 2^n \, T(0) - \left(\frac{1(2^n-1)}{2-1}\right)$$

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

$$T(n) = \cancel{2^n} - \cancel{2^n} + 1$$

$$T(n) = 1$$

$$\boxed{T(n) = O(1)}$$

(5)

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

i	s
$\boxed{\cancel{1}}$	$\boxed{\cancel{1}}$
$\cancel{2}$	$\cancel{2}$
$\cancel{3}$	$\cancel{4}$
4	7

sol^n: $i = 1, 2, 3, 4 \ldots k$

$$n = \frac{k(k+1)}{2} \; ; \; n = \frac{k^2+k}{2} \; ; \; n = k^2 \; ; \; k = \sqrt{n}$$

$$\boxed{\text{Time complexity} = O(\sqrt{n})}$$

(6)

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

$\underline{\underline{sol^n}}$

i $\boxed{\cancel{1}}$ $\cancel{4}$ $\cancel{9}$ 16

$i = 1, 4, 9, 16 \ldots k$

$k = n^2$

$\boxed{\text{Time complexity} = O(n^2)}$

7 void fun(int n)

```
{
    int i, j, k, count = 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++
}
```

$\underline{sol^n \text{ for } i}$ -

i $\boxed{\cancel{n/2}}$ $\cancel{\frac{n+2}{2}}$ $\cancel{\frac{n+4}{2}}$

$i = \frac{n}{2}, \frac{n}{2}, \frac{n}{2} \ldots k$

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

$\underline{\text{for } j}$ -

j $\boxed{\cancel{1}}$ $\cancel{2}$ $\cancel{4}$ 8

$j = 1, 2, 4, 8, 16 \ldots k$

$n = 1 \cdot 2^{k-1}$

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

{

$2n = 2^k$

$k = \log_2(2n)$

$k = \log_2 n$

<u>for k -</u>

$k\ \boxed{1}\ \cancel{2}\ \cancel{4}\ 8$

$k = 1, 2, 4, \cancel{8}\ 16$

$n = 1 \cdot 2^{k-1}$

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

$2n = 2^k$

$k = \log_2(2n)$

$k = \log_2(n)$

$\therefore$ The complexity of the function is

$\Rightarrow O\left(\frac{n}{2}\log_2 n \cdot \log_2 n\right)$

$\cancel{\Rightarrow O\left(\frac{n}{2}(\log n)\right)}$

$\cancel{\Rightarrow O\left(\frac{n}{2} \times 2 \log_2 n\right)}$

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

8 fun (int n)

{

if (n==1)

return;

for (i=1 to n)

{

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

Sol^n i loop executes n times
j loop executes n times.

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

9

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

Sol^n <u>for i</u>

$i \; \boxed{\cancel{1}} \; \cancel{2} \; 3$

$i = 1, 2, 3, \ldots n$

i executes n times

$j \; \boxed{\cancel{1}} \; \cancel{3} \; \cancel{6} \; 10$

$j = 1, 3, 6, 10, 15, 21, \ldots \cancel{n}$

$a_n = \frac{1}{2} n(n+1)$ (Triangular No. Sequence).

$a_n = \frac{1}{2}(n^2+n)$

Time complexity $= O(n(n^2+n))$

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

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

(11)
```
void fun(int n)
{
  int j=1, i=0;
  while (i<n)
  {
    i=i+j;
    j++;
  }
}
```

Sol^n

$j = \cancel{1} \; \cancel{2} \; \cancel{3} \; \cancel{4} \; 5$

$i = 0, 1, 3, 6, 10, 15 \ldots$

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

$a_n = \frac{1}{2}(n^2+n) \Rightarrow \boxed{O(n^2)}$

(12) Recurrence Relation of fibonacci series is :-

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

$$T(0) = T(1) = 1$$

$\Rightarrow$ $T(n-1) \approx T(n-2)$

$$T(n) = 2T(n-2) + c \qquad (c=4)$$

$$T(n) = 2(2T(n-4) + c) + c$$

$$T(n) = 4T(n-4) + 3c \qquad (k=2)$$

$$T(n) = 8T(n-6) + 7c \qquad (k=3)$$

$$T(n) = 16T(n-8) + 15c \qquad (k=4)$$

$$T(n) = 2^k T(n-2k) + (2^k - 1)c$$

$$n - 2k = 0$$

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

$$T(n) = 2^{n/2} T(0) + (2^{n/2} - 1)c$$

$$= 2^{n/2} + 2^{n/2}c - c$$

$$= 2^{n/2}(1+c) - c$$

$T(n) \propto 2^{n/2}$ (lower bound)

