DSA-LAB 4

Time Complexity

Question 1 (Estimated Time 20 mins) (Each proof contains 5 points)

Prove all the thetas using (halving technique or any way you like). Both find out its lower(Omega) and upper(Big O) bound by the same function so that you can declare it to be theta of the given function.

1. $1^2 + 2^2 + 3^2 + 4^2 + ... + N^2$ -- $\Theta(N^3)$

   Number of terms = N

   Upper Bound:

   $1^2 + 2^2 + 3^2 + 4^2 + ... + N^2 \quad <= N^2 + N^2 + N^2 + N^2 + ... + N^2$ (replacing each term with largest term N^2)

   $= N(N^2)$

   $= N^3$

   $= O(N^3)$

   Lower Bound:

   $1^2 + 2^2 + 3^2 + 4^2 + ... + N^2 \quad >= (N/2)^2 + (N/2+1)^2 + (N/2+2)^2 + (N/2+3)^2 + ... + (N-1)^2 + N^2$ (removing first half terms i.e., first N/2 terms from the series. Number of remaining terms is N/2)

   $>= (N/2)^2 + (N/2)^2 + (N/2)^2 + ... + (N/2)^2$ (replacing every term with the least term which is $(N/2)^2$)

   $= (N/2) * (N/2)^2 = N^3/8 = \Omega(N^3)$

2. $1 + 2 + 3 + 4 + ... + N^2$ -- $\Theta(N^4)$

   Number of terms = N^2

   Upper Bound:

   $1 + 2 + 3 + 4 + ... + N^2 \quad <= N^2 + N^2 + N^2 + N^2 + ... + N^2$ (replacing each term with largest term N^2)

   $= N^2(N^2)$

   $= N^4$

   $= O(N^4)$

   Lower Bound:

   $1 + 2 + 3 + 4 + ... + N^2 \quad >= (N^2/2) + (N^2/2)+1 + (N^2/2)+2 + (N^2/2)+3 + ...+ N^2-1 + N^2$ (removing first half terms i.e., first $N^2/2$ terms from the series. Number of remaining terms is $N^2/2$)

   $>= (N^2/2) + (N^2/2) + (N^2/2) + (N^2/2) + ... + (N^2/2)$ (replacing every term with the least term which is $(N^2/2)$)

   $= (N^2/2)*(N^2/2) = N^4/4 = \Omega(N^4)$

3. $1 + 3 + 5 + 7 + 9 + \ldots\ldots + (2N + 1)$ -- $\Theta(N^2)$

Number of terms = N+1

$1 + 3 + 5 + 7 + 9 + \ldots\ldots + (2N + 1) = (N+1) + (0 + 2 + 4 + 6 + 8 + \ldots\ldots + (2N))$ (taking 1 away from each term and grouping them as (N+1) as total terms are N+1)

$= (N+1) + 2(0+1+2+3+4+\ldots+N)$ **(taking 2 common)**

$= \Theta(N) + \Theta(N^2) = \Theta(N^2)$

4. $2 + 4 + 6 + 8 + \ldots\ldots + 2N$ -- $\Theta(N^2)$

$2(1+2+3+4+\ldots+N)$**(taking 2 common)**

$2\Theta(N^2) = \Theta(N^2)$

5. $1 + 2 + 3 + 4 + \ldots\ldots + (N/2)$ -- $\Theta(N^2)$

Number of terms = N/2

Upper Bound:

$1 + 2 + 3 + 4 + \ldots\ldots + (N/2) \quad <= (N/2) + (N/2) + (N/2) + (N/2) + \ldots\ldots + (N/2)$ (replacing each term with largest term N/2)

$= N/2 * N/2$

$= N^2/4$

$= O(N^2)$

Lower Bound:

$1 + 2 + 3 + 4 + \ldots\ldots + (N/2) \quad >= (N/4) + (N/4)+1 + (N/4)+2 + \ldots + (N/2)-1 + N/2$(removing first half terms i.e., first N/4 terms from the series. Number of remaining terms is N/4)

$>= N/4 + N/4 + N/4 + \ldots\ldots + N/4$ (replacing every term with the least term which is (N/4))

$= (N/4) * (N/4) = N^2/16 = \Omega(N^2)$

6. $1+2+4+8+16+ \ldots + N^2$ ---- $\Theta(N^2)$

Upper Bound:

