**Assignment – 3**

**Algorithms**

**Name :Tabed Roll#0516**

## Arithmetic Sequence Size

1, 2, 3, 4, 5, 6, .... , N ≤ O(N)

1, 3, 5, 7, .... , N ≤ N/2 i.e O(N)

1, 4, 7, 10, .... , N ≤ N/3 i.e. O(N)

**1, 1+k, 1+2k, 1+3k, 1+4k, 1+5k, .... , N** ≤ **N/k i.e. O(N) if k is a constant**

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Arithmetic Series Applications of

1+2+3+4+5+6+ ....N-3+ N-2+ N-1+ N ≤ O(N2)

1+2+3+4+5+6+ ....(N/2-3)+ (N/2-2)+ (N/2-1)+ N/2 ≤ O(N2)

1+2+3+4+5+6+ ....(N/3-3)+ (N/3-2)+ (N/3-1)+ N/3 ≤ O(N2)  O(N)

1+2+3+4+5+6+ .... + N2 ≤ O(N4)

1+2+3+4+5+6+ .... + N3 ≤ O(N6)

## 1+2+3+4+5+6+ .... +Nk <= O(NkxNk)

1+22+32+42+52+62+ .... + N2 ≤ O(N3) 1+23+33+43+53+63+ .... + N3 ≤ O(N4)

## 1k+2k+3k+4k+5k+6k+ .... +Nk <= O(Nk+1)

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Geometric Sequence Size

N, N/2, N/4, N/8, N/24, N/25 , N/26, …8, 4 , 2 , 1 <= log2 𝑁

N, N/3, N/9, N/27, N/34, N/35 , N/36, … , 33, 9, 3 , 1 <= log3 𝑁

N, N/5, N/25, N/125, N/54, N/55 , N/56, …, 53, 52, 5 , 1 <= log5 𝑁𝐤 O(log N)

**N, N/k, N/k2, N/k3, N/k4, N/k5 , N/k6, … , k3, k2, k, 1 <=** 𝐥𝐨𝐠 𝑵

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

## Application

√𝑁 ∗ √𝑁 = 𝑵 **for(int i=1; i\*i<=N; i++)** Complexity of this loop is **O(**√𝑵)

**Sum++;**

**N x N = N2 for(int i=1; i\*i<=N\*N; i++)** Complexity of this loop is **O(**𝑵)

**Sum++;**

|  |  |
| --- | --- |
| What is the algorithm’s complexity of the following piece of code  int Sum=0;  for(int i=0; i<N; i++) for(int j=0; j<N; j++)  Sum++;  **Big O = (N^2)** | What is the algorithm’s complexity of the following piece of code  int Sum=0;  for(int i=0; i<N; i++) => N  Sum++;  for(int j=0; j<N; j++) => N  Sum++;  **N + N= 2N**  Remove constant  So :  **Big O = (N)** |
| What is the algorithm’s complexity of the following piece of code  int Sum=0;  for(int i=0; i<N; i++)=>N for(int j=0; j<N; j++)=>N for(int k=0; k<N; k++)=>N  Sum++;  **Big O = N^3**    for(int i=0; i<N; i++)=>N  for(int j=0; j<N; j++)=>N for(int k=0; k<N; k++)=>N  Sum++;  **Big O = N^3**  So  **N^3 + N^3 = 2 N^3**  Remove constant =N^3  **Big O = (N^3)** | What is the algorithm’s complexity of the following piece of code  int Sum=0;  for(int i=0; i<N; i++) => N  Sum++;  for(int j=0; j<N; j++) =>N  Sum++;  for(int k=0; k<N; k++) =>N  Sum++;  for(int m=0; m<N; m++)=>N  Sum++;  for(int n=0; n<N; n++)=>N  Sum++;  for(int p=0; p<N; p++) =>  Sum++;  **N+ N+N+N+N+N = 6 N**  Remove constant  **Big O = (N)** |
| int Sum=0;  for(int i=0; i<N; i++)=>**N** for(int j=0; j<i; j++) =>**N** for(int k=0; k<j; k++)=>**N**  Sum++;  **N \* N \* N = N^3**  So:  **Big O =(N^3)** | int Sum=0;  for(int i=0; i<N; i+=2)=> **N/2**  for(int j=0; j<i; j+=2)=>**N/2** for(int k=0; k<j; k+=2)=>**N/2**  Sum++;  **N/2 + N/2+N/2 = 3 N/2**  Remove constant  So  **Big O = (N/2)** |
| int Sum=0;  for(int i=1; i<N; i\*=2)=**log2N**  for(int j=1; j<N; j\*=2) =**log2N**  Sum++;  **Log2N \* log2N = log2N^2**  **2 log2N**  So  **Big O = log2N** | intSum=0;  for(int i=1; i<N; i\*=2)=**log2N**  Sum++;  for(int j=1; j<N; j\*=2)=**log2N**  Sum++;  **Log2N \* log2N = log2N^2**  **2 log2N**  So  **Big O = log2N** |

