Name - Saakshi
Uni. Roll No - 2014820
Section - A
Class Roll No - 44

DAA Assignment 1

Ans 1

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

Different Asymptotic notations.

1.) Big Oh (O)

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

($f(n)$ can never go beyond $g(n)$)

$g(n)$ is "tight" upper bound of $f(n)$.

2.) Big Omega (Ω)

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

$g(n)$ is "tight" lower bound of $f(n)$.

3.) Theta (θ)

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

It gives tight upper & lower bound both.

4.) Small Oh (O).

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

o gives us upper bound.

5.) Small Omega (ω)

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

Ans

2- i) 1, 2, 4, 8 ... n

$a = 1 \quad r = 2$

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

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

$2n = 2^k$

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

$k = \log_2(n) + 1$

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

Ans

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

using backward sub

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

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

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

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

$\vdots$

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

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

$\because T(0) = 1$

Complexity $= O(3^n)$

Ans

4- $T(n) = 2T(n-1) - 1 \quad n > 0$, else 1

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

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

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

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

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

$\vdots$

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

③

$\because T(0)=1$

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

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

$\therefore$ Complexity $\Rightarrow O(1)$

Ans 5-

$i = 1, 3, 6, 10 \ldots n$

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

$K^2 = n$

$K = \sqrt{n}$

$\therefore$ Complexity $= O(\sqrt{n})$

Ans 6-

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

Ans 7-

Loops	i	j	K
	$n/2$	$\log n$	$\log n$

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

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

Ans 8-

Outermost loop	i	j
$\downarrow$	$\downarrow$	$\downarrow$
$n/3$	n	n

Complexity $\Rightarrow n^3$

Ans 9-

i	j
1	n times
2	n/2 times
3	n/3 times
⋮	⋮
$\frac{n}{n}$	$\frac{n/n \text{ times}}{\log n}$

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 $a=2$, $n_0=2$

Ans 11

j	i
2	1
3	3
4	6
5	10
	⋮

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

$$K^2 = n$$

$$K = \sqrt{n}$$

∴ Complexity $= \sqrt{n}$

Ans 12

$T(0) = 0 \qquad T(1) = 0$

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

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

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

using backward soln

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

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

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

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

$= 8\,T(n-3)$

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

$T(0) = 0$

$n-k = 0$

$n = k$

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

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

Complexity $\Rightarrow O(2^n)$

Ans 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 (6)

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

log log(n)

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

Ans

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

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 $\theta(n^2)$

(7)

Ans 15

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

$\log n$

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

Ans 16. i takes $2, 2^k, (2^k)^k, (2^k)^k - 2^{k^3} \ldots 2^{k \lg k (\log(n))}$

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

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

Hence, time complexity

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

Ans 17 $T(n) = T(9n/10) + T(n/10) + O(n)$

taking one branch 99% and other 1%

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

1^{st} level, $= n$

II^{nd} level, $= 99n/100 + n/100 = n$

So it remains same for any kind of partition.

$\therefore$ if we take longer 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

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

c.) $1 < \log_2 n < \log n! < n\log_2 n < n\log_6 n < 5n < n! < 8n^2 < 7n^3 < 8^{n2n}$

Ans 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.
```

Ans 20

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

~~Best~~ When input not known.

Ans 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	$\Omega(n \lg n)$	$\theta(n \lg n)$	$O(n^2)$
Merge	$\Omega(n \lg n)$	$\theta(n \lg n)$	$O(n \lg n)$

Ans 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 ~~ver~~ an in place algo.

Quicksort is not stable but in an inplace algo.

Heap sort is an inplace algo but is ~~also~~ 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.) $T(n) = T(n/2)+1$