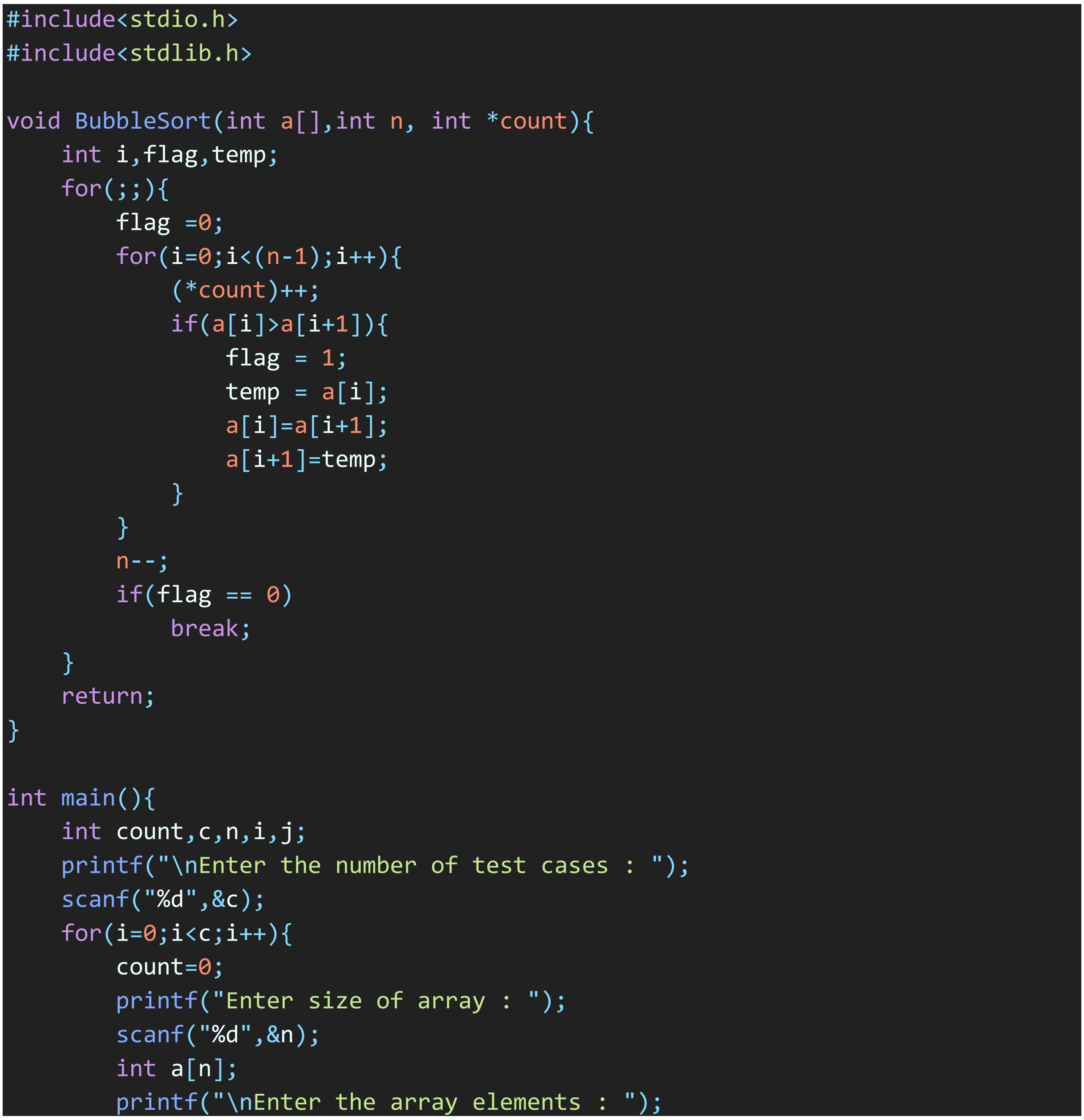
Ayush Goyal

190905522 CSE D 62

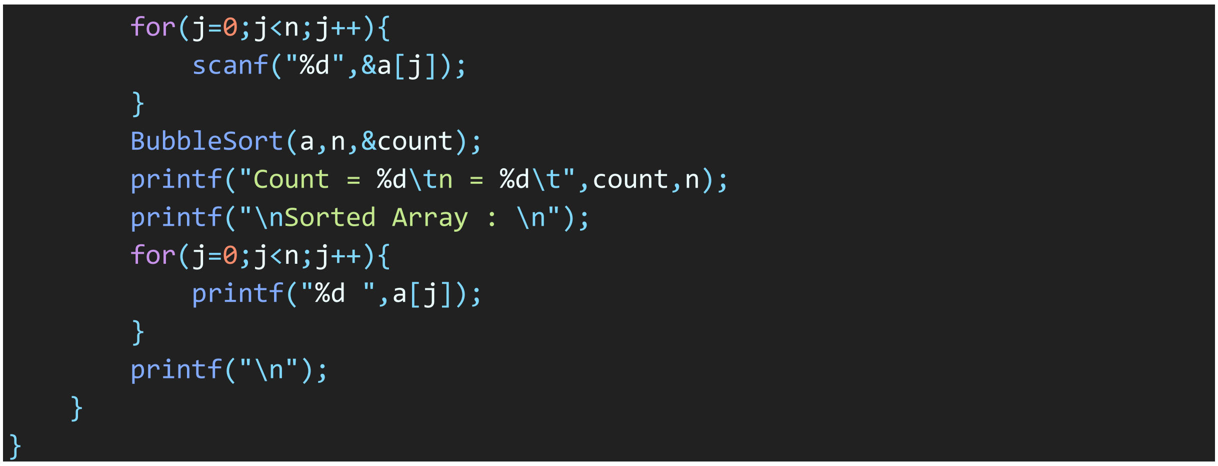
DAA Lab (Week 3) Brute Force Technique

Q1) Write a program to sort set of integers using bubble sort. Analyse its time efficiency. Obtain the experimental result of order of growth. Plot the result for the best and worst case.

CODE:

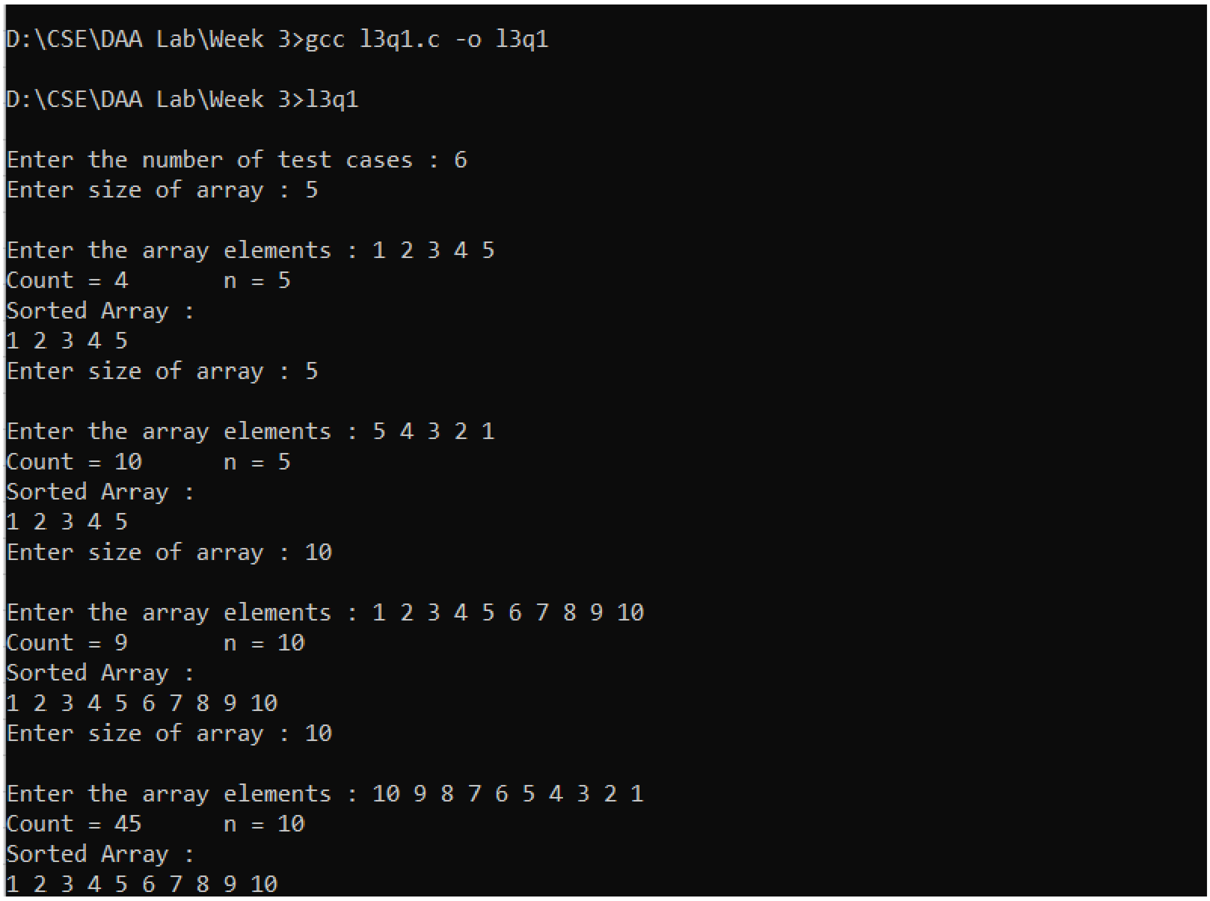


```
#include<stdio.h>
#include<stdlib.h>

void BubbleSort(int a[],int n, int *count){
    int i,flag,temp;
    for(;;){
        flag =0;
        for(i=0;i<(n-1);i++){
            (*count)++;
            if(a[i]>a[i+1]){
                flag = 1;
                temp = a[i];
                a[i]=a[i+1];
                a[i+1]=temp;
            }
        }
        n--;
        if(flag == 0)
            break;
    }
    return;
}

int main(){
    int count,c,n,i,j;
    printf("\nEnter the number of test cases : ");
    scanf("%d",&c);
    for(i=0;i<c;i++){
        count=0;
        printf("Enter size of array : ");
        scanf("%d",&n);
        int a[n];
        printf("\nEnter the array elements : ");
```



