Q1. For each of the following 6 program fragments, give a Big-Oh analysis of the running time (3 points) -

(1)

sum = 0;

f o r ( i = 0; i < n; i++ )

++sum;

Ans: O(n)

(2) sum = 0;

f o r ( i = 0 ; i < n ; i++ )

f o r ( j = 0; j < n; j++)

++sum;

Ans: O(n2)

(3) sum = 0;

f o r ( i = 0; i < n; i++ )

f o r ( j = 0; j < n\*m; j++)

++sum;

Ans: O(n2\*m)

(4) sum = 0;

f o r ( i = 0; i < n; i++ )

f o r ( j = 0; j < i; j++ )

++sum ;

Ans: O(n2 - n)

(5) sum = 0 ;

f o r ( i = 0 ; i < n ; i++ )

f o r ( j = 0 ; j < i\*i ; j++)

for (k = 0; k < j; k++)

++sum;

Ans: O(n\*(n-1)2 ((n-1)2 – 1))

(6) sum = 0 ;

f o r ( i = 0 ; i < n ; i++ )

f o r ( j = 0 ; j < i\*i ; j++)

if (j % i == 0)

for (k = 0; k < j; k++)

++sum;

Ans: O(n\*(n-1)2 ((n-1)2 – 1))

Q2. Programs A and B are analyzed and found to have worst-case running times no greater than 150Nlog2N and N2 , respectively. Answer the following questions (3 points) -

a. Which program has the better guarantee on the running time for large values of N (N > 10,000)?

Ans: Program A

b. Which program has the better guarantee on the running time for small values of N (N < 100)?

Ans: Program B

c. Which program will run faster on average for N = 1000?

Ans: Program B

Q3. Solve the following recurrence relations using the Master theorem (2 points) -

a. T(n) = 3T(n/2) + n/2

Ans: O(n) = O(nlog2(3))

b. T(n) = 4T(n/2) + n2.5

Ans: O(n) = O(n2.5)

Q4. Analyze the run time complexity of the following algorithms (2 points)

a. Given an array (or string), the task is to reverse the array/string.

Algorithm -

1) Initialize start and end indexes as start = 0, end = n-1

2) In a loop, swap arr[start] with arr[end] and change start and end as follows : start = start +1, end = end – 1

3) Repeat 2) while start < end

Ans: O(n/2)

Q5. Given an array A[], the task is to segregate even and odd numbers. All even numbers should appear first, followed by odd numbers.

Algorithm -

1) Initialize two index variables left and right: left = 0, right = size -1

2) Keep incrementing left index until we see an odd number.

3) Keep decrementing right index until we see an even number.

4) Swap arr[left] and arr[right]

5) Repeat 2 - 4 while left < right

Ans: O((n/2)(2n – 2))