|  |  |
| --- | --- |
| for(int i=1; i<=N\*N; **i+=2**)=**N^2/2** for(int j=1; j<N\*N; j\*=2) =l**og2N**  Sum++;  So  **Big O = (N^2/2 + log2N)** | for(int i=1; i<=N\*N; **i+=2)** =**N^2/2**  Sum++;  for(int j=1; j<N\*N; j\*=2) =**N^2/2**  Sum++;  **N^2/2 + N^2/2**  **Big O = (N^2/2)** |
| for(int i=1; i<=N\*N**i\*=2) log2N^2**  for(int j=1; j<N\*N j\*=2) **log2N^2**  Sum++;  **Log2N^2 + log2^2 = 2 log2N^2=>log2N**  So  **Big O = (log2N)** | for(int i=1; i<=N\*N; i\*=2)**log2N^2**  Sum++;  for(int j=1; j<N\*N; j\*=2) **log2N^2**  Sum++;  **Log2N^2 = 2log2N**  So  **Big O = (log2N + log2N**) |
| int Sum=0;  for(int i=1; i<=N; i\*=2**)log2N** for(int j=1; j<=N; j\*=2)**log2N** for(int k=1; k<=N; k\*=2)**log2N**  Sum++;  **Log2N \* log2N \* log2N**  So  **Big O = (log2N)** | intSum=0;  for(int i=1; i<=N; i\*=2)**log2N**  Sum++;  for(int j=1; j<=N; j\*=2)**log2N**  Sum++;  for(int k=1; k<=N; k\*=2)**log2N**  Sum++;  **Log2N + log2N+log2N**  So  **Big O = (log2N)** |
| Intsum,i,j; sum = 0;  for (i=1;i<n;i=i\*2)**log2N**  {  for (j=0;j<n;++j)=**N**  {  sum++;  }  }  So  **Big O = (log2N+N)** | **BE CAREFUL GEOMETRIC SERIES**  int sum,i,j;  sum = 0;  for (i=1; i<n; i=i\*2)**log2N**  {  for (j=0; j <i ; ++j) **log2N**  {  sum++;  }  }  So  **Big O = (log2N)** |
| **BE CAREFUL GEOMETRIC SERIES**  int sum,i,j;  sum = 0;  for (i=1; i<n; i=i\*5)l**og5N**  {  for (j=0; j<i; j+=2)**N**  {  sum++;  }  }  So  **Big O =( log5N+N)** | Intsum,i,j; sum = 0;  for (i=1; i<n; i=i\*4)**log4N**  {  for (j=0 ; j<n ; j+=3)**N/3**  {  sum++;  }  }  SO  **Big O =( log4N + N/3)** |

|  |  |
| --- | --- |
| 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;  }  cout<<Sum<<endl;  **Big O of Sum = (N)** | 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;  }  cout<<Sum<<endl;  **Big O of Sum = log2N** |
| What is the time complexity of the algorithm: int Sum = 0;  for(int i=1; i<=n; i+=1)=N  {  for(int j=1; j<=i; j++)=N  {  Sum++;  }  }  cout<<Sum<<endl;  **N \* N = N^2**  So  **Big O if Sum = (N^2)** | What is the time complexity of the algorithm: int Sum = 0;  for(int i=1; i<n; i\*=2)log2N  {  for(int j=1; j<=i; j++)Log2N  {  Sum++;  }  }  cout<<Sum<<endl;  so  **Big O = (log2N \* log2N)** |
| What is the time complexity of the algorithm:  void f1(int n)  {  for(int j=0**; j\*j<=n\*n;** j++**)N** K++; return K;  }  int main()  {  int Sum = 0; int n; cin>>n;  for(int i=1; **i<=f1(n);** i+=1) **N**  {  for(int j=1; j<=i; j++) N  {  Sum++;  }  }  cout<<Sum<<endl;  }  **N \* N= N^2**  So  **Big O = N^2** | What is the time complexity of the algorithm:  void f1(int n)  {  for(int j=1; **j\*j<=n;** j\*=2) **log2N**  K++;  return K;  }  int main()  {  int Sum = 0; int n; cin>>n;  for(int i=1; i<=f1(n); i+=1) log2N  {  for(int j=1; j<=i; j++)log2N  {  Sum++;  }  }  cout<<Sum<<endl;  }  **Log2N \* log2N**  **So Big O = (log2N)** |