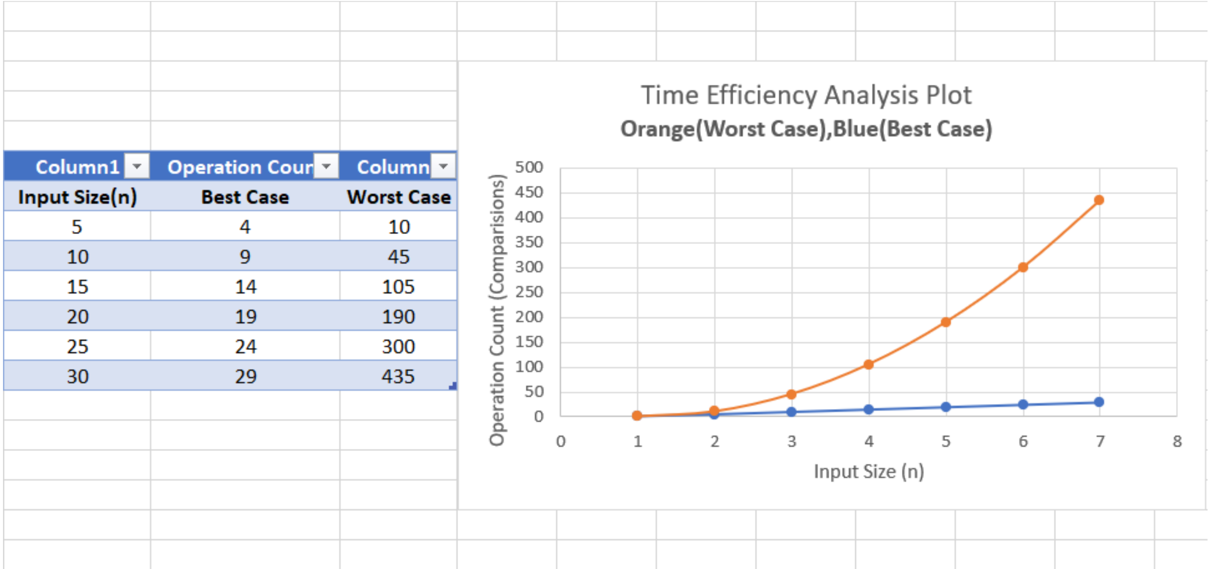
```c
        for(j=0;j<n;j++){
            scanf("%d",&a[j]);
        }
        BubbleSort(a,n,&count);
        printf("Count = %d\tn = %d\t",count,n);
        printf("\nSorted Array : \n");
        for(j=0;j<n;j++){
            printf("%d ",a[j]);
        }
        printf("\n");
    }
}
```