Method 1:

Writing it backward this is geometric series (ar^{n-1}) :

$N^2 + N^2/2 + N^2/4 + \ldots\ldots + 4 + 2 + 1$

With $a = N^2$ and $r = ½$

Formula for sum of geometric series to infinity = a/1-r

$N^2 + N^2/2 + N^2/4 + \ldots + 4 + 2 + 1 <= N^2 + N^2/2 + N^2/4 + \ldots + 4 + 2 + 1 + ½ + ¼ + ⅛ + \ldots 0$ (red part is included in sum upto infinity)

$N^2 + N^2/2 + N^2/4 + \ldots + 4 + 2 + 1 <= N^2/(1-½)$ (using formula given above)

$N^2 + N^2/2 + N^2/4 + \ldots + 4 + 2 + 1 <= N^2 * 2 = O(N^2)$

Method 2:

The above series is a geometric series. By the proof of geometric series, we know that summation of geometric series <= 2(the largest term in the series). In our case, summation of geometric series<=$2N^2$. So this is $O(n^2)$.

Lower Bound:

$N^2 + N^2/2 + N^2/4 + \ldots + 4 + 2 + 1 >= N^2$ (removing all values except the largest value which is N^2)

$= \Omega(N^2)$

1) What is the algorithm's complexity of the following piece of code - Sample Solution is in RED.

int Sum=0; // O(1) Time
for(int i=0; i<N; i++) //(1+1+1+...+1 - - - N Times =O(N) for(int j=0; j<N; j++) Sum++;
// (1+1+1+...+1) + (1+1 +... +1)+... + (1+1 +... +1) added N times // N + N +... + N = $O(N^2)$
Overall Complexity : O(1) +O(N) + $O(N^2)$ + $O(N^2)$ = $O(N^2)$

2) What is the algorithm's complexity of the following piece of code

int Sum=0; // O(1) Time
for(int i=0; i<N; i++)//(1+1+1+...+1 - - - N Times =O(N) Sum++; // O(N)
for(int j=0; j<N; j++)//(1+1+1+...+1 - - - N Times =O(N) Sum++; // O(N)

Overall Complexity : O(1) +O(N) + O(N) + O(N) + O(N) = O(N)

3)
What is the algorithm's complexity of the following piece of code

int Sum=0; // O(1)
for(int i=0; i<N; i++)//(1+1+1+...+1 - - - N Times =O(N)
for(int j=0; j<N; j++)//(N+N+N+...+N - - - N Times =$O(N^2)$ for(int k=0; k<N; k++)//(N^2+ N^2+ N^2+...+ N^2- - - N Times =$O(N^3)$
Sum++;//$O(N^3)$

for(int i=0; i<N; i++)//(1+1+1+...+1 - - - N Times =O(N)
for(int j=0; j<N; j++)//(N+N+N+...+N - - - N Times =$O(N^2)$ for(int k=0; k<N; k++)//(N^2+ N^2+ N^2+...+ N^2- - - N Times =$O(N^3)$ Sum++;//$O(N^3)$

Overall Complexity = $O(N^3)$

4)
What is the algorithm's complexity of the following piece of code

int Sum=0; // O(1)
for(int i=0; i<N; i++)//(1+1+1+...+1 - - - N Times =O(N) Sum++;// O(N)
for(int j=0; j<N; j++)//(1+1+1+...+1 - - - N Times =O(N) Sum++;// O(N)
for(int k=0; k<N; k++)//(1+1+1+...+1 - - - N Times =O(N) Sum++;// O(N)
for(int m=0; m<N; m++)//(1+1+1+...+1 - - - N Times =O(N) Sum++; // O(N)
for(int n=0; n<N; n++)//(1+1+1+...+1 - - - N Times =O(N) Sum++;// O(N)
for(int p=0; p<N; p++)//(1+1+1+...+1 - - - N Times =O(N) Sum++;// O(N)

Overall Complexity = O(N)

