Date-

Assignment No. :

Problem Statement:

Program in C to find the least common multiple and highest common factor of a set of integers.

Theory:

The highest common factor of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers. The lowest common multiple of two integers *a* and *b*, is the smallest positive integer that is divisible by both *a* and *b*. Since division of integers by zero is undefined, this definition has meaning only if *a* and *b* are both different from zero.

Example : The HCF of 12 and 16 is 4. The LCM of 12 and 16 is 48.

Algorithm:

Input Specification: A array of numbers to find the LCM and HCF of, say A.

Output Specification: The LCM and HCF of the given numbers.

Steps:

Algorithm for method HCF(m, n): //The numbers to find HCF of, say m
// and n

Step 1: If(m == 0) Then
Step 2: Return n

Step 3: Else If(n == 0) Then
Step 4: Return m
Step 5: Else
Step 6: Return Hcf(n, m%n)
[End of if structure]

Algorithm for method LCM(m, n, x): //m is the First(larger) number, n is //the Second(smaller) number and x is //Increment factor

Step 1: If(m % n != 0) Then
Step 2: Return Lcm(m + x, n, x)
Step 3: Else
Step 4: Return m
[End of if structure]

Algorithm for method Main():

Step 1: Print "Enter number of elements : "
Step 2: Input num
Step 3: If(num < 1) Then
Step 4: Print "[Error] Number of elements must be positive!"
[End of if structure]
Step 5: Print "Enter 1st number : "
Step 6: Input n
Step 7: Set A[1] = n, lc = A[1], hc = A[1]
Step 8: Set i = 2
Step 9: Repeat through step 10 to 28 While (i <= num)
Step 10: Set suf = i%10
Step 11: Print "Enter ",i
Step 12: If(suf == 1) Then
Step 13: Print "st"
Step 14: Else If(suf == 2) Then
Step 15: Print "nd"

Step 16: Else If(suf == 3) Then
Step 17: Print "rd"
Step 18: Else
Step 19: Print "th"
[End of if structure]
Step 20: Print " number : "
Step 21: Input n
Step 22: Set A[i] = n
Step 23: If(lc < A[i]) Then
Step 24: Set lc = LCM(A[i], lc, A[i])
Step 25: Else
Step 26: Set lc = LCM(lc, A[i], lc)
[End of if structure]
Step 27: Set hc = HCF(hc, A[i])
Step 28: Set i = i + 1
[End of while loop]
Step 29: Print "\nHighest Common Factor : ", hc
Step 30: Print "\nLowest Common Mulitple : ", lc, "\n"

Source Code:

```
#include <stdio.h>
// Procedure to find HCF between m and n
int HCF(int m, int n) {
   if (!m) // m is zero, so n is hcf
      return n;
   if (!n) // n is zero, so m is hcf
      return m;
        // When both are non-zero
   return (HCF(n, m % n)); // Recursive call
}

int LCM(int m, int n, int x) {
   if (m % n)                  // Remainder is greater than zero, so we will
                           // go to the next factor of m, i.e. (m+x)
```

```
    return (LCM((m + x), n, x)); // Recursive call
  else
    return m; // m is completely divisible by n
}

int main() {
  int n, i, lc, hc;
  printf("\nEnter number of elements : ");
  scanf("%d", &n);
  if (n < 1) {
    printf("\n[Error] Number of elements must be positive!");
    return 1;
  }
  int A[n];
  printf("Enter 1st number : ");
  scanf("%d", &A[0]);
  lc = hc = A[0];
  for (i = 1; i < n; i++) {
    int j = i + 1, suf = j % 10; // for display purposes
    printf("Enter %d%s number : ", j,
        suf == 1 ? "st" : // *1st
        suf == 2 ? "nd" : // *2nd
        suf == 3 ? "rd" : "th"); // *3rd / *th
    scanf("%d", &A[i]); // input
    if (lc < A[i])
      lc = LCM(A[i], lc, A[i]); // LCM Function calling
    else
      lc = LCM(lc, A[i], lc);
    hc = HCF(hc, A[i]); // HCF function calling
  }
  printf("\nHighest Common Factor : %d", hc);
  printf("\nLowest Common Multiple : %d", lc);
}
```

Input & Output:

Set 1:

```
Enter number of elements : 3
Enter 1st number : 12
Enter 2nd number : 88
Enter 3rd number : 96

Highest Common Factor : 4
Lowest Common Multiple : 1056
```

Set 2:

```
Enter number of elements : 5
Enter 1st number : 68
Enter 2nd number : 99
Enter 3rd number : 14
Enter 4th number : 108
Enter 5th number : 202

Highest Common Factor : 1
Lowest Common Multiple : 14278572
```

Discussion:

1. For large datasets, this program is infeasible.
2. The recursive call can result to a stack overflow depending on the size of the call stack, and hence it is machine dependent.
3. For large numbers, complexity of this algorithm is high.