INPUT/OUTPUT:



```
D:\CSE\DAA Lab\Week 3>gcc l3q1.c -o l3q1

D:\CSE\DAA Lab\Week 3>l3q1

Enter the number of test cases : 6
Enter size of array : 5

Enter the array elements : 1 2 3 4 5
Count = 4       n = 5
Sorted Array :
1 2 3 4 5
Enter size of array : 5

Enter the array elements : 5 4 3 2 1
Count = 10      n = 5
Sorted Array :
1 2 3 4 5
Enter size of array : 10

Enter the array elements : 1 2 3 4 5 6 7 8 9 10
Count = 9       n = 10
Sorted Array :
1 2 3 4 5 6 7 8 9 10
Enter size of array : 10

Enter the array elements : 10 9 8 7 6 5 4 3 2 1
Count = 45      n = 10
Sorted Array :
1 2 3 4 5 6 7 8 9 10
```

Graph Plot for Time Efficiency for best and worst cases:



Column1	Operation Cour	Column
Input Size(n)	Best Case	Worst Case
5	4	10
10	9	45
15	14	105
20	19	190
25	24	300
30	29	435

Time Efficiency Analysis Plot
Orange(Worst Case),Blue(Best Case)
Operation Count (Comparisions)
500
450
400
350
300
250
200
150
100
50
0
0
1
2
3
4
5
6
7
8
Input Size (n)

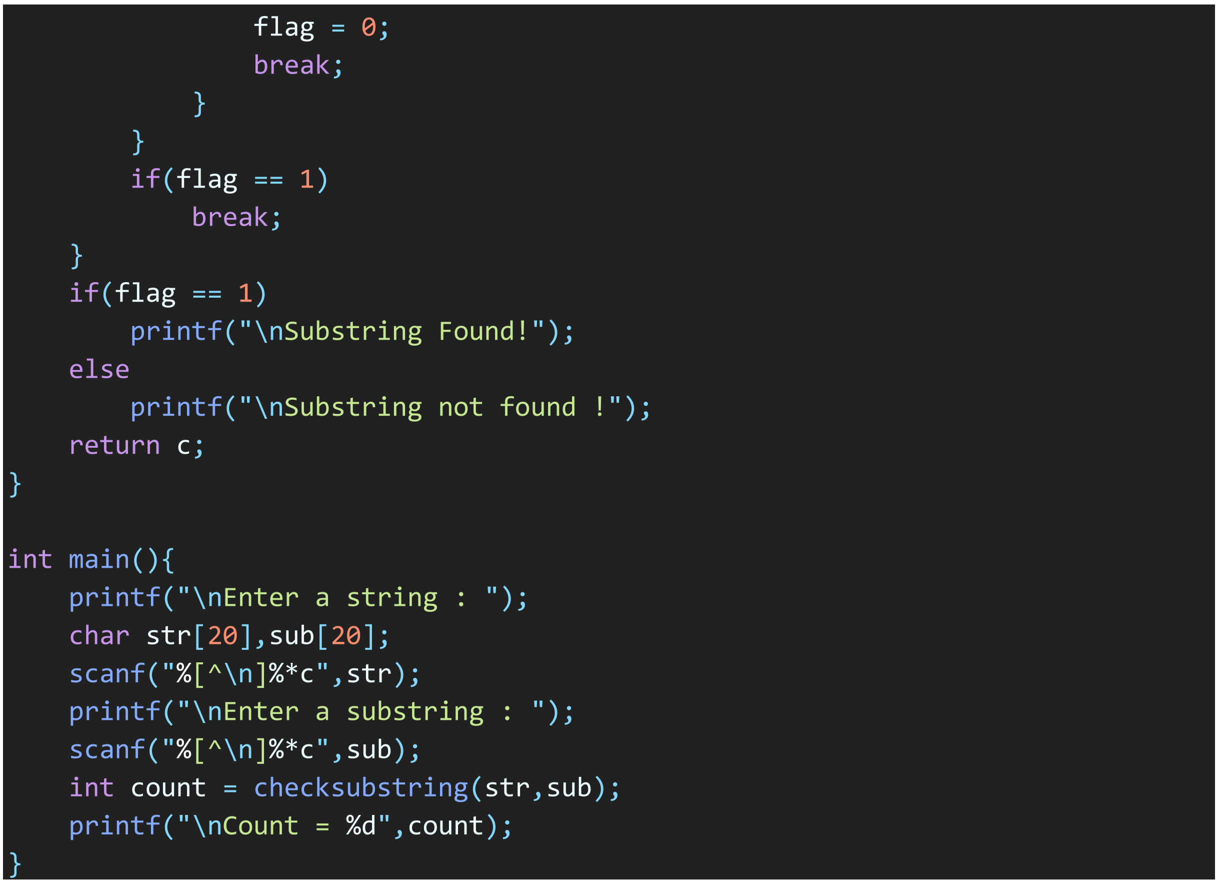
Time Efficiency Analysis:

The best case for this bubble sort algorithm is when the input array is already sorted and then the time complexity is **O(n)**. The worst case for this algorithm is when the input array is in decreasing order and then the time complexity is **O(n^2)**.

Q2) Write a program to implement brute-force string matching. Analyse its time efficiency.

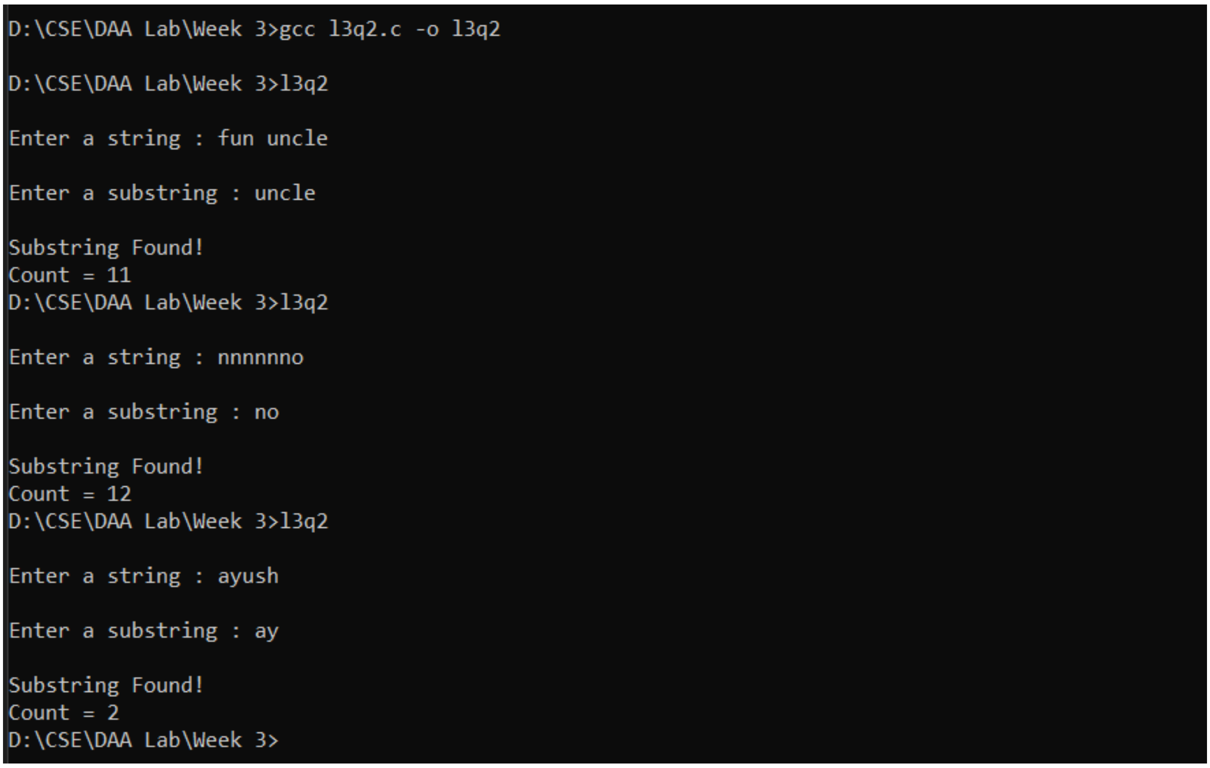
CODE :

