Name → Naitik Singhal
Sec/Roll no → ML / 39

DAA Assignment - 1

Q1 → * Asymptotic notations are mathematical tools used for analysis of algorithms that describe their

* They provide a way to express the time and space complexity of an algorithm

1) Big O: Represents the upper bound of an algo.
ex) → If an algo has time complexity of $O(n^2)$ it means its worst-case running time grows quadratically ~~linearly~~ with input size

2) Omega Ω : Represents the lower bound of algo
ex) → If an algo has time complexity of $\Omega(n)$ it means that it's best-case running time grows linearly with input size

3) Theta notation Θ → Represents range of both upper and lower bound
ex) → If an algo has a time complexity of $\Theta(n)$, it means its running time grows linearly with input size, both best and worst cases

Q2 → for (i=1 to n) — 1, 2, 4, 8, 16, -, - $\frac{n}{2}$, n
{ i = i*2;
}

→ $2^K > n$

Taking log on both sides

$K \geq \log_2 (n)$

Complexity ⇒ $O(\log_2 n)$

Q3 → $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

So, $T(0) = 1$

$T(1) = 3T(0) = 3(1) = 3$

$T(2) = 3T(1) = 3(3) = 9$

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

Verify → $T(n) = 3T(n-1) = 3 \cdot 3^{n-1} = 3^n$

Complexity → $O(3^n)$

Q4 → $T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

So, $T(0) = 1$

$T(1) = 2T(1-1) - 1 \Rightarrow 2T(0) - 1$
$\Rightarrow 2(1) - 1$
$\Rightarrow 2 - 1 \Rightarrow 1$

$T(2) = 2T(2-1) - 1 \Rightarrow 2T(1) - 1$
$\Rightarrow 1$

$T(3) = 2T(3-1) - 1 \Rightarrow 2T(2) - 1$
$\Rightarrow 1$

So, $T(n) = 1$

Complexity → $O(1)$

Q5 →
```
int i = 1, s = 1;   — 1
while (s <= n)
{
    i++;
    s = s+1;
    print ("#");
}
```

$\left.\begin{array}{l} i = 1, 2, 3, 4, 5 \; --- \\ s = 1, 3, 6, 10, 15 \; --- \; n \end{array}\right\} \rightarrow s = \frac{i(i+1)}{2}$

$$\frac{i(i+1)}{2} \leq n$$

$$i(i+1) \leq 2n$$

$$i^2 + i - 2n \leq 0$$

$$\therefore \; i < \frac{-1 + \sqrt{1+8n}}{2}$$

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

Q6 → void function (int n)

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

$i = 1, 2, 3, 4 \ldots \ldots \sqrt{n}$

$i^2 = 1, 4, 9, 16 \ldots \ldots \ldots n^2$

Complexity → $O(\sqrt{n})$

Q7→ void function (int n)

{

int i, j, k, count=0;

$n/2 \leftarrow$ for (i= n/2; i<=n; i++)

{

$\log_2 n \leftarrow$ for(j=1; j<=n; j=j*2)

{

$\log_2 n \leftarrow$ for (k=1; k<=n; k=k*2)

{ count ++;

}

}

}

}

So, $\frac{n}{2} \propto \log_2(n) \propto \log_2(n)$

complexity → $O(n \log^2(n))$

Q8→ function (int n) ——— $T(n)$

{ if (n= =1) return;

for (i=1 to n){ → n

for (j=1 to n){ → n^2

print ("*"); → n^2

}

} function (n-3); } → $T(n-3)$

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

So, Time complexity $\rightarrow$ $O(n^2)$

Q9 $\rightarrow$

```
Void function (int n)
{
  for (i=1 to N)
  {
    for( j=1; j<=n; j=j+i)
    { print ("*");
    }
  }
```

$i = 1, 2, 3, 4 ---$

$j = 1, 3, 6, 10 ---$

$i = 1$ ---- n times

$i = 2$ ---- $n/2$ times

$i = 3$ ---- $n/3$ times

$1 + \frac{n}{2} + \frac{n}{3}$ ------ $+1$

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

Q10 → n^k $(k \geq 1)$ $c^n (c > 1)$

C^n grows faster than n^k