**Complexity Analysis – Practice questions**

*Data Structures (Section B & C)*

**Question 1.** Write the tight big-O for the following expressions and **find c and n0**

1. 9n3+400n2
2. n32n + n23n
3. n2/logn + n
4. nk+a+2n
5. nk+a+nk log n
6. 5n3+logn+ n\*(2n+n log n)
7. (100n+logn) \* (25n+log n)
8. n3+ 4n+ 2*n*

**Question 2. For each of the following program fragments give an analysis of the running time in *T(N)* and as well as in *tight* *Big-O*.**

**TO DO:** Dry run the code for different values of N in rough before estimating. Assume cost of cout<< is 1.

|  |
| --- |
| for (int i=1; i <= n ; i = i \* 2)  { for ( j = 1 ; j <= i ; j = j \* 2)  { cout<<”\*”;  }  } |
| for (i=n; i>n; i=i\4){  cout << i;  for (j=0; j<n; j=j+2)  sum++  } |
| int sum, i, j;  sum = 0;  for (i=n;i>=1;i=i-3)  for (j=n;j>0;j--)  sum++; |
| sum = 0;  for( i = 1; i < n; ++i )  for( j = 1; j < i \* i; ++j )  for( k = 0; k <n; ++k )  ++sum; |

**Question 3. Find out what does each of the following algorithm do. Then estimate the best-case and the worst-case running time in term of tight big Oh for each of the following codes**

|  |
| --- |
| a)  int Func(int n)  {  int i;  i = 0;  while (n%3 == 0) {  n = n/3;  i++;  }  return i;  } |
| **b)**  len=1;  for (i = 0; i < n-1; i++) {  i1 = i2 = i;  for (j = i; j < n-1 && a[j] < a[j+1]; j++, i2++);  if ( len < i2 - i1 + 1)  length = i2 - i1 + 1;  } |
| **c)**  int Mystery( int a[], int asize ){  int mSum = 0;  for( int i = 0; i < asize( ); ++i ){  int thisSum = 0;  for( int j = i; j < asize( ); ++j ){  thisSum += a[ j ];  if( thisSum > mSum )  mSum = thisSum;  }  }  return mSum;  } |

**Question 4.** Write an algorithm for following problems and **derive tight Big-O of your algorithm**

|  |
| --- |
| * Reverse an array of size n: O(n) * Find if the given array is a palindrome or not * Sort array using bubble sort * Sort array using selection sort * Sort array using insertion sort * Print a square matrix of size nxn: O(n2) * Sum two matrices of size nxn: O(n2) * Product of two matrices of size nxn: O(n3) * Transpose of a matrix * Printing all numbers that can be represented by n bits: O(2n) * Printing all subsets of numbers in an array of size n: O(2n) * Printing all permutations of numbers in array of size n: O(n!) |