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int checksubstring(char str[], char sub[]){
    int c=0,c1,flag=1;
    int i,j,n,m;
    n = strlen(str);
    m = strlen(sub);
    for(i=0;i<=(n-m);i++){
        c1=0;
        flag = 1;
        for(j=i; c1<m; j++,c1++){
            c++;
            if(str[j] != sub[c1]){
```



```
                flag = 0;
                break;
            }
        }
        if(flag == 1)
            break;
    }
    if(flag == 1)
        printf("\nSubstring Found!");
    else
        printf("\nSubstring not found !");
    return c;
}

int main(){
    printf("\nEnter a string : ");
    char str[20],sub[20];
    scanf("%[^\n]%*c",str);
    printf("\nEnter a substring : ");
    scanf("%[^\n]%*c",sub);
    int count = checksubstring(str,sub);
    printf("\nCount = %d",count);
}
```

INPUT/OUTPUT:



```
D:\CSE\DAA Lab\Week 3>gcc l3q2.c -o l3q2

D:\CSE\DAA Lab\Week 3>l3q2

Enter a string : fun uncle

Enter a substring : uncle

Substring Found!
Count = 11
D:\CSE\DAA Lab\Week 3>l3q2

Enter a string : nnnnnno

Enter a substring : no

Substring Found!
Count = 12
