Rajshahi University of Engineering & Technology
Department of Computer Science of Engineering

EXPERIMENT NO: 01

NAME OF EXPERIMENT: Complexity of Algorithms

SUBMITTED TO:

RIZOAN TOUFIQ
ASSISTANT PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

SUBMITTED BY:

NAME: MD. ARIFUL ISLAM
ROLL NO.: 1803046
GROUP: 2ND THIRTY
DATE OF EXP.: 01-09-2019
DATE OF SUB. : 08-09-2019
SERIES: 18

MACHINE CONFIGURATION:

ASUS X510UF
CORE i5 8TH GENERATION
8GB RAM
WIN 10 OS

THEORY: Algorithm complexity is a measure which evaluates the order of the count of operations, performed by a given or algorithm as a function of the size of the input data. Complexity is a rough approximation of the number of steps necessary to execute an algorithm. It can be Time Complexity or can be Space Complexity.

Big-O Notation: f(n) is O(g(n)) if and only if there exist two constants c and n0such that |f(n)| < c|g(n)| for all n>no. f(n) will normally represent the computing time of some algorithm. When we say that the computing time of an algorithm is O(g(n)) we mean that its execution takes no more than a constant times *g*(*n*). *n* is a parameter which characterizes the inputs and/or outputs. For example *n* might be the number of inputs or the number of outputs or their sum or the magnitude of one of them.

PROBLEM 1 : Finding the Complexity of a Loop.

ALGORITHM:

```
Step 1. Repeat for K = 1 to n by 1
Step 2. Write: K
        [End of Step 1 loop]
Step 3. Exit
```

CODE:

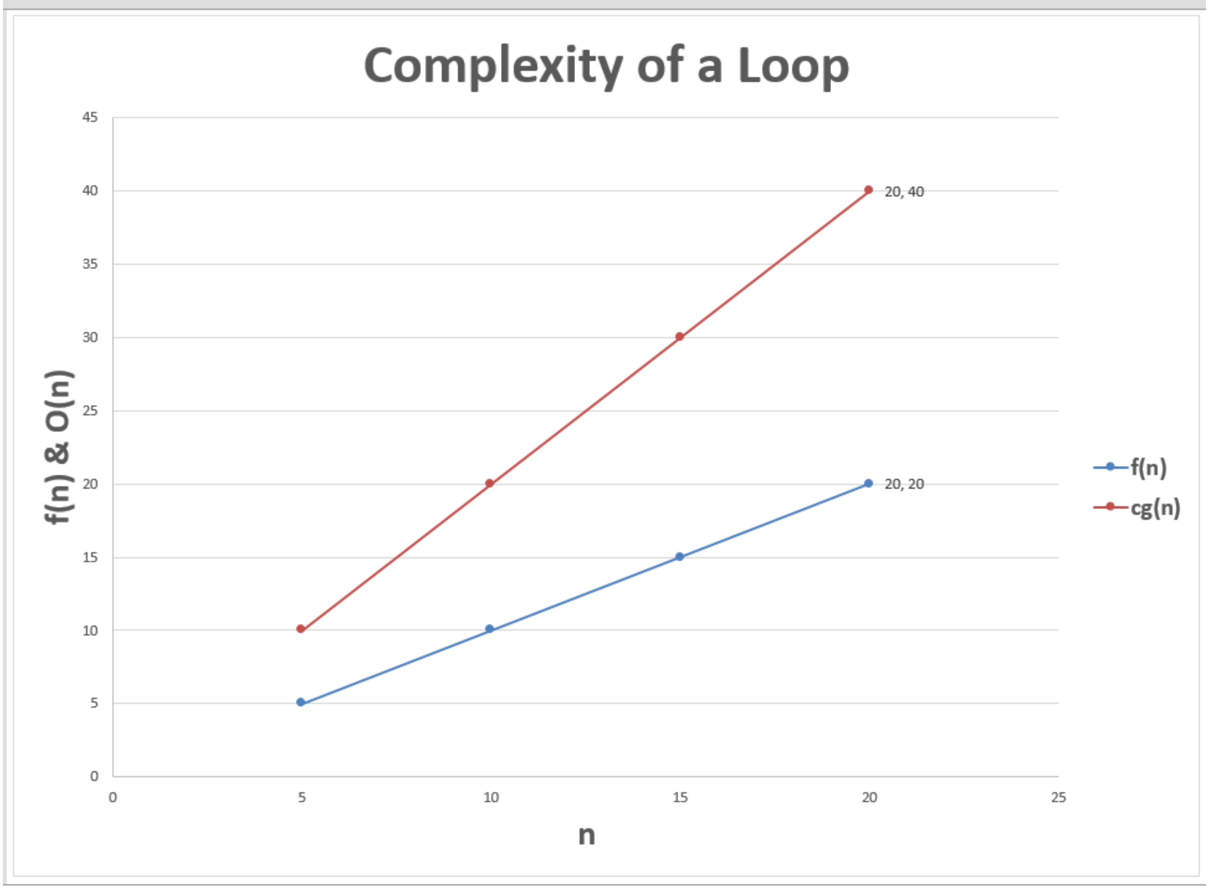
```
#include<stdio.h>

int main()
{
   int k,n,count=0;

   scanf("%d",&n);
   for(k=1;k<=n;k++)
   {
      printf("K ");
      count++;
   }
   printf("\n%d",count);

   return 0;
}
```

