Design and Analysis of Algorithm (DAA)

Que: 1:

```
main ()
{
    for (i=1; i<=n; i++)          → n steps
    {
        printf (" unacademy");    → n times
    }
```

Time complexity O(n)

Que: 2:

```
main ()
{
                                  s.p      destination
    for(i=n/2; i<=n; i++ )     → n/2       n
    {
        printf (" Sandeep Kumar"); → n/2 times
```

$O(n/2)$

Time Complexity => O(n)

Que: 3:

```
main ()
{   int n;                        Source
{                                   n
    for(n > 0)        }
    {  n = n/2        }  infinite
    }
}
```

Que: 4:-

```
main()
{ int n;
  for(n>1)
    { n = n/2
    }
}
```

n	2^3	2^4	2^{100}	2^k
loop executes	3	4	100	k / $\log n$

$1, n/2, n/2^2, n/2^3 \cdots 1$

2 divides

ex: ① $2^3 > 1$ → inside the loop

② $2^2 > 1$ → inside the loop

③ $2^1 > 1$ → inside the loop

$1 > 1$ → false

→ 3 times

$n = 2^k$

$\log_2 n = k$

$k = \log_2 n$

Time complexity = $O(\log_2 n)$

Que: 5:

```
f()
{ for(i=1; i<=n; i=2*i)
    { printf("Gate");
    }
}
```

Source	destination	
1	n	n/2
n	1	

$1, 2, 2^2, \ldots n = 2^k$

2 multiply, k steps

$\log_2 n = k$

Time complexity = $O(\log_2 n)$

Que: 6:- f()

```
{
    for(i=1; i<=n; i=3i)
    {   print (" sandeep");
    }
}
```

source	destination
1	n

$1, 1\times3, 3^2, 3^3 \ldots 3^k = n$

$n = 3^k$

$\log_3 n = k$

$K = \log_3 n$

Time complexity $= O(\log_3 n)$

Que: 7:- f()

```
{ int n = 2^(2^k);
  int j = 2;
  for(i = 1 -> n)        -> n
  { while (j <= n)       -> k
      { j = j^2
      }
  }
}
```

n=2	2^1	2^2	2^{2^2}	2^{2^3}	2^{2^4}
i					
inner while					

k	1	2	... 1000	k
n	2^{2^1}	2^{2^2}		2^{2^k}
inner while	2	3	1001	(k+1)

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

$$\log(\log_2 n) = k$$

$$k = \log(\log_2 n) \Rightarrow \log(\log_2 n + 1)$$

Time complexity = ~~$O(n \log(\log_2 n))$~~

Time complexity = $O(n \log(\log_2 n))$

Que: main()

```
{ int i, j
  for(i=1, i<=n; i++)  ⟶  n
  { for (j=1; j<=i; j++) ⟶ n*n/2+3+n
    { printf("Sandeep")
    }
  }
}
```

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

$$= O(n^2)$$

Ques

```
f()
{
  int n;
  j = 2;
  for (i = 2; i <= n; i = i^2)  -> log (log n)
  {
    for (j = 1; j <= n; j = 2*j)  -> 2^0, 2^1, 2^2, ...
    {
      printf("sandeep");
    }
  }
}
```

i → 2^k

$2^k = n$

$k = \log_2 n$

Inner loop

Source → Destination

$i = 2, 2^2, 2^{2^2}, 2^{2^3} \ldots 2^{2^k}$

$k = \log(\log_2 n)$

Time complexity $= O(\log n * (\log(\log_2 n)))$