D:\CSE\DAA Lab\Week 3>l3q2

Enter a string : ayush

Enter a substring : ay

Substring Found!
Count = 2
D:\CSE\DAA Lab\Week 3>
```

Time Efficiency Analysis:

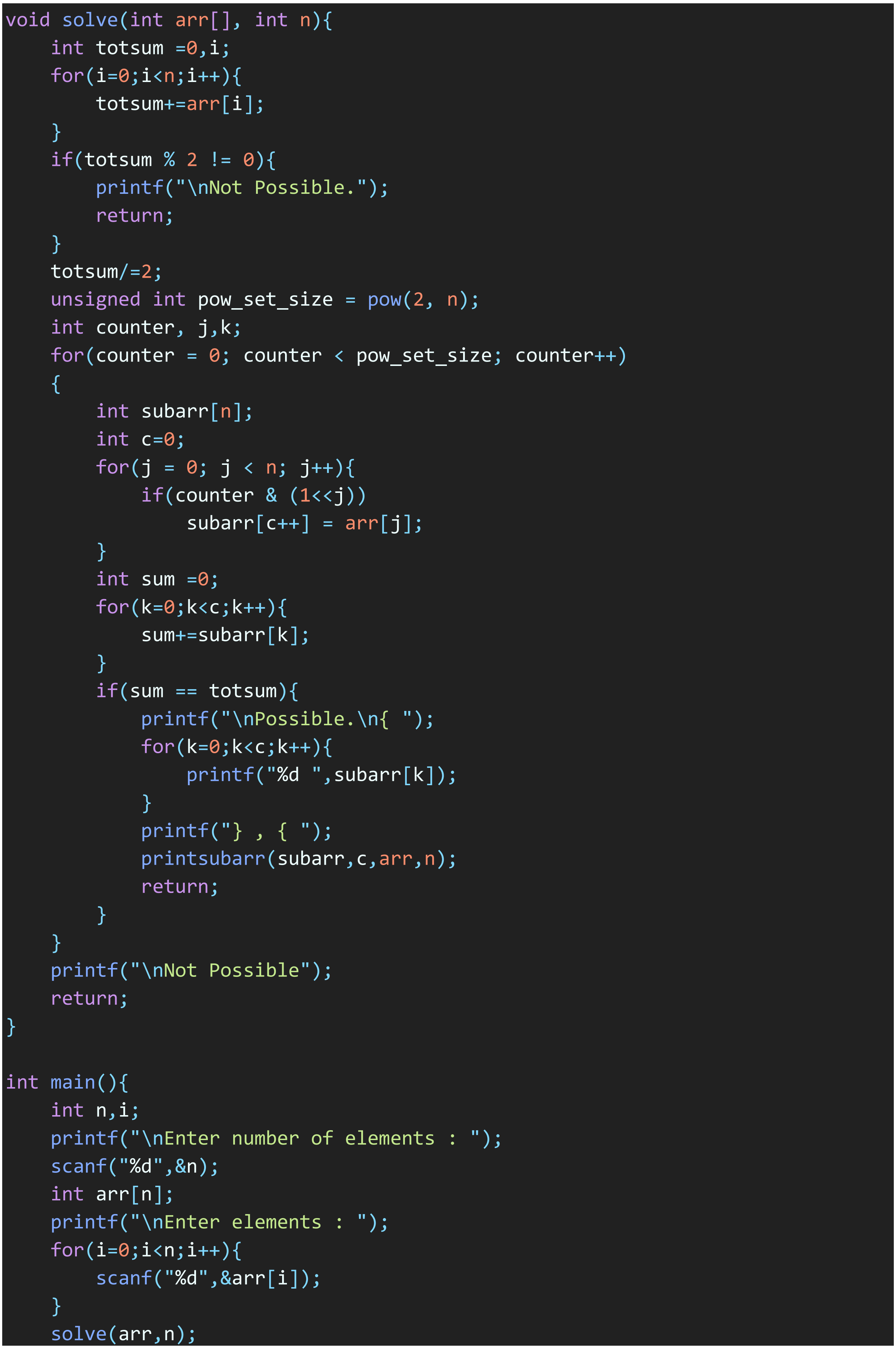
As observed from the algorithm, the best case is when the string is found at index 0 and its time complexity is **O(m)** where m is the length of the substring to be found/searched.

We can see that the worst case is when either the string is found at the maximum index possible or not found at all. Then its time complexity is **O(nm)** where m is the length of the substring to be searched and n is the length of the original input string to be searched from.

Q3) Write a program to implement solution to partition problem using brute-force technique and analyse its time efficiency theoretically. A partition problem takes a set of numbers and finds two disjoint sets such that the sum of the elements in the first set is equal to the second set. [Hint: You may generate power set]

CODE:

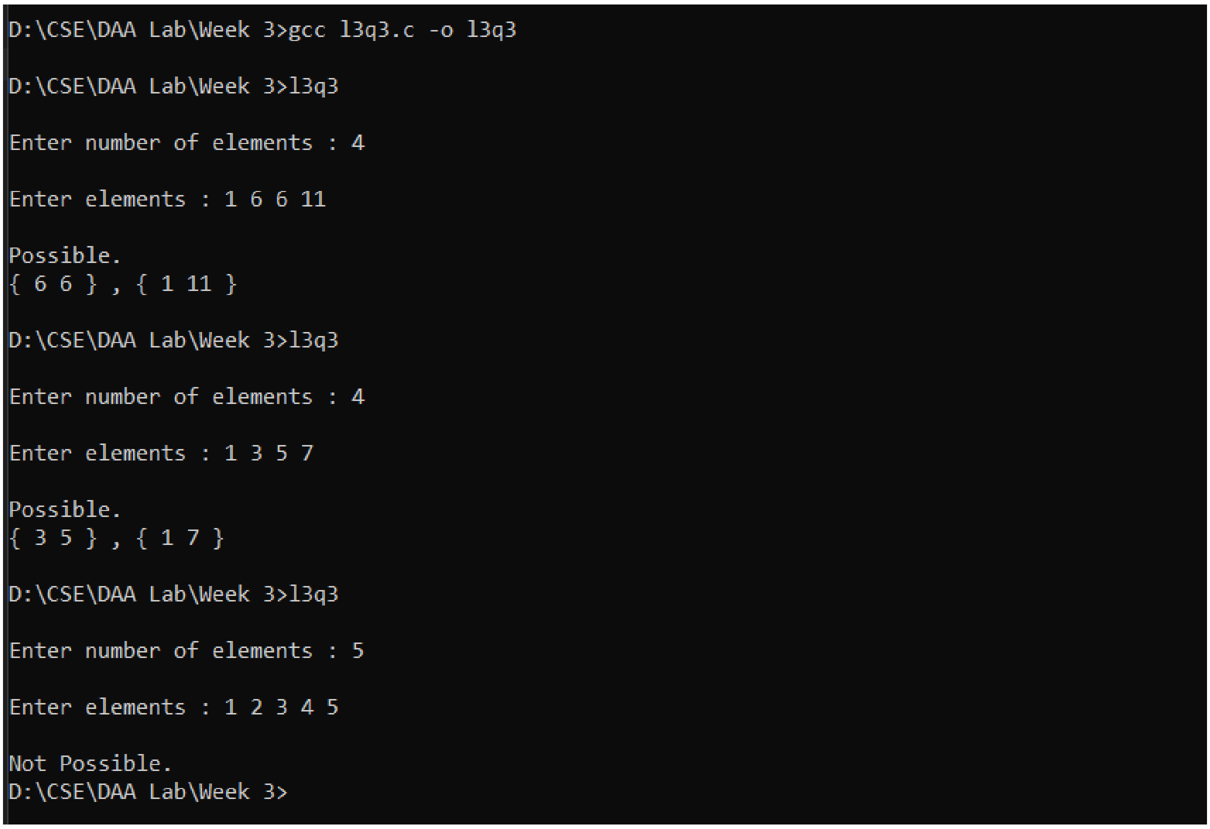
```
#include<stdio.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>