COMPLEXITY ANALYSIS: The computing time of the loop, f(n) = n. So the complexity of the above algorithm is O(n). Here f(n)≤cg(n), c =2, n0 = 0, g(n) = n.

COMPLEXITY TABLE:

N	f(n) (From Program)	cg(n)
5	5	10
10	10	20
15	15	30
20	20	40

GRAPH:



PROBLEM 2: Finding the Complexity of the following Program.

ALGORITHM:

```
Step 1. Repeat for K = 1 to n by 1
Step 2. Repeat for L = 1 to n by 1
Step 3. Write: L
            [End of Step 2 loop]
Step 4. Write: K
            [End of Step 1 loop]
Step 5. Exit
```

CODE:

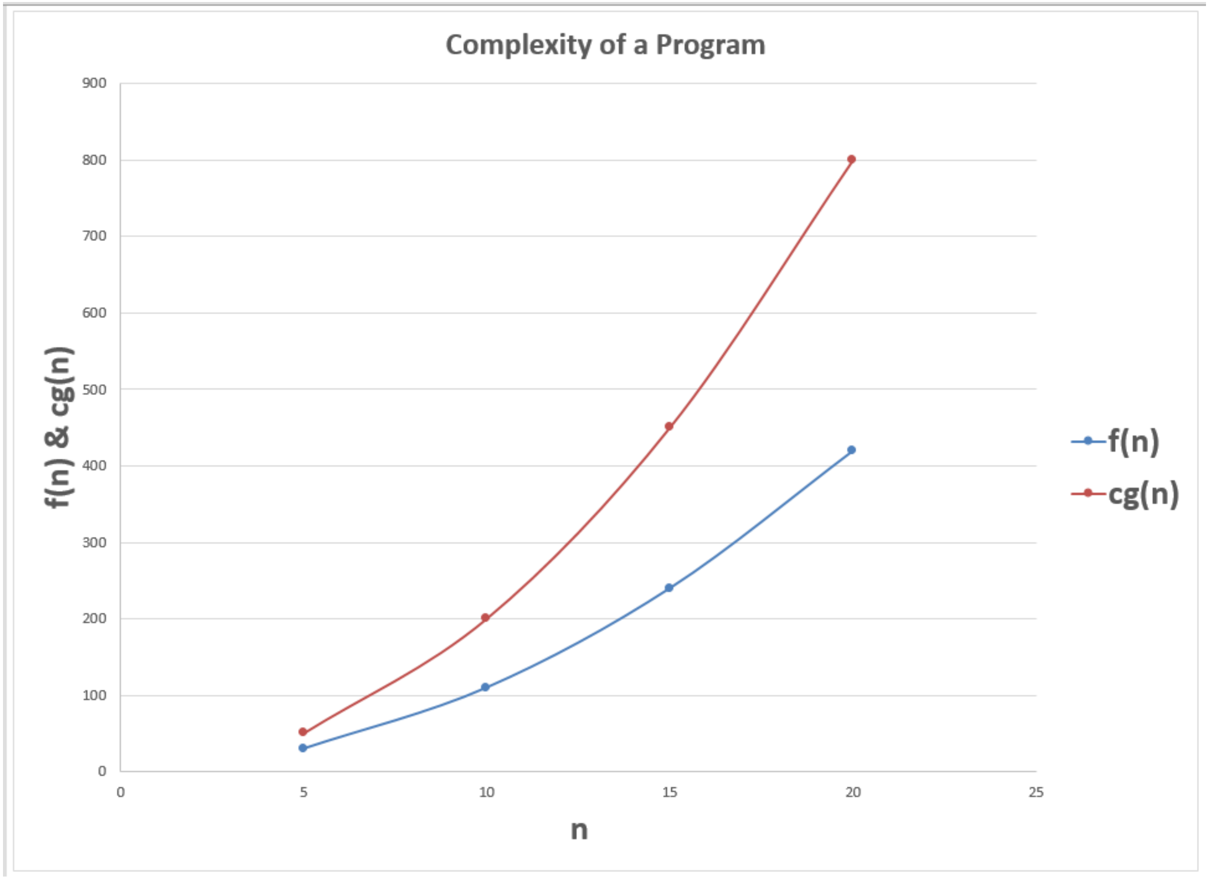
```
#include<stdio.h>
#include<math.h>

int main()
{
    int k,l,n,c=0;

    scanf("%d",&n);
    for(k=1;k<=n;k++)
    {
        for(l=1;l<=n;l++)
        {
            printf("L ");
            c++;
        }
        printf("K ");
        c++;
    }
    printf("\n%d",c);

    return 0;
}
```

COMPLEXITY ANALYSIS: The computing time of the loop, $f(n) = (n^2 + n)$, So the complexity of the above algorithm is $O(n^2)$. (Let $c= 2$)

COMPLEXITY TABLE:

N	**f(n) (From Program)**	**cg(n)**
5	30	50
10	110	200
15	240	450
20	420	800

GRAPH:



PROBLEM 3: Find the Complexity of the elementary Sort algorithm.

ALGORITHM: (Given a nonempty array A with n numerical values**.** This algorithm sorts the values)

```
Step 1. Repeat for i = 2 to n by 1
Step 2. Repeat for k = i to 1 by -1
Step 3. If A[k]<A[k-1] then:
                Swap (A[k], A[k-1])
                [End of If Structure]
                [End of Step 2 loop]
                [End of Step 1 loop]
Step 4. Exit
```

CODE:

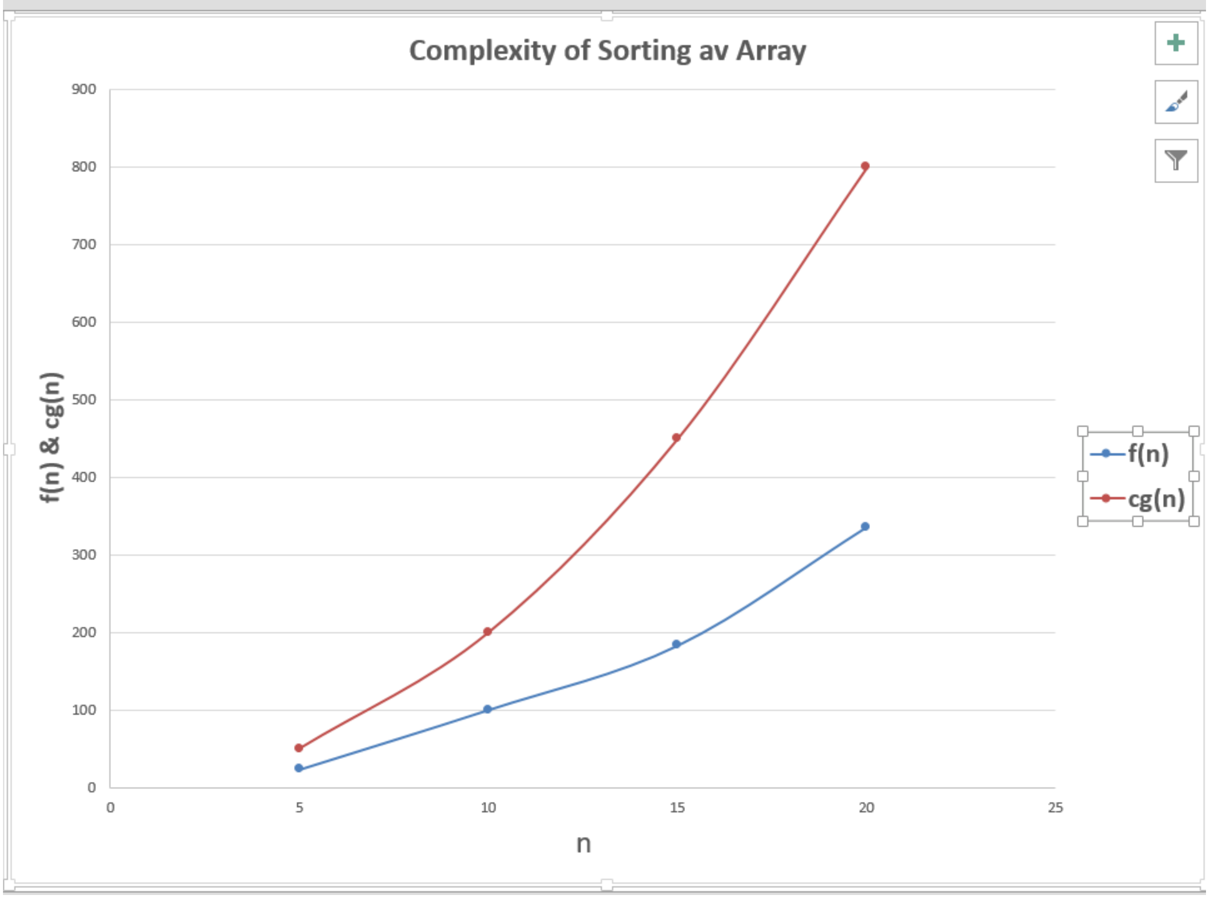
```
#include<stdio.h>

int main()
{
     int a,i,k,c=0,n,A[20]={14,56,31,57,87,11,7,23,42,2,45,78,93,91,79,81,9,63,37,1};

    scanf("%d",&n);
    for(i=1;i<n;i++)
    {
        for(k=i;k>=0;k--)
         {
           if(A[k]<A[k-1])
           {
               a=A[k];
               A[k]=A[k-1];
               A[k-1]=a;
               c++;
           }
           c++;
         }
        c++;
    }
    printf("%d",c);


    return 0;
}
```

COMPLEXITY ANALYSIS: The complexity of the above algorithm is O(n2). (Let c= 2).

COMPLEXITY TABLE:

N	f(n) (From Program)	cg(n)
5	23	50
10	100	200
15	183	450
20	336	800

GRAPH:



PROBLEM 4: Finding the largest element in Array.