|  |  |
| --- | --- |
| 25  What is the time complexity of the algorithm:  void f1(int n)  {  for(int j=1**; j\*j<=n;** j++)  K++;    return **K\*K;**  }  int main()  {  int Sum = 0; int n; cin>>n;  int **Terminator** **= f1(n);**  for(int i=1; i<= **Terminator;** i+=1) **N^2**  {  for(int j=1; j<=i; j++)**N^2**  {  Sum++;  }  }  cout<<Sum<<endl;  }  **N^2 \* N^2 = N^4**  So  **Big O = (N^4)** | What is the time complexity of the algorithm:  void f1(int n)  {  for(int j=0; **j\*j<=n;** j++)  K++;    return K;  }  int main()  {  int Sum = 0; int n; cin>>n;  int **Terminator** **= f1(n);**  for(int i=1; i<=**Terminator**; i+=1)  {  for(int j=1; j<=i; j++)  {  Sum++;  }  }  cout<<Sum<<endl;  }  So  **Big O = (N**) |
| for (i=1;i<n;i=i\*4) **log4N**  {  cout << i;  for (j=0;j<n;j=j+2)**N/2**  {  cout << j;  sum++  }  cout << sum;  }  So  **Big O = (log4N + N/2)** | for (i=1;i<n;i=i\*4) l**og4N**  {  cout << i;  for (j=0;j<i; j=j+2) **N/2**  {  cout << j;  sum++  }  cout << sum;  }  SO  **Big O = (log4N + N/2)** |

|  |  |
| --- | --- |
| for (i=1;i<=n\*n;++i)**N^2**  {  cout << i;  Sum=0;  for (j=1; j<=i; ++j)**N^2**  {  Sum++;  cout << i;  }  cout << Sum;  }  **N^2 \* N^2**  **Big O = (N^4)** | for (i=1;i<=n\*n\*n;++i)**N^3**  {  cout << i;  Sum=0;  for (j=1; j<=i; ++j)**N^3**  {  Sum++;  cout << i;  }  cout << Sum;  }  **N^3\*N^3**  **Big O = (N^6)** |
| for (i=1;i<=n\*n\*n; i\*=2)**log2N^3**  {  cout << i;  Sum=0;  for (j=1;j<=i; j++)**log2N^3**  {  Sum++;  cout << i;  }  cout << Sum;  }  **Log2N ^3 = 3log2N**  SO  **Big O =(log2N)** | for (i=1;i<=n\*n\*n; i\*=2)**log2N^3**  {  cout << i;  Sum=0;  for (j=1;j<=n; j++)**N**  {  Sum++;  cout << i;  }  for (k=1;k<=n; k++)**N**  {  Sum++;  cout << i;  }  cout << Sum;  }  **Log2N^3 \* N \* N**  **3log2N \* N^2**  **Big O = (log2N \*N^2)** |
| for (i=1;i<=n\*n\*n; i\*=2) **log2N^3**  {  cout << i;  Sum=0;  for (j=1;j<=i; j++)**log2N^3**  {  Sum++;  cout << i;  }    **for (j=1;j<=n; j\*=2)log2N**  {  Sum++;  cout << i;  }    cout << Sum;  }  **Log2N^3 = 3log2N**  **Big O = (log2N )** | for (i=1;i<=n\*n\*n; i\*=2)**log2N^3**  {  cout << i;  Sum=0;  for (j=1;j<=i; j++)**log2N^3**  {  Sum++;  cout << i;  }    **for (j=1;j<=n; j++)N**  {  Sum++;  cout << i;  }    cout << Sum;  }  **Log2N^3=3log2N**  **Big O = (log2N \*N)** |
| 35-36   |  | | --- | | for (int i=1; i <= n ; i = i \* 2)**log2N**  {  for ( j = 1 ; j <= i ; j = j \* 2)**log2N**  cout<<”\*”;  }  So  **Big O = (log2N \* log2N)** | | for (int i=1; i <= n ; i = i \* 2)**log2N**  for ( j = 1 ; j <= i ; j = j \* 2)**log2N**  cout<<”\*”;    for (int i=1; i <= n ; i = i \* 2) **log2N**  for ( j = 1 ; j <= i ; j = j \* 2) **log2N**  cout<<”\*”;  So  **Big O = (log2N + log2N)** | | for (i=0; i<n; i=i+3)**N/3**  {  cout << i;  for (j=1; j<n; j=j\*3)**log3N**  {  cout << j;  sum++  }  for (k=1;k<n;k=k\*3)**log3N**  {  cout << j;  sum++  }    cout << sum;  }  So  **Big O = (log3N \* N/3)** |
| for (int i=1; i <= n ; i = i \* 2) **log2N**  {  for ( j = 1 ; j <= i ; j = j \* 2)**log2N**  {  cout<<”\*”;  }  }  for(int i=0; i<=N; i++) N  {  Sum++;  }  So  **Big O = (Log2N + N)** | for (i=0; i<n; i=i+3) **=N/3**  {  cout << i;  for (j=1; j<n; j=j\*3) **log3N**  {  sum++  }  }  for (k=1;k<n;k=k\*3**) log3N**  {  cout << j;  sum++  }  cout << sum;  **Big O = (N/3 \* log3N)** |
|  |  |