**LAB-3**

**NAME – TUSHAR MAROO**

**SECTION – A**

**REGISTRATION NUMBER – 190905238**

**ROLL NUMBER – 40**

**Q1.**

**ALGORITHM**

Sorts a given array by 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**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int n, i, j, swapped, opcnt = 0;

printf("Enter the number of elements in the array : ");

scanf("%d", &n);

int A[n];

printf("Enter the elements of the array : ");

for (i = 0; i < n; i++)

scanf("%d", &A[i]);

for (i = 0; i < n - 1; i++)

{

swapped = 0;

for (j = 0; j < n - i - 1; j++)

{

opcnt++;

if (A[j] > A[j + 1])

{

int temp = A[j];

A[j] = A[j + 1];

A[j + 1] = temp;

swapped = 1;

}

}

if (!swapped)

break;

}

for (i = 0; i < n; i++)

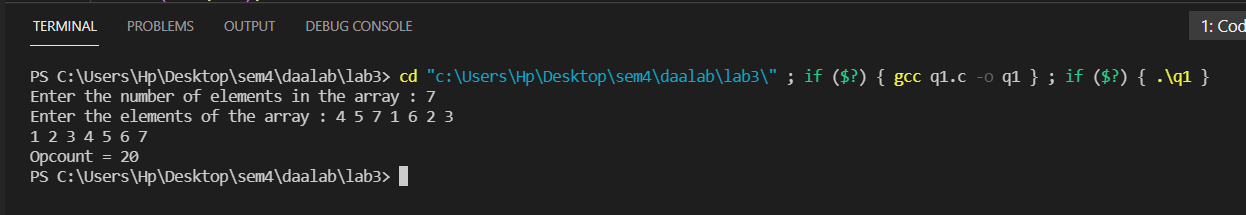
printf("%d ", A[i]);

printf("\nOpcount = %d", opcnt);