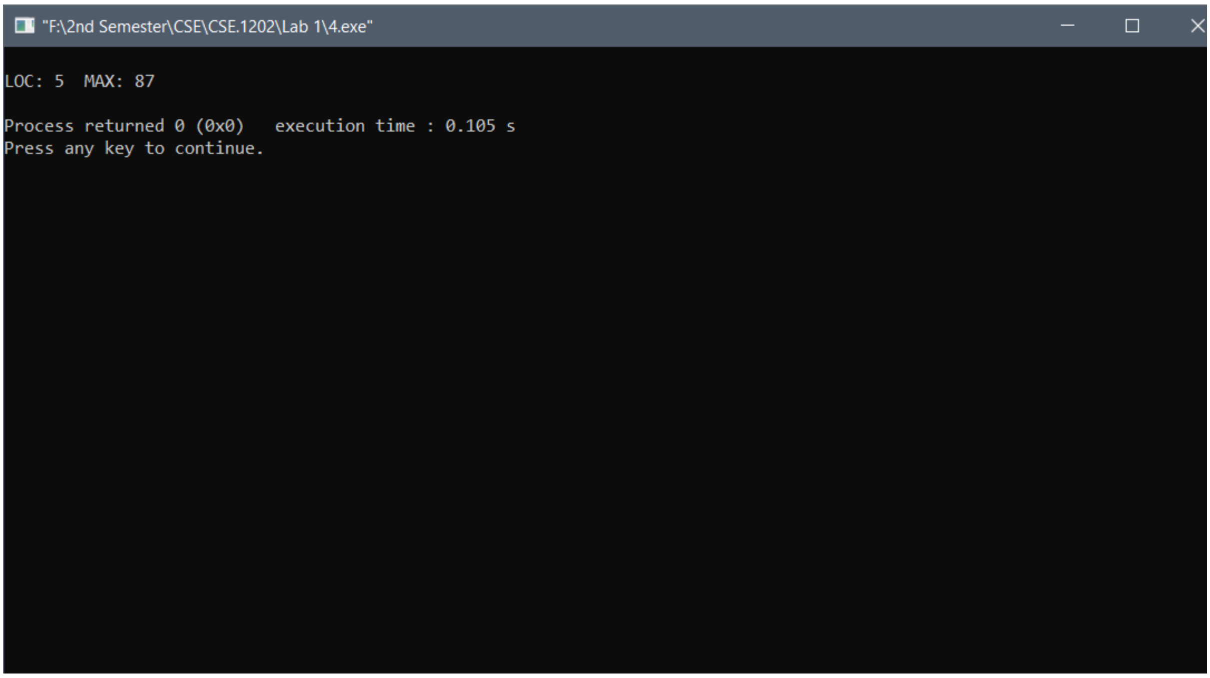
ALGORITHM: (Given a nonempty array A with n numerical values. This algorithm finds the location LOC and the value MAX of the largest element of A)

```
1. Set K:=1, LOC:=1 and MAX:=A[1]
2. Repeat steps 3 and 4 while K≤n
3. IF MAX < A[K] then:
        Set LOC:=K and MAX:=A[K].
        [End of If structure]
4. K:=K+1.
        [End of step 2 loop]
5. Write: LOC, MAX.
6. Exit
```

CODE:

```c
#include<stdio.h>

int main()
{
   int k=1,l=1,A[10]={14,56,31,57,87,11,7,23,42,2};
   int m=A[0];

   while(k<10)
   {
      if(m<A[k])
      {
         m=A[k];
         l=k+1;
      }
      k++;
   }
   printf("\nLOC: %d  MAX: %d\n",l,m);



   return 0;
}
```

OUTPUT:



```
LOC: 5  MAX: 87

Process returned 0 (0x0)   execution time : 0.105 s
Press any key to continue.
```

PROBLEM 4: Linear Search.