5)
int Sum=0; // O(1)
for(int i=0; i<N; i++)//(1+1+1+...+1 - - - N Times =O(N)
for(int j=0; j<i; j++)//values of i: 1, 2, 3, 4, 5,.....N (arithmetic series) 1+2+3+4+…+N = $O(N^2)$
for Ω, $(\frac{N}{2}) + (\frac{N}{2}+ 1) + (\frac{N}{2}+ 2) + \cdots + N$ (N⁄2 Terms) $= (\frac{N}{2}) + (\frac{N}{2}) + (\frac{N}{2}) + \cdots + (\frac{N}{2})$
$= \frac{N}{2} \cdot n = \Omega(N^2)$
for(int k=0; k<j; k++)//N+N+N+…+N (N^2terms) = $O(N^3)$
for Ω, $(\frac{N}{2}) + (\frac{N}{2}+ 1) + (\frac{N}{2}+ 2) + \cdots + N$ (N^2⁄2 Terms) $= (\frac{N}{2}) + (\frac{N}{2}) + (\frac{N}{2}) + \cdots + (\frac{N}{2})$
$= \frac{N^2}{2} \cdot \frac{N}{2} = \Omega(N^3)$
Sum++; //$O(N^3)$

Overall complexity = $\Theta(N^3)$

6)
int Sum=0; // O(1)
for(int i=0; i<N; i+=2) //(1+1+1+...+1- - - N⁄2 Terms =O(N)
for(int j=0; j<i; j+=2) //values of i: 0, 2, 4, 6,...N
2(0+1+2+3+…+N - - - N terms) = $O(N^2)$ for Ω, $(\frac{N}{2}) + (\frac{N}{2}+ 1) + (\frac{N}{2}+ 2) + \cdots + N$ (N Terms) $= (\frac{N}{2}) + (\frac{N}{2}) + (\frac{N}{2}) + \cdots + (\frac{N}{2})$
$= N \cdot N = \Omega(N^2)$
for(int k=0; k<j; k+=2) //N+N+N+…+N (N^2/2 terms) = $O(N^3)$ for Ω, $(\frac{N}{2}) + (\frac{N}{2}+ 1) + (\frac{N}{2}+ 2) + \cdots + N$ (N^2 Terms) $= (\frac{N}{2}) + (\frac{N}{2}) + (\frac{N}{2}) + \cdots + (\frac{N}{2})$
$= N^2 \cdot N = \Omega(N^2)$
Sum++; //$O(N^3)$

Overall complexity = $\Theta(N^3)$

7
int Sum=0; // O(1)
for(int i=1; i<N; i*=2) //values of i: 1, 2, 4, 8, 16,.....N (geometric series) = 1+1+1+1+1 - - - $\log_2 N$ terms = O(logN)
for(int j=1; j<N; j*=2)
= (1+1+1+…+1) + (1+1+1+…+1) + (….) +…. + (….) = log N + logN + log N + ⋯ + logN (log N terms) = logN · logN
= $O(\log^2 N)$
Sum++; //$O(\log^2 N)$

8
int Sum=0; // O(1)
for(int i=1; i<N; i*=2) //1, 2, 4, 8, 16,.....N (geometric series) = 1+1+1+1+1 - - - $\log_2 N$ terms = O(log N)
Sum++; //O(log N)
for(int j=1; j<N; j*=2) //1, 2, 4, 8, 16,.....N (geometric series) = 1+1+1+1+1 - - - $\log_2 N$ terms = O(log N)
Sum++; //O(log N)

Overall complexity = O($\log^2$N)	Overall complexity = O(log N)

9
for(int i=1; i<=N*N; i+=2)//values of i: 1, 3, 5, 7,…,N^2(arthimetic series) = (1+1+1+…+1) - - - (N^2/2 Terms) = O(N^2)
 	
for(int j=1; j<N*N; j*=2)
//values of j:1, 2, 4, 8,…, N^2(geometric series) = (1+1+1+…+1) + (1+1+1+…+1) + (….) +…. + (….) = log N^2 + log N^2 + log N^2 + ⋯ + log N^2(N^2terms) = N^2· logN^2
= O(N^2log N)
 	
Sum++; //O(N^2logN)
 	
Overall complexity = O(N^2log N)
	10
	for(int i=1; i<=N*N; i+=2) //values of i: 1, 3, 5, 7,…,N^2 = (1+1+1+…+1) - - - (N^2½ Terms) = O(N^2)
	Sum++; //O(N^2)
	
	for(int j=1; j<N*N; j*=2)
	//values of j:1, 2, 4, 8,…, N^2(geometric series)
	= (1+1+1+ … +1) log N^2
	= O(logN)
	Sum++; //O(logN)
	
