Question 1 [10 Marks]

Write an algorithm (in **pseudocode** form) to find the difference between the largest and the smallest elements of an array of **n** random integers and also write the asymptotic time complexity of the algorithm. Your solution should not take more than linear steps (time complexity) and constant extra space.

```
    max = array[1]
    min = array [1]
    FOR i = 2 to Length(array)
        IF max < array [i]
            max = array [i]
        ELSE IF min > array[i]
            min = array[i]
        END IF
    END FOR
Diff = max - min
Time Complexity: Θ(n)
```

Question 2 [5x5=25 Marks]

Provide a worst-case asymptotic time complexity of the following algorithms by using a suitable asymptotic notation considering a nearest function. Assume that there are no errors/ bugs in the algorithms. Show the meaningful working behind your answer in the rough work column.

		Time Complexity	**Rough Work**

a)	Func(n)
	BEGIN
	  FOR (i = 1 to n)
	    FOR (j = 1 to i×i)
	      IF (j modulus i = 1) THEN
	        PRINT i
	    END FOR
	  END FOR
	END
		$\Theta(n^3)$	$\frac{n(n+1)(2n+1)}{6}$

b)	for (int i = 1; i <= n*n; i = i * 3) {
	  for (int j = 0; j < i; j++)
	    cout << j;
	}
		$\Theta(n^2)$	$1+3+9+\ldots+n^2$ (Geometric series)

c)	for (int i = 0; i <= n; i = i+2) {
	  for (int j = n; j >= i; j--) {
	    for (k = 100; k >= 1; k = k/2)
	      cout << k;
	  }
	}
		$\Theta(n^2)$	i-loop: n/2 times
			j-loop: i-times, maximum n-times
			k-loop: constant, log 100

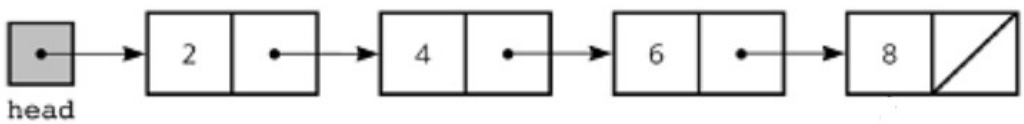
d)	for (int i = 1; i*i <= n; i++)
	{
	  if (n % i == 0)
	    cout << i;
	}
		$\Theta(\sqrt{n})$	Assume n=64
			i=1; 1x1<=64
			  2; 2x2<=64
			…
			  8; 8x8<=64
			Loop: 8 times
			log 64= 6
			$\sqrt{64} = 8$

e)	*// Algorithm to Search in a Binary*
	// Search Tree
	while x ≠ NIL and k ≠ x.*key*
	    if k < x.*key*
	        x = x.*left*
	    else x = x.*right*
	return x
		Θ(n)	In worst case, the tree can be skewed, i.e., totally unbalanced.

Question 3 [12.5+12.5=25 Marks]

Write recursive algorithms for the following problems. Write the recurrence relations and the asymptotic time complexities of your algorithms.

a) Suppose there are ‘n’ integers stored in a Singly Linked List. Write a recursive algorithm (in **pseudocode** form) to sum all elements of the list and write the *worst case* asymptotic time complexity.




```
Sum(nodeptr)
BEGIN
   IF (nodeptr = NULL) THEN
       RETURN 0
   ELSE
       RETURN nodeptr ->data + Sum(nodeptr ->next)
   END IF
END

T(n) = T(n-1) + c
T(0) = c
T(n) = Θ(n)
```

b) Given an array, size of the array, and an element to be searched, write a recursive function (in **pseudocode** form) to find the last occurrence of the element in the array and write the asymptotic *best-case* time complexity.

```
Find(array, size, target)
BEGIN
    IF (size < 1) THEN
        RETURN -1
    END IF
    IF (array[size] = target) THEN
        RETURN size
    END IF
    RETURN Find(array, size-1, target)
END

T(n) = T(n-1) + c
```

In the best case, $T(n) = T(1) = \Omega(1)$

Question 4 [5+5+10=20 Marks]

a) Can the Master theorem (for solving recurrence) be applied to the following recurrence relation:
$T(n)=7T(n/2) + n^3 + n\log n$
Why or why not? If yes, then give an asymptotic upper bound for this recurrence using Master Theorem.

Solution:
Yes.
$n^{\log_b a}$
$n^{\log_2 7}= n^{2.8}$
$f(n)=n^3$
Case 3 (of Master Theorem) as f(n) grows faster than $n^{\log_b a}$
Check regularity condition: $af(n/b) <= cf(n)$
$7(n/2)^3 <= n^3$ satisfies with c=1
Therefore, $T(n) = \Theta(n^3)$

b) Solve the following recurrence relation using iteration method. Show all steps.
$T(n) = T(n-1) + n^2$; for all $n > 0$
$T(0) = 1$

Solution:
$T(n) = T(n-1)+ n^2$
$= T(n-2)+ (n-1)^2 +n^2$
$= T(n-3) + (n-2)^2 + (n-1)^2 +n^2$
.
.
.
$= T(n-k)+ (n-k+1)^2 + (n-k+2)^2 +.....+ (n-2)^2 + (n-1)^2 +n^2$
$= T(n-n)+ (n-n+1)^2 + (n-n+2)^2 +.....+ (n-2)^2 + (n-1)^2 +n^2$ (Putting k = n)
$= 0+1+2^2+3^2...+n^2$ (Formula # 3 in the formula guide)
$= (n(n+1)(2n+1))/6$
$= O(n^3)$

c) Solve the following recurrence using recursion tree method and write the asymptotic upper bound (Big-O):
$T(n) = 2T(n/4) + n\log n$

Solution:

```
      (nlogn)                              nlogn
      /   \
(n/4)log(n/4)   (n/4)(log(n/4)     ----- (n/2)logn -n
/\
(n/16)log(n/16)                    ----- (n/4)logn -n
…
…
```

Time complexity:

$T(n) = n\log n + (n/2)\log n - n + (n/4)\log n - n + \ldots + (n/2^k)\log n - n$

$= \log n(n+n/2+n/4+\ldots.+1) - (n+n+n+\ldots+n)$ (Putting $2^k = n$)

$= \log n(n+n/2+n/4+\ldots.+1) - n\log n$

$\le n\log n$

$= O(n\log n)$