$\Rightarrow$ $T(n-2) \approx T(n-1)$

$$T(n) = 2T(n-1) + c \qquad (c=4)$$

$$T(n) = 4T(n-2) + 3c$$

$$T(n) = 8T(n-3) + 7c$$

$$T(n) = 2^K \, T(n-K) + (2^K - 1)c$$

$$n - K = 0 \Rightarrow K = n$$

$$T(n) = 2^n \, T(0) + (2^n - 1)\, c$$
$$= (1+C)\, 2^n - C$$

$T(n) \propto 2^n$ (upper bound)

(exponential time algo)
$O(2^n) \rightarrow$ Fib (recursion)

$O(n) \rightarrow$ Fib (iterative)
(linear time algo)

(13) Write programs which have complexity $n(\log n)$, n^3, $\log(\log(n))$

(i) $n(\log n)$

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

(ii) n^3

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

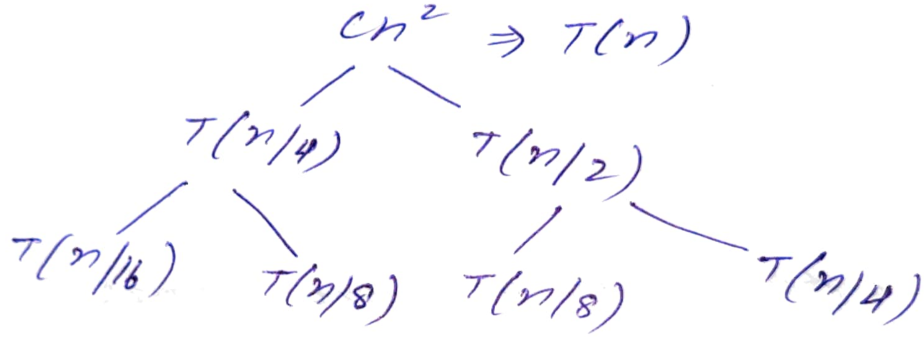
(iii) $\log(\log(n))$

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

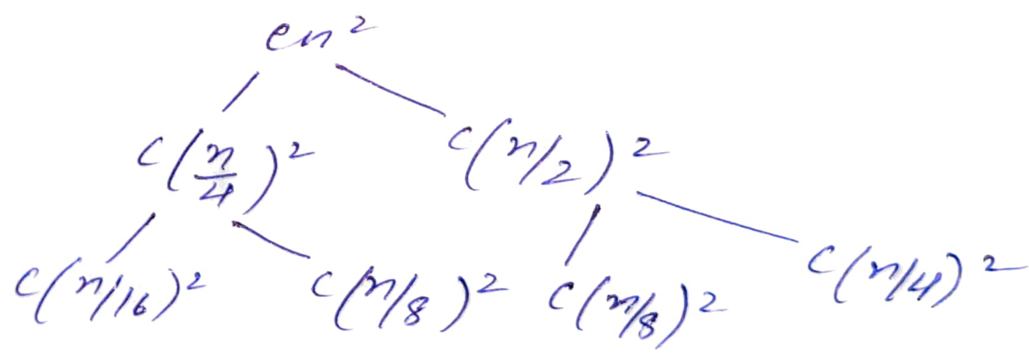
(14) solve recurrence relation:

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

sol



$T(n) \Rightarrow$



Sum of three levels -

$$= cn^2 + c(n/4)^2 + c(n/2)^2 + c(n/16)^2 + c(n/8)^2 + c(n/8)^2 + c(n/4)^2$$

$$= c(n^2 + 5n^2/16 + 25n^2/256 + \cdots \infty)$$

GP with, $r = 5/16$

$a = cn^2$

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

$$= \frac{cn^2}{\left(1-\frac{5}{16}\right)}$$

$$= \frac{c16n^2}{11} = \underline{O(n^2)}$$

OR

GP with $r = 5/16$, $a = cn^2$

terms $\rightarrow \log_2 n$

$$S = cn^2 \frac{\left(1-(5/16)^{\log_2 n}\right)}{(1-5/16)}$$

$$\Rightarrow \underline{O(n^2)}$$

(15)

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

$i=1 \longrightarrow n$ times

ith iteration $\longrightarrow n/i$ times

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

$$= \underline{O(n \log n)}$$

(16) for (int i = 2; i <= n; i = pow(i, k))

```
{
  // some O(1) task
}
```

where k is constt.