	Overall complexity = O(N^2)

11
for(int i=1; i<=N*N; i*=2) //O(logN)
for(int j=1; j<N*N; j*=2)
// = log N + logN + log N + ⋯ + log N (log N terms) = logN · logN
= O($\log^2$N)
 	
Sum++; //O($\log^2$N)
 	
Overall complexity = O($\log^2$N)
	12
	for(int i=1; i<=N*N; i*=2) //O(logN)
	Sum++; //O(logN)
	
	for(int j=1; j<N*N; j*=2) //O(logN)
	Sum++; //O(logN)
	
	Overall complexity = O(logN)

13
int Sum=0; //O(1)
for(int i=1; i<=N; i*=2) //O(logN)
for(int j=1; j<=N; j*=2)
= (1+1+1+…+1) + (1+1+1+…+1) + (….) +…. + (….) = log N + logN + log N + ⋯ + logN (log N terms) = logN · logN
= O($\log^2$N)
for(int k=1; k<=N; k*=2)
= (1+1+1+…+1) + (1+1+1+…+1) + (….) +…. + (….) = log N + logN + log N + ⋯ + logN ($\log^2$N terms) = logN · $\log^2$N
= O($\log^3$N)
Sum++; //O($\log^3$N)
 	
Overall complexity = O($\log^3$N)
	14
	int Sum=0; //O(1)
	for(int i=1; i<=N; i*=2) //O(logN)
	Sum++;//O(logN)
	for(int j=1; j<=N; j*=2) //O(logN)
	Sum++; //O(logN)
	for(int k=1; k<=N; k*=2) //O(logN)
	Sum++;//O(logN)
	
	Overall complexity = O(logN)

15
int sum,i,j; //O(1)
sum = 0; //O(1)
for (i=1;i<n;i=i*2) //O(log N)
{
for (j=0;j<n;++j) {
= (1+1+1+…+1) + (1+1+1+…+1) + (….) +…. + (….) = n + n + n + ⋯ + n (log n terms)
= n · log n
= O(n log n)
sum++; //O(n log n)
}
}
 	
Overall complexity = O(n log n)
	16
	<u>BE CAREFUL GEOMETRIC SERIES</u>
	int sum,i,j; //O(1)
	sum = 0; //O(1)
	for (i=1; i<n; i=i*2) //O(logN)
	{
	for (j=0; j <i ; ++j) { // values of i: 1, 2, 4, 8, 16, …, n
	= (1) + (1+1) + (1+1+1+1) + …. + (1+1+1+…+1 - - - n terms) = 1 + 2+ 4+ 8+ 16 + ….. + n (geometric series) = O(n)
	sum++; //O(n)
	}
	}
	
	Overall complexity = O(n)

17
<u>BE CAREFUL GEOMETRIC SERIES</u>
int sum,i,j; //O(1)
sum = 0; //O(1)
for (i=1; i<n; i=i*5) { // 1, 5, 25, 125, …., n (geometric sequence) = (1+1+1+….+1) - - - $\log_5 n$ terms
= O(log n)
for (j=0; j<i; j+=2){ // values of i: 1, 5, 25, 125, …, n
= (1) + (1+1+1+1+1) + …. + (1+1+1+…+1 - - - n terms) = 1 + 5 + 25 + 125 +….. + n - - - n/2 terms
= 2n · n/2
= O(n)
sum++; // O(n)
}
}
Overall complexity = O(n)

18
int sum,i,j; //O(1)
sum = 0; //O(1)
for (i=1; i<n; i=i*4){ // 1, 4, 16, 64, …., n (geometric sequence) = (1+1+1+….+1) - - - $\log_4 n$ terms = O(log n)
for (j=0 ; j<n ; j+=3){
= (1+1+1+….+1 - - - n/3 terms) log n times = n · log n
= O(n log n)
sum++; // O(n log n)
}
}

Overall complexity = O(n log n)

