DAA - Assignment.

Ans 1. Asymptotic notations are the languages that allow us to analyse an algorithm running time by identifying its behaviour as the input size increases.

Types :

1. Theta (Θ) : Avg. value.
2. Big Oh (O) : Worst case, Upper bound
3. Omega (Ω) : lower bound, Best case.

Ans 2. $T(n) = O(\log n)$

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

Using,
Subtract and Conquer Theorem :

$$T(n) = 3T(n-1) \longrightarrow (1)$$
$$T(n) = a\,T(n-b) + f(x) \longrightarrow (2)$$

Comparing (1) and (2)

$$a=3 \; ; \; b=1 \; ; \; f(x)=0 \; ; \; k=0$$

$$T(n) = O\left(n^k \sum_{i=0}^{n/i} a^i\right)$$

$$= O\left(n^0 \sum_{i=0}^{n/i} a^i\right)$$

$$T(n) = O(3^{n/i})$$

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

Page No.
Date

Ans4 $T(n) = 2T(n-1) - 1$

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

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

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

$\vdots$

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

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

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

Ans 5.

$\rightarrow 1 + 3 + 6 + 10 + \ldots + k = n$

$= \frac{k(k+1)(k+2)}{6} = n$

$O(k^3) = n$

$\boxed{\therefore k = \sqrt[3]{n}}$

Ans 6. $O(\sqrt{n})$

Ans 7. i n/2 Complexity $\rightarrow \frac{n}{2} \cdot \log n \cdot \log n$

j log n

k log n $\rightarrow O(n \log^2(n))$

Ans 8 Outermost loop = n/3

i^{th} loop = n

j^{th} loop = n

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

Ans 9. $i \rightarrow 0, 1, 2 \ldots n$

$j \rightarrow n, n/2, n/3 \ldots \rightarrow \frac{n/n}{\log n}$

Complexity = $O(n \log n)$

Ans 10. Since polynomials grow slower than exponentials n^k has an asymptomatic upper bound of $O(a^n)$ for $a = 2, n = 2$

Ans 11. $j \rightarrow 2, 3, 4, 5$

$i \rightarrow 1, 3, 6$

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

$$\therefore k^2 = n$$

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

Complexity = $\sqrt{n}$.

Ans 12. $T(0) = 0$

$T(1) = 0$

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

let

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

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

Using backward Sol^n:

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

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

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

$T(n) = 2.2.2\,T(n-3)+1+1+1$
$= 8T(n-3)+3$

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

$T(0) = 0$

$n-k = 0$

$n = k$

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

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

Ans 13. (i) $n\log n$ →

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

(ii) n^3 →

```
for (i=1; i<n; i++) {
    for (j=1; j<n; j++ {
        for (k=1; j<n; k++){
            // some code
}}}
```

(iii) $\log(\log n)$ →

```
void fun (int n) {
for (i=n; i>1 ; i = fun(i,n))
{ // some O(1) task.
}
```

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

Assume that ;

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

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

$c = \log_b a$

$= \log_2 2 = 1$

$\therefore n^2 < t(n)$

Complexity = $\Theta(n^2)$

Ans 15. $i = 1, 2, 3, \ldots, n$

$j = n, n/2, n/3$

Complexity = $O(n \log n)$

Ans 16. $2, 2^k, (2^k)^k, ((2^k)^k)^k = 2^{k^3} - 2^{k \log k (\log n)}$

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

$2^{\log n} = n$

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

Ans 17. $T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + O(n)$

Taking 99% & 1% other

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

1st level = n

2nd level = $\frac{99n}{100} + \frac{n}{100} \rightarrow n$

→ So, It remains same for any kind of partition

∴ If we take longer branch = $O(n \log_{\frac{100}{99}} n)$

Shorter branch = $\Omega(n \log_{10} n)$

Ans 18

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

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

Ans 19.

```
int linear (int *a, int n, int key) {
    for (i=0 ; i<n-1; i++) {
        if (*(a+i) == key)
            return i;
    }
    return -1;
}
```

Ans 20. Iterative :

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

Reccursive :

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

Ans 21.

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

Ans 22. (i) Bubble sort, Insertion sort and selection sort are inplace sorting algo

(ii) Bubble and Insertion sort can be applied on stable data but selection sort can't.

(iii) Merge sort is a stable algo but not in-place.

(iv) QuickSort is not stable but inplace algo.

(v) Heap Sort is an inplace but not stable.

Ans 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;
```

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