Sol^n i takes $2, 2^k, (2^k)^k, \ldots$

$2^{k^{\log_k \log(n)}}$

$\Rightarrow 2^{\log_2 n} = n \rightarrow$ last term

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

$2^{k^0}, 2^{k^1}, 2^{k^2} \ldots \ldots 2^{k^{c-1}}$

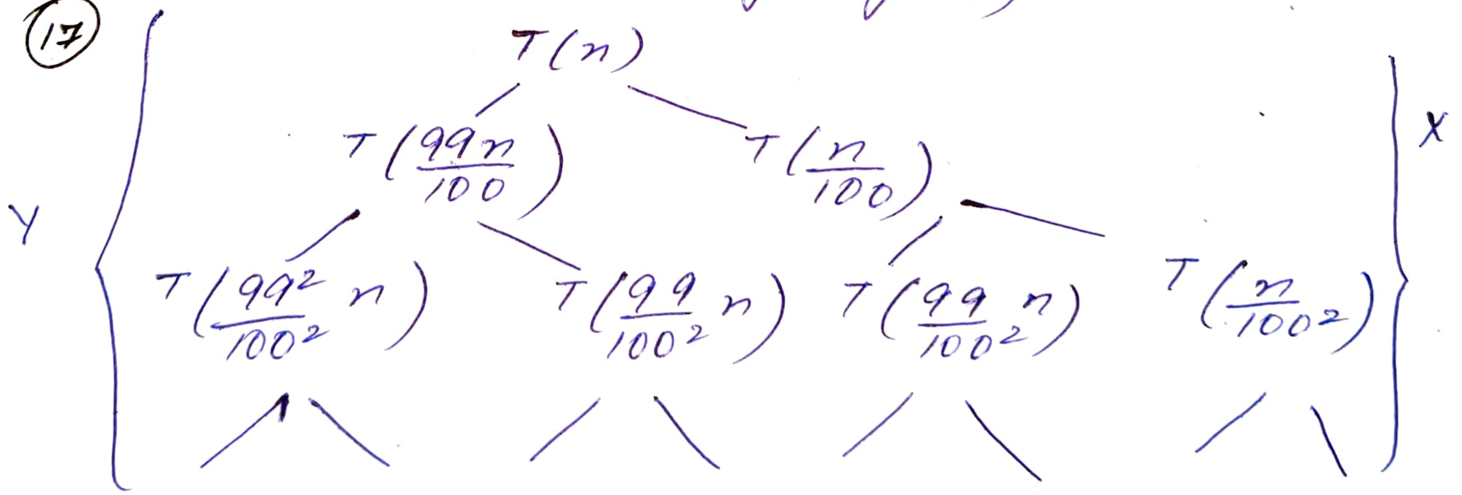
$2^{k^{c-1}} < n \Rightarrow k^{c-1} < \log_2 n$

$c - 1 < \log_k \log_2 n$

(base does not matter) $c < \log_k \log_2 n$

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

(17)



$Y = \log_{\frac{100}{99}} n \rightarrow$ height of left side

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

Diff b/w heights of extreme part

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

99 parts and 1 part

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

↳ Recurrence Relation

<u>analysis</u>-

Every level of tree has cost cn, until the recursion reaches a boundary condition at depth $\log_{100} n \Rightarrow O(\log n)$ for the right side and the level have cost n.
For left side the recursion terminates at $\log_{\frac{100}{99}} n = O(\log n)$.
Total cost of quick sort is therefore $n \log n \Rightarrow O(n \log n)$. since the split has constt proportionality it yields $O(n \log n)$

(18)

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

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

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

(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) <u>Iterative Insertion Sort</u>

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

Insertion sort is online sorting algorithm because online algorithms is one that can process its input piece by piece in serial fashion, i.e in order that the input is fed to the algorithm with-out having entire input available from beginning and insertion does not know the whole input.

Offline Algos-
bubble sort, merge sort, selection sort, Quick sort.

Online Algos-
Insertion sort

(21) <u>complexity of all sorting algorithms -</u>

	Best	Average	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)

<u>Sorting Algo.</u>	<u>Inplace</u>	<u>Stable</u>	<u>Online</u>
Selection Sort	✓	✗	✗
Bubble Sort	✓	✓	✗
Insertion Sort	✓	✓	✓
Quick sort	✓	✗	✗
Merge sort	✗	✓	✗

(23) <u>Recursive Binary Search</u> :-

```
int binarySearch (int arr [], int l, int r,
                                     int x)
{
 if (r>=l) {
 int mid = l+(r-1)/2;
 if (arr [mid] == x)
     return mid;
 if (arr [mid] > x)
 return binarySearch (arr, l, mid-1, x);
 return binarySearch (arr, mid+1, r, x);
}
```

```
return -1;
}
```

T.C - O(log n)
S.C - O(log n)

Iterative binary search -

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

T.C - O(log n)
S.C - O(1)

(24) Recurrence relation for binary search

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