19 What will be the output (the value of **Sum**) of the program asymptotically in BIG-O notation, I am not asking here the complexity of loop rather the asymptotic bound on the value of Sum:

int Sum = 0;
for(int i=1; i<=n; i+=1)
{
Sum+=i; // values of i are being added
= 1+2+3+4+5+…..+n (n terms) (arithmetic series) Replacing all values with greatest term for Big O = n + n + n + … + n (n terms)
= n · n
= $O(n^2)$
}
cout<<Sum<<endl;

sum ≤ $O(n^2)$

20 What will be the output(the value of **Sum**) of the program asymptotically in BIG-O notation:

int Sum = 0;
for(int i=1; i<=n; i*=2)
{
Sum+=i; // values of i are being added
= 1+2+4+8+16+…..+n/2+n (geometric series) = O(n)
}
cout<<Sum<<endl;

sum ≤ O(n)

21 What is the time complexity of the algorithm:

int Sum = 0;//O(1)
for(int i=1; i<=n; i+=1) {//O(n)
for(int j=1; j<=i; j++){ // values of i: 1, 2, 3, 4,…., n
= (1) + (1+1) + (1+1+1) + …. + (1+1+1+…+1 - - - n terms) = 1 + 2+ 3+ 4 + ….. + n (arithmetic series)
= $O(n^2)$

for Ω, $(\frac{n}{2}) + (\frac{n}{2}+1) + (\frac{n}{2}+2) + \cdots + n$ (n/2 Terms)
$= (\frac{n}{2}) + (\frac{n}{2}) + (\frac{n}{2}) + \cdots + (\frac{n}{2})$
$= \frac{n}{2} \cdot n = \Omega(n^2)$

Sum++; //$O(n^2)$
}
}
cout<<Sum<<endl;

Overall complexity = $\Theta(N^2)$

22 What is the time complexity of the algorithm:

int Sum = 0; //O(1)
for(int i=1; i<n; i*=2){// 1, 2, 4, 8, …, n (geometric sequence) = (1+1+1+…..+1) - - - $\log_2 n$ terms = O(log n)
for(int j=1; j<=i; j++){// values of i: 1, 2, 4, 8, 16, …, n
= (1) + (1+1) + (1+1+1+1) + …. + (1+1+1+…+1 - - - n terms) = 1 + 2+ 4+ 8+ 16 + ….. + n (geometric series) = O(n)
Sum++; //O(n)
}
}
cout<<Sum<<endl;
}

Overall complexity = O(n)

40 Complexity of primeNumber function.

```
int sqrt(int N)
{
int d; //O(1)
for(d=0; d*d<=N; d++) { } // d ≤ √N
= (1+1+1+….+1) - - - √N terms   = O(√N)
return d-1;
}
bool primeNumber(int n)
{
        bool isPrime = true; //O(1)
        int lmt = (sqrt(n)); //O(√N )
        for (int d=2; d <=lmt ;++d) { // lmt = √N
= (1+1+1+….+1) - - - √N terms   = O(N)
                if (n%d==0) //O(√N )
                        return false;
        }
        return true;
}
```

Overall complexity = $O(\sqrt{n})$

41 Complexity of primeNumber function.

```
int sqrt(int N)
{
int d; //O(1)
for(d=0; d*d<=N; d++){ } // d ≤ √N
= (1+1+1+….+1) - - - √N terms   = O(N)
return d-1;
}
bool primeNumber(int n)
{
        bool isPrime = true;
        for (int d=2; d <= sqrt(n) ;++d)
        //( √N + √N + √N + ⋯ + √N ) - - - √N
        terms  = √N · √N
         = O(N)
        {
                if (n%d==0) //O(N)
return false;
        }
        return true;
}
```

Overall complexity = $O(n)$

23 What is the time complexity of the algorithm:

```
int f1(int n)
{
        int K=0; //O(1)
        for(int j=0; j*j<=n*n; j++) K++; //O(n)
        return K;
}
int main()
{
        int Sum = 0, n; //O(1)
        cin>>n; //O(1)
for(int i=1; i<=f1(n); i+=1) // value returned by fl(n) =
n
        = 1+1+1+…+1 - - - n times
        = O(n)
        for(int j=1; j<=i; j++) Sum++;
        // = 1+2+3+4+….+n - - - n times
        = n · n
 = O(n^2)
        cout<<Sum<<endl;
}
```