void printsubarr(int sub[],int ls,int mainarr[],int lm){
    int l=lm-ls;
    int i,j,k=0,flag;
    //int *comparr = (int*)calloc(l,sizeof(int));
    for(i=0;i<lm;i++){
        flag = 1;
        for(j=0;j<ls;j++){
            if(mainarr[i] == sub[j]){
                flag = 0;
                break;
            }
        }
        if(flag == 1){
            //comparr[k++] = mainarr[i];
            printf("%d ",mainarr[i]);
        }
    }
    printf("}\n");
}
```



```
void solve(int arr[], int n){
    int totsum =0,i;
    for(i=0;i<n;i++){
        totsum+=arr[i];
    }
    if(totsum % 2 != 0){
        printf("\nNot Possible.");
        return;
    }
    totsum/=2;
    unsigned int pow_set_size = pow(2, n);
    int counter, j,k;
    for(counter = 0; counter < pow_set_size; counter++)
    {
        int subarr[n];
        int c=0;
        for(j = 0; j < n; j++){
            if(counter & (1<<j))
                subarr[c++] = arr[j];
        }
        int sum =0;
        for(k=0;k<c;k++){
            sum+=subarr[k];
        }
        if(sum == totsum){
            printf("\nPossible.\n{ ");
            for(k=0;k<c;k++){
                printf("%d ",subarr[k]);
            }
            printf("} , { ");
            printsubarr(subarr,c,arr,n);
            return;
        }
    }
    printf("\nNot Possible");
    return;
}

int main(){
    int n,i;
    printf("\nEnter number of elements : ");
    scanf("%d",&n);
    int arr[n];
    printf("\nEnter elements : ");
    for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    solve(arr,n);
```

```
    return 0;
}
```

INPUT/OUTPUT :



```
D:\CSE\DAA Lab\Week 3>gcc l3q3.c -o l3q3

D:\CSE\DAA Lab\Week 3>l3q3

Enter number of elements : 4

Enter elements : 1 6 6 11

Possible.
{ 6 6 } , { 1 11 }

D:\CSE\DAA Lab\Week 3>l3q3

Enter number of elements : 4

Enter elements : 1 3 5 7

Possible.
{ 3 5 } , { 1 7 }

D:\CSE\DAA Lab\Week 3>l3q3

Enter number of elements : 5

Enter elements : 1 2 3 4 5

Not Possible.
D:\CSE\DAA Lab\Week 3>
```

Time Efficiency Analysis:

We can see from the algorithm, that we create the power set of the given input set. If the number of elements in the input is n then the number of elements in the power set is 2^n .

The recurrence relation of the algorithm is given by:

T(n) = 2 * T(n-1) This, on solving gives us the time complexity of this algorithm as **$O(2^n)$**

THE END