

LABORATORY REPORT

**Algorithm Laboratory (CS-39001)**

**B.Tech Program in ECSc**

Submitted By

**Name:- SANNIDHI DEB**

**Roll No: 2330044**



**Kalinga Institute of Industrial Technology  
(Deemed to be University) Bhubaneswar, India**

Autumn, 2025

# Table of Contents

[illegible]

Experiment Number	1.1
Experiment Title	<p>Aim of the program: Write a program to find out the second smallest and second largest element stored in an array of n integers.</p> <p>Input: Size of the array is 'n' and read 'n' number of elements from a disc file.</p> <p>Output: Second smallest, Second largest</p>
Date of Experiment	24/07/2025
Date of Submission	06/08/2025

## 1. Algorithm:-

Sannidhi Deb (2330044)

1.1: SECOND\_LARGEST\_AND\_SMALLEST(A)

Input: An array A

Output: Second smallest and second largest elements.

1. smallest  $\leftarrow A[1]$
2. largest  $\leftarrow A[1]$
3. initialise second-smallest, second-largest
4. for  $i \leftarrow 2$  to A.length
5. if  $A[i] < \text{smallest}$  then
6. second-smallest  $\leftarrow \text{smallest}$
7. smallest  $\leftarrow A[i]$
8. else if  $A[i] < \text{second-smallest}$  and  $A[i] \neq \text{smallest}$  then
9. second-smallest  $\leftarrow A[i]$
10. for  $i \leftarrow 2$  to A.length
11. if  $A[i] > \text{largest}$  then
12. second-largest  $\leftarrow \text{largest}$
13. largest  $\leftarrow A[i]$
14. else if  $A[i] > \text{second-largest}$  and  $A[i] \neq \text{largest}$  then
15. second-largest  $\leftarrow A[i]$
16. print second-smallest, second-largest

## 2. Code:-

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

void findSecondMinMax(int A[], int n) {
    int smallest = A[0], largest = A[0];
    int second_smallest = INT_MAX, second_largest = INT_MIN;

    for (int i = 1; i < n; i++) {
        if (A[i] < smallest) {
            second_smallest = smallest;
            smallest = A[i];
        } else if (A[i] < second_smallest && A[i] != smallest) {
            second_smallest = A[i];
        }
    }

    for (int i = 1; i < n; i++) {
        if (A[i] > largest) {
            second_largest = largest;
            largest = A[i];
        } else if (A[i] > second_largest && A[i] != largest) {
            second_largest = A[i];
        }
    }

    printf("\nSannidhi Deb\n 2330044\n\n");

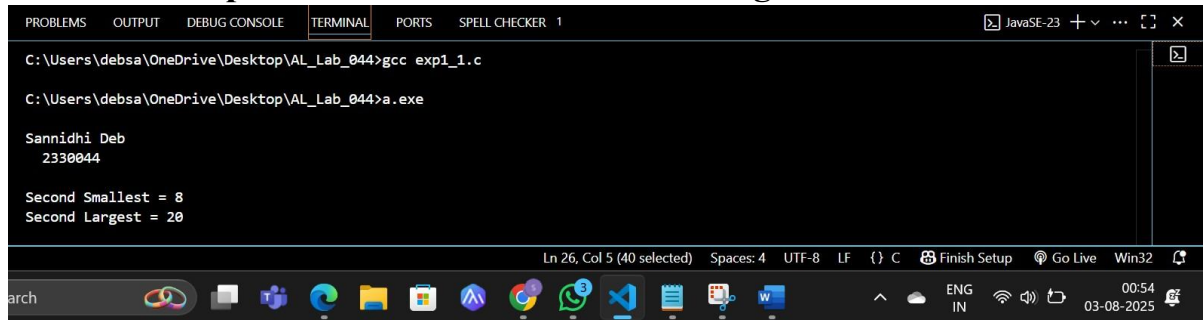
    printf("Second Smallest = %d\n", second_smallest);
    printf("Second Largest = %d\n", second_largest);
}

int main() {
    int A[] = {10,
5, 20, 8, 30};
    int n = sizeof(A) / sizeof(A[0]);