Overall complexity = $O(n^2)$

24 What is the time complexity of the algorithm:

```
int f1(int n)
{
  int K=0; //O(1)
for(int j=1; j*j<=n; j*=2)
K++; // j ≤ √n
1, 2, 4, 8,…., √n
= (1+1+1+….+1) - - - log_2 √n terms  = O(log
n)
return K; // returns log √n
}
int main()
{
int Sum = 0; //O(1)
int n; //O(1)
cin>>n; //O(1)
for(int i=1; i<=;f1(n) i+=1)//function with complexity of
O(log n) called O(log n) times, so O(log^2 n)
      for(int j=1; j<=i; j++)//1+2+3+4+...+log √n=O(log
√n)^2=O(log^2 n)
Sum++;
cout<<Sum<<endl;
}
```

Overall complexity = $O(\log^2 n)+O(\log^2 n)=O(\log^2 n)$

25 What is the time complexity of the algorithm:

int f1(int n)
{ int K=0;//O(1)
for(int j=1; j*j<=n; j++) K++; // j ≤ √n
= (1+1+1+….+1) - - - √n terms   = O(√n)
return K*K;
}
int main()
{
int Sum = 0; //O(1)
int n; //O(1)
cin>>n; //O(1)
int Terminator = f1(n); // O(√n)
for(int i=1; i<= Terminator; i+=1) {
// value of terminator = √n · √n = n
= (1+1+1+…+1) n terms
= O(n)
for(int j=1; j<=i; j++){ // values of I: 1, 3, 4, … n
= (1+2+3+4+…..+n) n terms
= n · n
= $O(n^2)$

Sum++; // $O(n^2)$
}
}
cout<<Sum<<endl;
}

Overall complexity = $O(n^2)$

26 What is the time complexity of the algorithm:

int f1(int n){
int K=0; //O(1)
for(int j=0; j*j<=n; j++) K++; // j ≤ √n
= (1+1+1+….+1) - - - √n terms   = O(√n)
return K;
}
int main()
{
int Sum = 0; //O(1)
int n; //O(1)
cin>>n; //O(1)
int Terminator = f1(n); // O(√n)
for(int i=1; i<=Terminator; i+=1){
// value of terminator = √n
= (1+2+3+4+…..+√n) - - - √n terms   = √n · √n
= O(n)
for(int j=1; j<=i; j++) {// values of I: 1, 2, 3, 4,…, √n
= (1+2+3+4+…..+√n) - - - √n terms   = √n · √n
= O(n)

Sum++;// O(n)
}
}
cout<<Sum<<endl;
}

Overall complexity = O(n)

27
for (i=1;i<n;i=i*4){ // 1, 4, 16, 64,…, n
= (1+1+1+…..+1) - - - $\log_4 n$ terms
= O(log n)
cout << i;
for (j=0;j<n;j=j+2){// (1+1+1+…+1) - - - n/2 terms
(log n times) = O(n log n)
cout << j;
sum++
}
cout << sum;
}

Overall complexity = O(n log n)

28
for (i=1;i<n;i=i*4){ //O(log n)
cout << i;
for (j=0;j<i; j=j+2)
{
// values of i : 1, 4, 16, 64, …. n
= (1)+(1+1+1+1)+(1+1+…1 - - - 16 terms) + …. + (n terms) = 1+ 4 + 16 + 64 + … + n (geometric series)
= O(n)
cout << j;
sum++
}
cout << sum;
}

Overall complexity = O(n)

29

```
for (i=1;i<=n*n;++i){ // (1+1+1+….+1) - - - n^2terms
= O(n^2)
cout << i;
          Sum=0;
          for (j=1; j<=i; ++j)
          {
           // (1) + (1+1) + (1+1+1) + ….. +(1+1+1+ … 1 ---
          n^2terms)  = 1+2+3+4+….+n^2- - - n^2times
           = n^2· n^2
           = O(n^4)

                    Sum++;
                    cout << i;
          }
          cout << Sum;
}
```

Overall complexity = $O(n^4)$

30

```
for (i=1;i<=n*n*n;++i){ //(1+1+1+…+1) - - -
n^3terms
 = O(n^3)
cout << i;
          Sum=0;
          for (j=1; j<=i; ++j)
          {
  // (1) + (1+1) + (1+1+1) + ….. +(1+1+1+ … 1 ---
 n^3terms)  = 1+2+3+4+….+n^3- - - n^3times
  = n^3· n^3
  = O(n^6)

                    Sum++;
                    cout << i;
          }
          cout << Sum;
}
```

