: Rakshit
: B/39
DAA - assignment.

PAGE No.
DATE

Q1 Are languages that allow us to analyse an algorithm running time by identifying its behaviour as the input size increases

Types - Theta (Θ) → avg-value

Big oh (O) → worst case, upper bound

Omega (Ω) → used to define lower bound or best case.

Q.2 $i \Rightarrow 1, 2, 4, 8 \dots n$

$a = 1 \quad r = 2$

$t_k = a r^{k-1}$

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

$2n = 2^k$

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

$k = \log_2(n) + 1$

$\Rightarrow O(\log_2(n) + 1) \Rightarrow O(\log n)$

Q.3 $T(n) = 3T(n-1)$ $n>0$, else 1

using backward sub

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

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

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

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

$\vdots$

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

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

$\because T(0) = 1$

$\therefore$ Complexity $\Rightarrow O(3^n)$

Q.4 $T(n) = 2T(n-1) - 1$ $n>0$, else 1

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

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

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

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

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

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

$\vdots$

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

$\because T(0) = 1$

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

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

∴ complexity $\Rightarrow O(1)$

~~i: 1, 1+1, 1+1+1~~

~~s: 1, 1+2, 1+2+3, 1+2+3+4 ... n~~

A.5. s = 1, 3, 6, 10 ... n

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

$k^2 = n$

$k = \sqrt{n}$

∴ complexity = $O(\sqrt{n})$

A.6 ~~loops runs till~~

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

A.7. loops) i j k

$n/2$ $\log n$ $\log n$

complexity $\Rightarrow \frac{n}{2} \times \log n \times \log n$

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

A-8

Outermost loop	i	j
↓	↓	↓
n/3	n	n

Complexity :- n^3

A-9

i	j
1	n times
2	n/2 times
3	n/3 times
⋮	⋮
n	n/n times
n	log n

complexity :- $O(n \log n)$

A-10 Since polynomials grow slower than exponentials. n^k has an asymptomatic upper bound of $O(a^n)$

for, $a = 2$, $n_0 = 2$

A.11

j	i
2	1
3	3
4	6
5	10
	⋮

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

$$k^2 = n$$

$$k = \sqrt{n}$$

$\therefore$ Complexity $= \sqrt{n}$

A.12 $T(0) = 0$, $T(1) = 0$

$T(n) = T(n-1) + T(n-2) + 1 \quad n > 1$

Let $T(n-1) \simeq T(n-2)$

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

using backward soln.

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

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

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

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

$= 8\,T(n-$

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

$$T(0) = 0$$

$$n - k = 0$$

$$n = k$$

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

$$= 2^n + 2^n - 1$$

$\therefore$ Complexity $\Rightarrow$ $O(2^n)$

A-13 $n \log n$

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

n^3

```
void fun(int n){
    for(i=1 to n){
        for(j=1 to n){
            for(k=1 to n)
                //some O(1) fast.
        }
    }
}
```

log log(n)

```
void func(int n){
    for(i=n; i>1; i=pow(i,k))
        //some O(1) stmt
}
```

PAGE No.
DATE

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

assume $T(n/2) >= T(n/4)$

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

$c = \log_b^a$

$= \log_2^2 = 1$

$\therefore n^c < f(n)$

Complexity = $\Theta(n^2)$

A.5

i	j
1	n times
2	n/2 times
3	n/3 times
.	.
.	.
n	n/n times

$\frac{n}{n}$ $\underline{\log n}$

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

A.16. i. takes $= 2, 2^k, (2^k)^k, (2^{k^2})^k = 2^{k^3} \dots 2^{k^{\log_k(\log(n))}}$

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

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

Hence ~~d~~, ~~(b)~~ time complexity $\Rightarrow$ ~~(b)~~ $O(\log\log(n))$

A.17 ~~Taking first Eq~~

$$T(n) = T(9n/10) + T(n/10) + O(n)$$

taking one branch 99% and other 1%.

$$T(n) = T(99n/100) + T(n/100) + O(n)$$

1st level, $= n$

IInd level, $= 99n/100 + n/100 = n$

So ~~complexity~~ 2d remains same for any kind of partition.

$\therefore$ if we take lager branch $= O(n \log_{100/99} n)$

for, shorter branch $= \Omega(n \log_{10} n)$

either way base complexity of $O(n \log n)$ remains.

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

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

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

Q.19

```
Linear_search ( Array, size, key, flag )
Begin
    for(i=0 to n-1) by 1 do
        if (Array[i] = key)
            set flag = 1
            Break

    if flag = 1
        return flag
    else
        return -1

End.
```

A.20

Iterative

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

Recursive

```
Insertion (int a[], int i,
           int n)
{
  int val = a[i], j = i
  while (j>0 && a[j-1]> va
  {
    a[j] = a[j-1];
    j--;
  }
  a[j] = val;
  if (i+1 <= n)
    insertion (a, i+1, n
}
```

Because online algo doesnot know the whole input.

A.21

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

PAGE No.
DATE

A.22. Bubble sort, insertion sort and selection sort are inplace sorting algo.

Bubble and insertion sort can be applied as stable algo but selection sort cannot.

Merge sort is a stable algo but not an in place algo.

Quicksort is not stable but is an inplace algo.

Heap sort is an inplace algo but is not stable.

A.23
```
int binary ( int [] A , int x)
{
    int low = 0, high = A.length - 1;
    while ( low <= high )
    {
        int mid = (low+high) /2 ;
        if (x == A[mid]) return mid;
        else if (x < A[mid])
            high = mid-1;
        else
            low = mid +1;
    return -1;
}
```

A.24 $T(n) = T(n/2)+1$