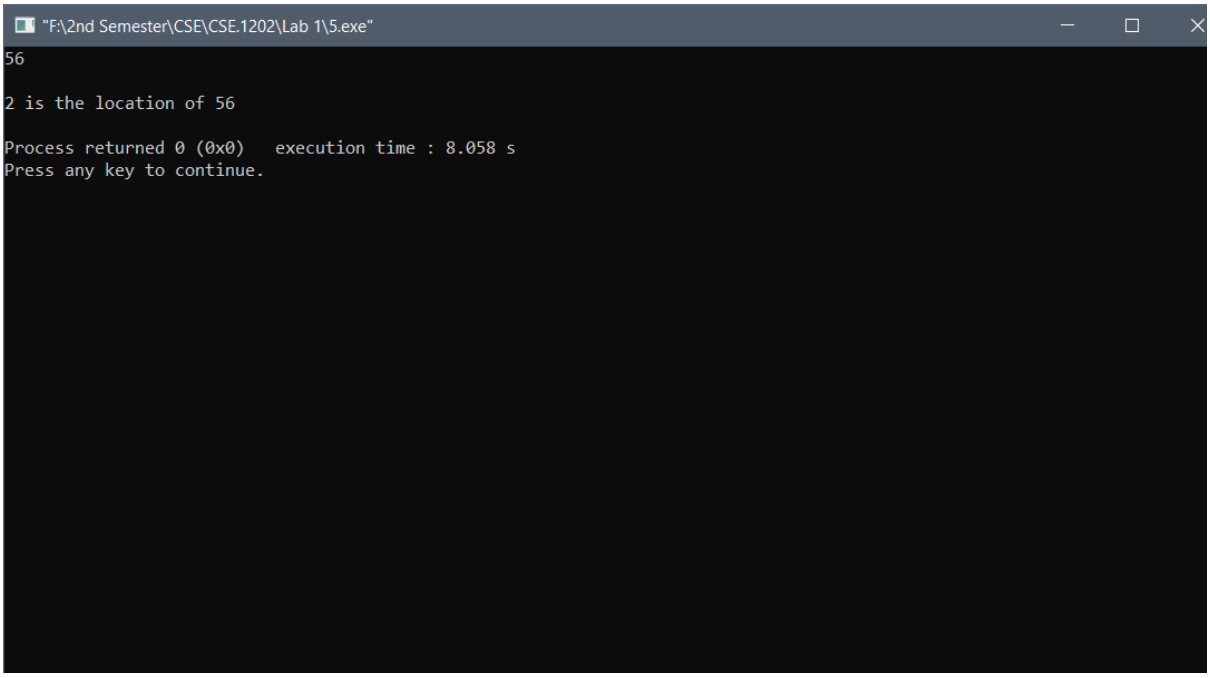
ALGORITHM: (Given a nonempty array A with n numerical values and a specific x of information is given**.** This algorithm finds the location LOC of x in the array A or Sets LOC=-1)

```
1. Set K:=1, LOC:=-1
2. Repeat steps 3 and 4 while LOC = -1 and K≤n
3. IF x = A[K] then:
        Set LOC:=K. [End of If structure]
4. K:=K+1.
        [End of step 2 loop]
5. If LOC = -1 then: Write: x is not in the array A.
Else: Write: LOC is the location of x [End of If structure]
6. Exit
```

CODE:

```
#include<stdio.h>

int main()
{
    int k=0,l=-1,x,A[10]={14,56,31,57,87,11,7,23,42,2};

    scanf("%d",&x);
    while(l==-1&&k<10)
    {
        if(x==A[k])
            l=k+1;
        k++;
    }
    if(l==-1)
        printf("\n%d is not in the array A.\n",x);
    else
        printf("\n%d is the location of %d\n",l,x);



    return 0;
}
```

OUTPUT:



```
56

2 is the location of 56

Process returned 0 (0x0)   execution time : 8.058 s
Press any key to continue.
```