LAB 3 :

ALGORITHM : BubbleSort(A[0..n − 1])
Purpose : Sorts a given array by using bubble sort .
Input: An array A[0..n − 1] of orderable elements
Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 0 to n − 2 do
for j ← 0 to n − 2 − i do
if A[j + 1] < A[j] swap A[j] and A[j + 1]

CODE :pgm1.c

```
#include <stdio.h>
#include <stdlib.h>
void bubbleSort(int *ptr, unsigned int n)
{
unsigned int i;
unsigned int j;
unsigned int temp;
int opcount = 0;
for (i = 0; i < n - 1; i++)
{
for (j = 0; j < n - i - 1; j++)
{
opcount++;
if (ptr[j] > ptr[j + 1])
{
temp = ptr[j];
ptr[j] = ptr[j + 1];
ptr[j + 1] = temp;
}
}
}
printf("\n");
printf("\nOperation count is = %d ", opcount);
printf("\n");
```

```c
}
int main()
{
int *ptr;
int i;
int n;
int j;
printf("\nEnter the input size : \n");
scanf("%d", &n);
for (int j = 0; j < 4; j++)
{
ptr = (int *)malloc(sizeof(int) * n);
for (int k = 0; k < n; k++)
ptr[k] = n - k;
printf("\nThe Elements are : \n");
for (i = 0; i < n; i++)
printf("%d ", ptr[i]);
bubbleSort(ptr, n);
printf("\nAfter Sorting Bubble sort : \n");
for (i = 0; i < n; i++)
printf("%d", ptr[i]);
printf("\n");
free(ptr);
n = n + 6;
}
return 0;
}
```

1. Time complexity

$$T(n) = \sum_{i=0}^{n-1} \left[\sum_{i=0}^{n-i-1} [1] \right]$$

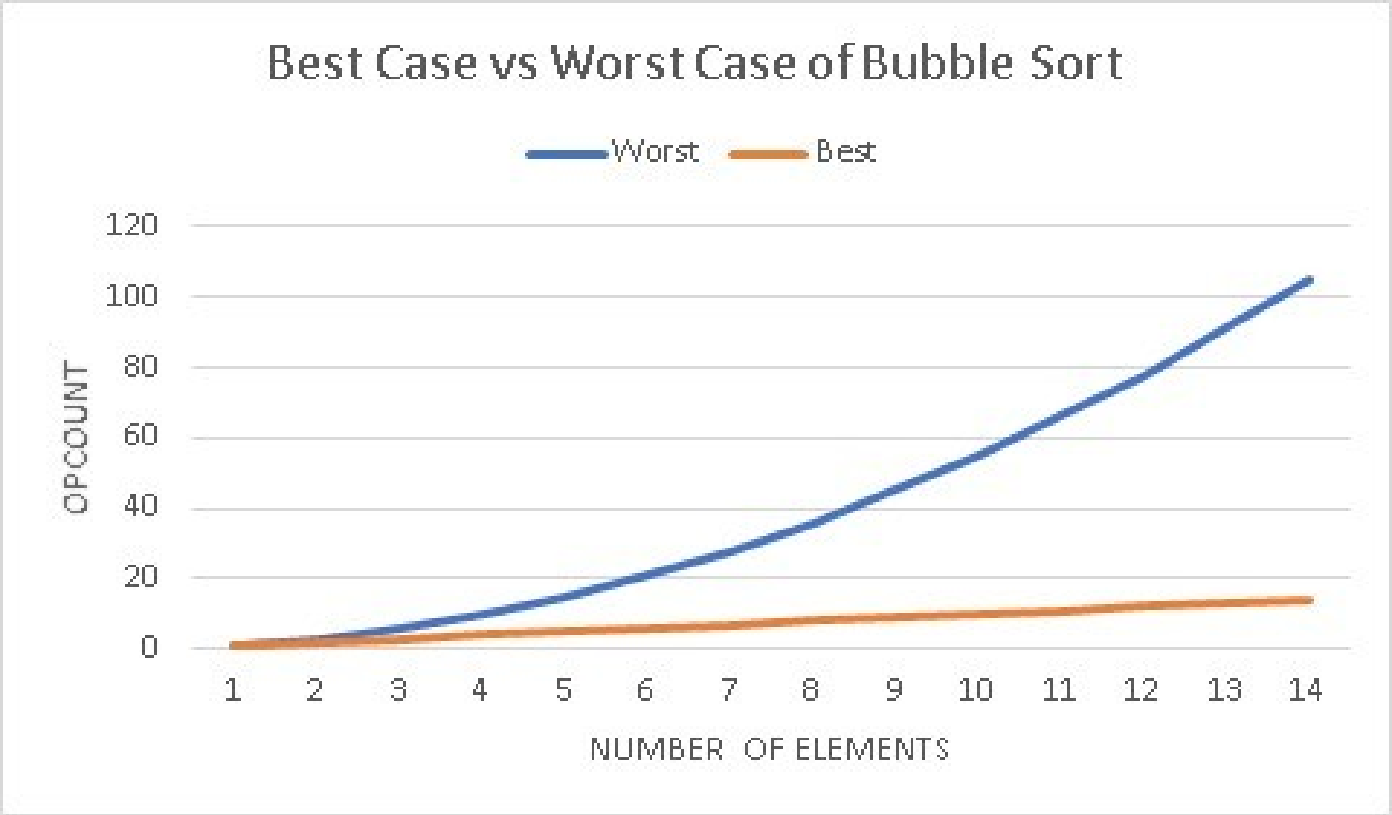
$$T(n) = \sum_{i=0}^{n-1} [(n-1) - (i+1) + 1]$$