}

**GRAPH**

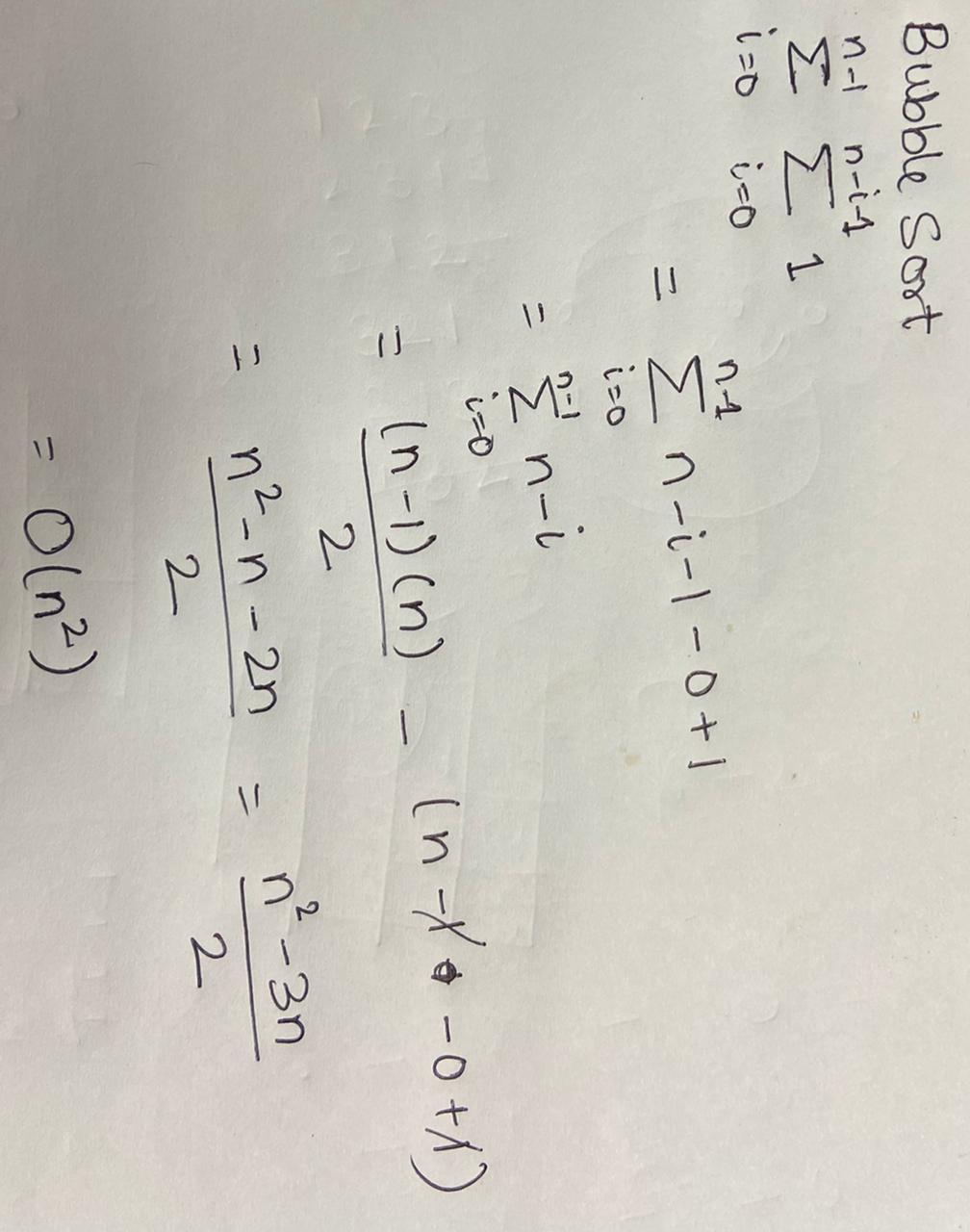
**OUTPUT**

****

**ANALYSIS**

Best case: When the array is sorted O(n)

Worst case: When the array is sorted in reverse order O(n^2)



**Q2.**

**ALGORITHM**

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

**CODE**

#include <stdio.h>

#include <string.h>

int main()

{

char str[100], substr[100];

printf("Enter the string : ");

scanf("%s", str);

printf("Enter the sub-string : ");

scanf("%s", substr);

int i, flag = 0, opcount = 0;

for (i = 0; i < strlen(str) - strlen(substr) + 1; i++)

{

opcount++;

if (str[i] != substr[0])

continue;

int j = 1;

while (substr[j] != '\0' && substr[j] == str[i + j])

{

j++;

opcount++;

}

if (substr[j] == '\0')

{

flag = 1;

break;

}

}

if (flag)

printf("Substring found at index %d.\n", i);

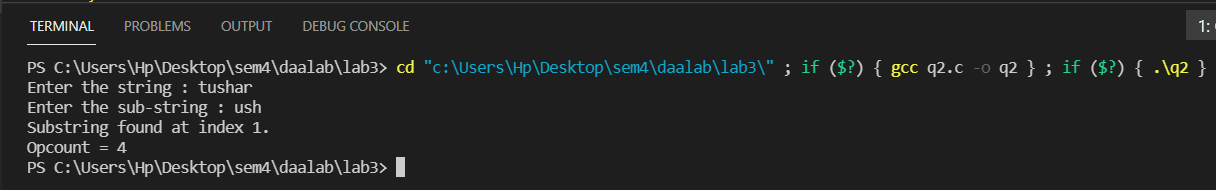
else

printf("Substring not found.\n");

printf("Opcount = %d",opcount);