Overall complexity = $O(n^6)$

31

```
for (i=1;i<=n*n*n; i*=2){ // 1, 2, 4, 8, 16, ….. , n^3
= (1+1+1+…..+1) - - - log_2 n^3terms  = O(log n)
cout << i;
          Sum=0;
          for (j=1;j<=i; j++)
          { // values of i: 1, 2, 4, 8, …., n^3
 = (1) + (1+1) + (1+1+1+1) + …. + (1+1+1+…+1 - -
-n^3terms)  = 1 + 2+ 4+ 8+ 16 + ….. + n^3(geometric series)
= O(n^3)
                    Sum++;
                    cout << i;
          }
          cout << Sum;
}
```

Overall complexity = $O(n^3)$

32

```
for (i=1;i<=n*n*n; i*=2) { // 1, 2, 4, 8, 16, ….. , n^3
= (1+1+1+…..+1) - - - log_2 n^3terms  = O(log n)

cout << i;
          Sum=0;
          for (j=1;j<=n; j++)
          {
           //(1+1+1+….+1) - - - n terms (log n times)
           = O(n log n)
                    Sum++;
                    cout << i;
          }
          for (k=1;k<=n; k++)
          {
           //(1+1+1+….+1) - - - n terms (log n times)
           = O(n log n)
                    Sum++;
                    cout << i;
          }
          cout << Sum;
}
```

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

33

for (i=1;i<=n*n*n; i*=2){ // 1, 2, 4, 8, 16, ….. , n^3
= (1+1+1+…..+1) - - - $\log_2 n^3$ terms = O(log n)

cout << i;
Sum=0;
for (j=1;j<=i; j++)
{
// values of i: 1, 2, 4, 8, …., n^3
= (1) + (1+1) + (1+1+1+1) + …. + (1+1+1+…+1 - - -n^3terms) = 1 + 2+ 4+ 8+ 16 + ….. + n^3(geometric series)
= $O(n^3)$

Sum++;
cout << i;
}

for (j=1;j<=n; j*=2){// 1, 2, 4, 8, 16, ….. , n
= (1+1+1+…..+1) - - - $\log_2 n$ terms (log n times) = log n · log n
= $O(\log^2 n)$
Sum++;
cout << i;
}

cout << Sum;
}

Overall complexity = $O(n^3)$

34

for (i=1;i<=n*n*n; i*=2){ // 1, 2, 4, 8, 16, ….. , n^3
= (1+1+1+…..+1) - - - $\log_2 n^3$terms = O(log n)
cout << i;
Sum=0;
for (j=1;j<=i; j++)
{
// values of i: 1, 2, 4, 8, …., n^3
= (1) + (1+1) + (1+1+1+1) + …. + (1+1+1+…+1 - - -n^3terms) = 1 + 2+ 4+ 8+ 16 + ….. + n^3(geometric series) = $O(n^3)$

Sum++;
cout << i;
}

for (j=1;j<=n; j++){
// (1+1+1+….+1) - - - n terms (log n times)
= O(n log n)
Sum++;
cout << i;
}

cout << Sum;
}

Overall complexity = $O(n^3)$

35-36

for (int i=1; i <= n ; i = i * 2){ //1, 2, 4, 8, …, n
= (1+1+1+…+1) - - - $\log_2 n$ terms = O(log n)
for (j = 1 ; j <= i ; j = j * 2)
//values of i: 1, 2, 4, 8, …, n
= (1+2+3+…..+$\log_2 n$)
= $(\log n)^2$
= $O(\log^2 n)$
cout<<"*";
}

for (int i=1; i <= n ; i = i * 2)
for (j = 1 ; j <= i ; j = j * 2)
cout<<"*";

for (int i=1; i <= n ; i = i * 2)
for (j = 1 ; j <= i ; j = j * 2)
cout<<"*";