$$T(n) = n(n-1)/2 \in O(n^2)$$

OUTPUT :

```c
#include <stdio.h>
#include <stdlib.h>
void bubbleSort(int *ptr, unsigned int n)
{
    unsigned int i;
    unsigned int j;
    unsigned int temp;
    int opcount = 0;
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            opcount++;
            if (ptr[j] > ptr[j + 1])
            {
                temp = ptr[j];
                ptr[j] = ptr[j + 1];
                ptr[j + 1] = temp;
            }
        }
    }
    printf("\n");
    printf("Operation count is = %d ", opcount);
    printf("\n");
}
int main()
{
    int *ptr;
    int i;
    int n;
    int j;
    printf("\nEnter the input size : \n");
    scanf("%d", &n);
    printf("\n");
    for (int j = 0; j < 4; j++)
    {
```

```
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab3$ gcc pgm2.c -o pgm2
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab3$ ./pgm2

Enter the input size :
5

The Elements are :  5 4 3 2 1

Operation count is = 10
After Sorting Bubble sort :1    2       3       4       5
The Elements are :  11 10 9 8 7 6 5 4 3 2 1

Operation count is = 55
After Sorting Bubble sort :1    2       3       4       5       6       7   8
9       10      11
The Elements are :  17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Operation count is = 136
After Sorting Bubble sort :1    2       3       4       5       6       7   8
9       10      11      12      13      14      15      16      17
The Elements are :  23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2
 1

Operation count is = 253
After Sorting Bubble sort :1    2       3       4       5       6       7   8
9       10      11      12      13      14      15      16      17      18  1
9       20      21      22      23
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab3$
```



2.

ALGORITHM : BruteForceStringMatch(T [0..n - 1], P[0..m - 1])

Purpose : Implements brute-force string matching

Input: An array T [0..n - 1] of n characters representing a text and an array P[0..m - 1] of m characters representing a pattern

Output : The index of the first character in the text that starts a
matching substring or -1 if the search is unsuccessful

```
for i ←0 to n - m do
      j ←0
     while j <m and P[j ]= T [i + j ] do
                j ←j + 1
      if j = m return i
return -1
```