}

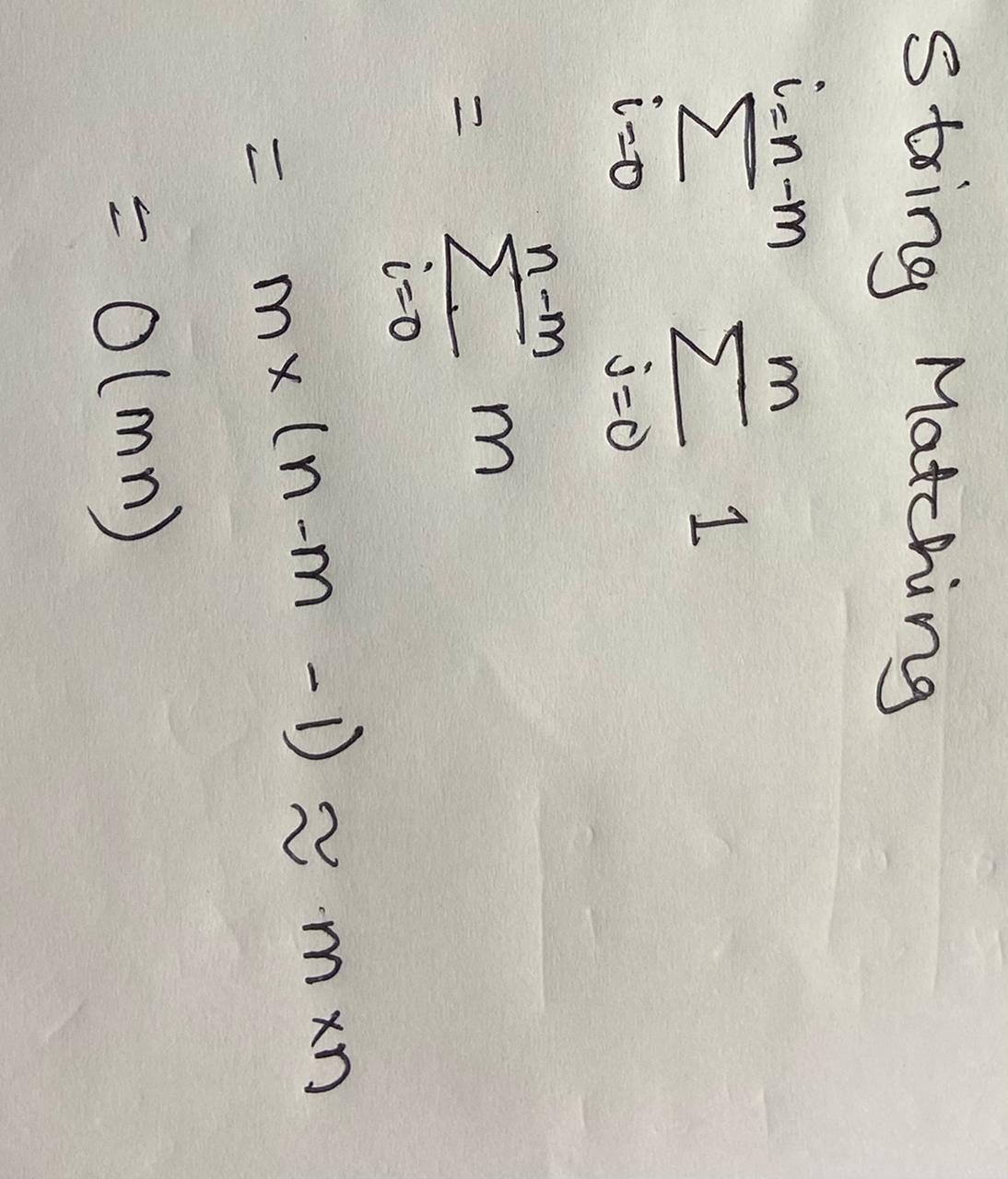
**OUTPUT**

****

**ANALYSIS**

Best case: When the substring is present at index 0. Time complexity O(m) m being the length of the substring

Worst case: When substring is present at index n-m of the string. Time complexity O(mn) m being length of substring and n being the length of the string.



**Q3.**

**CODE**

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

typedef enum

{

false,

true

} bool;

int size;

int opcount;

int \*kc;

int \*dkc;

bool subset(int arr[], int n, int sum, int a[])

{

int \*k;

int \*dk;

k = (int \*)calloc(size, sizeof(int));

dk = (int \*)calloc(size, sizeof(int));

for (int i = 0; i < size; i++)

{

k[i] = a[i];

dk[i] = a[i];

}

if (sum == 0)

{

printf("Possible to partition\n");

printf("Subset 1: ");

for (int i = 0; i < size; i++)

{

if (a[i])

printf("%d ", a[i]);

}

printf("\nSubset 2: ");

int flag;

for (int i = 0; i < size; i++)

{

flag = 1;

for (int j = 0; j < size; j++)

{

if (arr[i] == a[j])

flag = 0;

}

if (flag)

printf("%d ", arr[i]);

}

printf("\n");

return true;

}

if (n == 0 && sum != 0)

return false;

dk[n - 1] = 0;

opcount++;

return subset(arr, n - 1, sum, k) || subset(arr, n - 1, sum - arr[n - 1], dk);

}

bool partition(int arr[], int n)

{

int sum = 0;

kc = (int \*)calloc(size, sizeof(int));

dkc = (int \*)calloc(size, sizeof(int));

for (int i = 0; i < size; i++)

{

kc[i] = arr[i];

dkc[i] = arr[i];

}

for (int i = 0; i < n; i++)

sum += arr[i];

if (sum % 2 != 0)

return false;

return subset(arr, n, sum / 2, arr);

}

int main()

{

printf("Enter size of set: \n");

scanf("%d", &size);

int \*arr = (int \*)calloc(size, sizeof(int));

printf("Enter the set: \n");

for (int j = 0; j < size; j++)

{

scanf("%d", &arr[j]);