// same as Q35

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

37

for (i=0; i<n; i=i+3){// 0, 3, 6, 9,…., n
= (1+1+1+…..+1) - - - n/3 terms = O(n)
cout << i;
for (j=1; j<n; j=j*3)
{//1, 3, 9, 27,…, n
= (1+1+1+….+1) - - - $\log_3 n$ terms (n times)
= $\log_3 n \cdot n$
= O(n log n)

cout << j;
sum++
}
for (k=1;k<n;k=k*3)
{ //1, 3, 9, 27,…, n
= (1+1+1+….+1) - - - $\log_3 n$ terms (n times)
= $\log_3 n \cdot n$
= O(n log n)
cout << j;
sum++
}
cout << sum;
}

Overall complexity = O(n log n)

38
for (int i=1; i <= n ; i = i * 2){ //values of i: 1, 2, 4, 8, 16,.....n = 1+1+1+1+1 - - - $\log_2 n$ terms
= O(log n)
 	
for (j = 1 ; j <= i ; j = j * 2)
 	
{ // 1, 2, 4, 8, 16, ... , n
= (1+2+3+4+.....+ $\log_2 n$) = (log n)^2
= $O(\log^2 n)$
 	
cout<<"*";
}
}
 	
for(int i=0; i<=N; i++)
 	
{ // (1+1+1+...+1) - - - N terms
= O(N)
Sum++;
}
 	
Overall complexity = O(N)
	39
	for (i=0; i<n; i=i+3){ // 0, 3, 6, 9,...., n
	= (1+1+1+.....+1) - - - n/3 terms = O(n)
	cout << i;
	for (j=1; j<n; j=j*3)
	{ //1, 3, 9, 27,..., n
	= (1+1+1+....+1) - - - $\log_3 n$ terms (n times)
	= $\log_3 n \cdot n$
	= O(n log n)
	
	sum++
	}
	}
	for (k=1;k<n;k=k*3)
	{ //1, 3, 9, 27,..., n
	= (1+1+1+....+1) - - - $\log_3 n$ terms
	= $\log_3 n$
	= O(log n)
	
	cout << j;
	sum++
	}
	cout << sum;
	
	Overall complexity = O(n log n)

Question 3 Analyze the complexity of the following functions in terms of N. 1*5+5 = 10 points

int f1(int N) **{** **int Count = 0;** **for(int i = 1; i<=N ; i*= 2)** // $\log_2 N$ **for(int j=1; j<= i ; j++)** **// 1+2+4+8....+N< 2N approximately N** **Count++;** **return Count;** **}** **Time complexity= O(N)**	**int f2(int N)** **{** **int Count=0;** **int C = f1(N);** **//Complexity of which is already calculated to be O(N)** **for(int i=0; i<C; i++)** **//C<2N** **Count++;** **return Count;** **}** **Time complexity=O(N)**
int f3(int N) **{** **int Count=0;** **int C = sqrt(f1(N));** **// O(N)** **for(int i=1; i<C; i*=2)** **//O(**$\log_2(\sqrt{N})$**)** **Count++;** **return Count;** **}** **Time complexity=O(N)**+O($\log_2(\sqrt{N})$)**=O(N)**	**int f4(int N)** **{** **int Count=0;** **for(int i=0; i<f1(N) * f1(N); i++)** //loop iterates $n*n=n^2$ times. At every iteration, two function calls of complexity O(n) are made. So overall complexity will be $O(n^2)*(2n)=O(n^3)$ **Count++;** **return Count;** **}** **Time complexity** **O(N^3)**

```
int f5(int N)
{
  int Count=0;
    for(int i=0; i<sqrt(f1(N) * f1(N)); i++) //loop iterates n times. At every iteration, two function calls of complexity O(n) are made. So overall complexity would be O(n) * 2(n) = O(n^2)   (Here, we have assumed that sqrt() function takes O(1) time. If you mention the complexity of sqrt() function you have assumed and solve the problem accordingly, you will get marks)

      Count++;
  return Count;
}
```

Time complexity
$O(N^2)$

```
Int Sum=0;
int f6(int N)
{
  if(N==1)
  return 1;
  Sum+=f1(N);  //O(N)
  Sum+=f2(N);  //O(N)
  Sum+=f3(N);  //O(N)
  Sum+=f4(N);  //O(N^3)
  Sum+=f5(N);  //O(N^2)
Total = O(N^3)
  return Sum;
```

Time complexity=O(N^3)