pgm2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int stringMatching(char string1[], char string2[])
{
int i, j, opcount = 0;
int stringLength1 = strlen(string1);
int stringLength2 = strlen(string2);
for (i = 0; i ≤ stringLength1 - stringLength2; i++)
{
opcount++;
j = 0;
while ((j < stringLength2) & (string2[j] == string1[i + j]))
{
opcount++;
j++;
}
if (j == stringLength2)
```

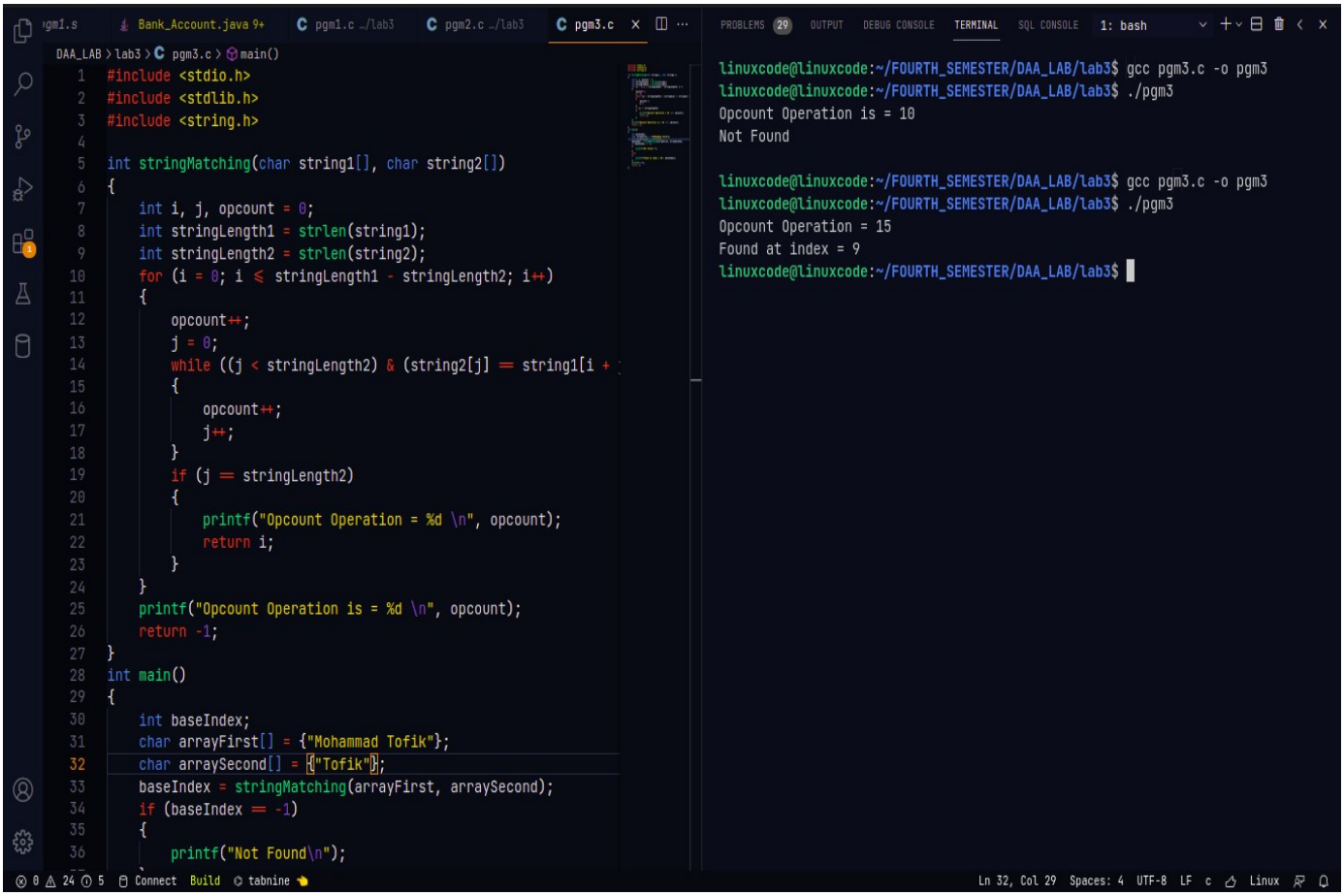
```
{
printf("Opcount Operation = %d \n", opcount);
return i;
}
}
printf("Opcount Operation is = %d \n", opcount);
return -1;
}
int main()
{
int baseIndex;
char arrayFirst[] = {"Mohammad Tofik"};
char arraySecond[] = {"Tofik"};
baseIndex = stringMatching(arrayFirst, arraySecond);
if (baseIndex == -1)
{
printf("Not Found\n");
}
else
{
printf("Found at index = %d", baseIndex);
}
printf("\n");
return 0;
}
```

2. Time complexity Analysis

$$T(n) = \sum_{i=0}^{n-m} [\sum_{i=0}^{m} [1]$$

$$T(n) = m(m-n+1) \in O(mn)$$

OUTPUT :



3. pgm3.c

```c
#include <stdbool.h>
#include <stdio.h>
bool isSubsetSum(int arr[], int n, int sum)
{
if (sum == 0)
return true;
if (n == 0 && sum ≠ 0)
return false;
if (arr[n - 1] > sum)
return isSubsetSum(arr, n - 1, sum);


return isSubsetSum(arr, n - 1, sum) || isSubsetSum(arr, n - 1, sum - arr[n -
1]);
```

```
}

bool findPartiion(int arr[], int n)
{
int sum = 0;
for (int i = 0; i < n; i++)
sum += arr[i];
if (sum % 2 ≠ 0)
return false;
return isSubsetSum(arr, n, sum / 2);
}
int main()
{
int arr[] = { 3, 1, 9, 12 };
int n = sizeof(arr) / sizeof(arr[0]);

if (findPartiion(arr, n) == true)
printf("TRUE");
else
printf("FALSE");
return 0;
}
```

(3) The running time $T(n)$ for partition algorithm is given by recurrence relation

$$T(n) = 2T(n-1)$$

By applying the method of backward substitution, we get

$$T(n) \in O(2^n)$$