}

opcount = 0;

bool flag = partition(arr, size);

if (flag)

printf("");

else

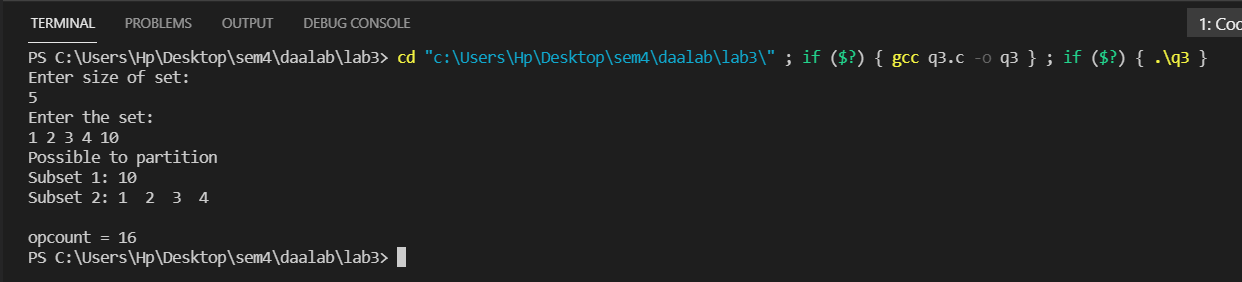
printf("Not possible to partition\n");

printf("\nopcount = %d\n", opcount);

return 0;

}

**OUTPUT**

****

**ANALYSIS**

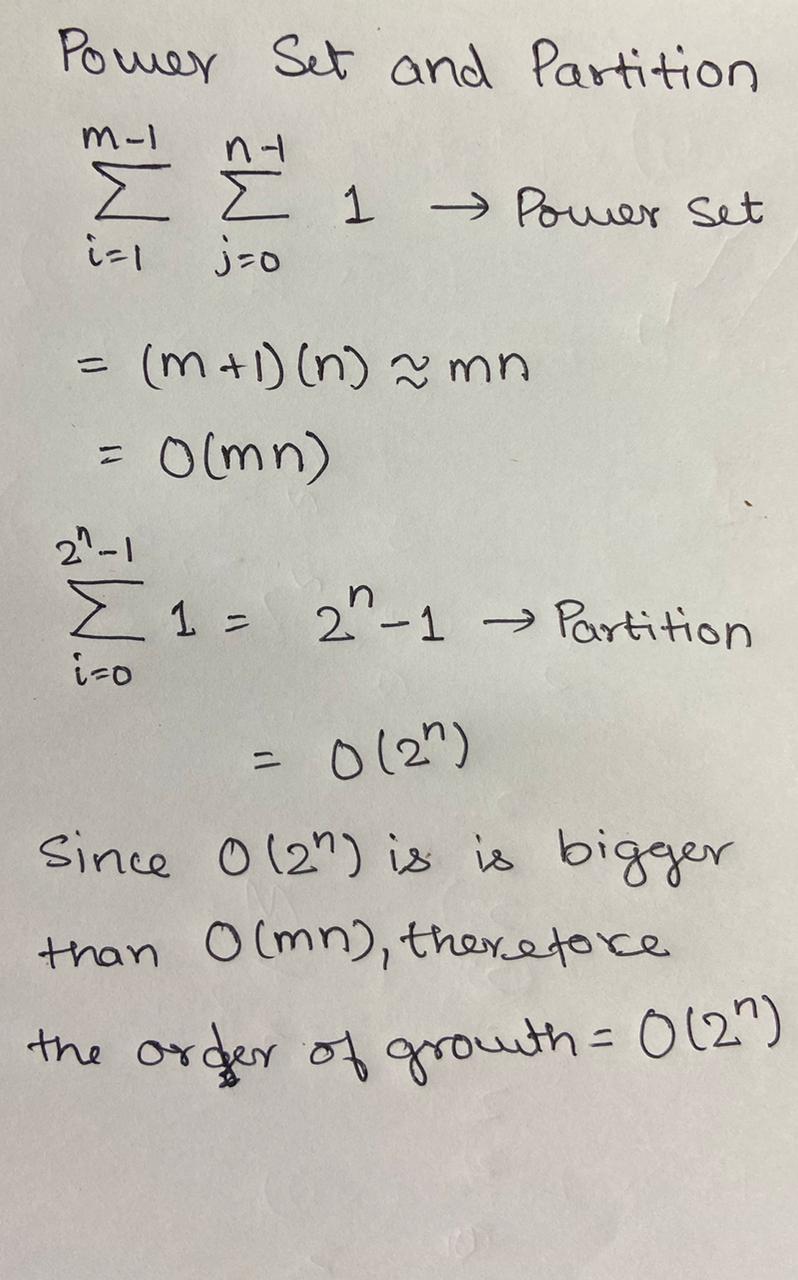
The running time T(n) for partition algorithm is given by recurrence relation:

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

By applying the method of backward substitution, we get:

T(n) ∈ O(2^n)

Time complexity for best, worst and average case is the same.

****