    findSecondMinMax(A, n);

    return 0;
}
```

### 3. Results/Output:- Entire Screen Shot including Date & Time:-



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 1
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>gcc exp1_1.c
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>a.exe

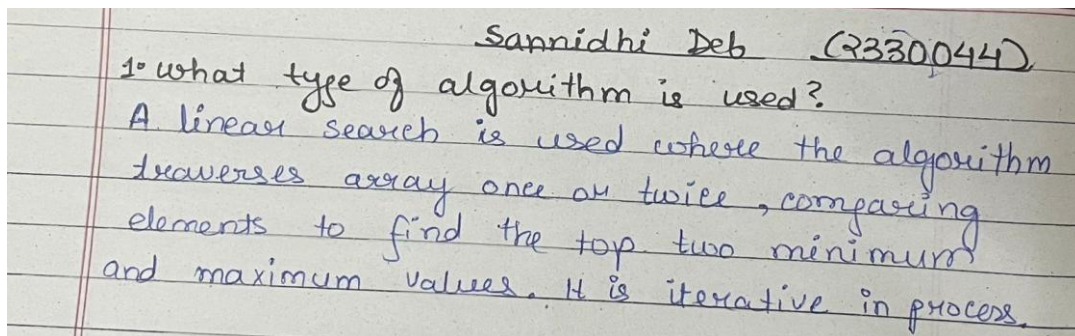
Sannidhi Deb
2330044

Second Smallest = 8
Second Largest = 20

Ln 26, Col 5 (40 selected) Spaces: 4 UTF-8 LF {} C Finish Setup Go Live Win32
03-08-2025 00:54
```

### 4. Remarks:-

1. What type of algorithm is used?

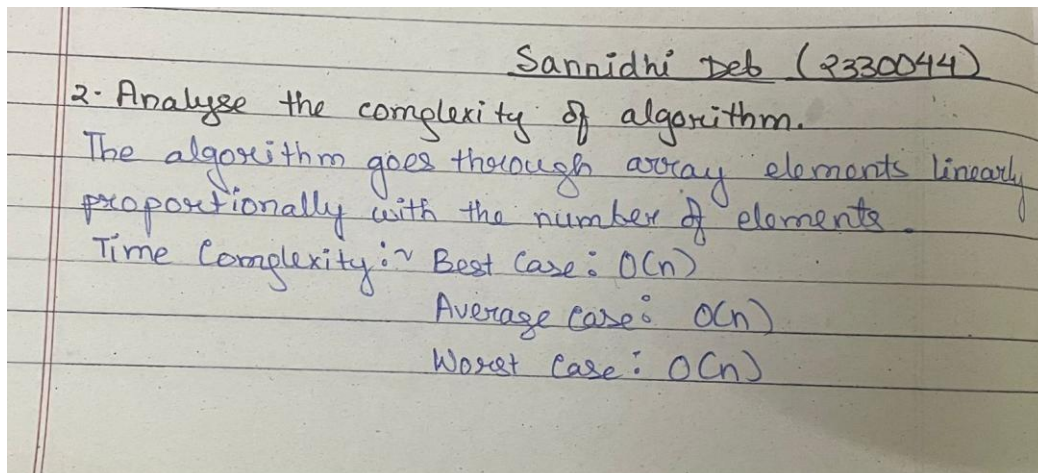


Sannidhi Deb (2330044)

1. what type of algorithm is used?

A linear search is used where the algorithm traverses array once or twice, comparing elements to find the top two minimum and maximum values. It is iterative in process.

2. Analyze the complexity of your algorithm.



Sannidhi Deb (2330044)

2. Analyse the complexity of algorithm.

The algorithm goes through array elements linearly proportionally with the number of elements.

Time Complexity: ~ Best Case:  $O(n)$   
Average Case:  $O(n)$   
Worst Case:  $O(n)$

3. Any other observations?

Sannidhi Deb (2330044)

3. Any other observation.  
The algorithm finds second smallest and second largest elements without sorting the array, using linear traversal only.

Sannidhi Deb  
(2330044)

---

Sannidhi Deb

Signature of the FIC

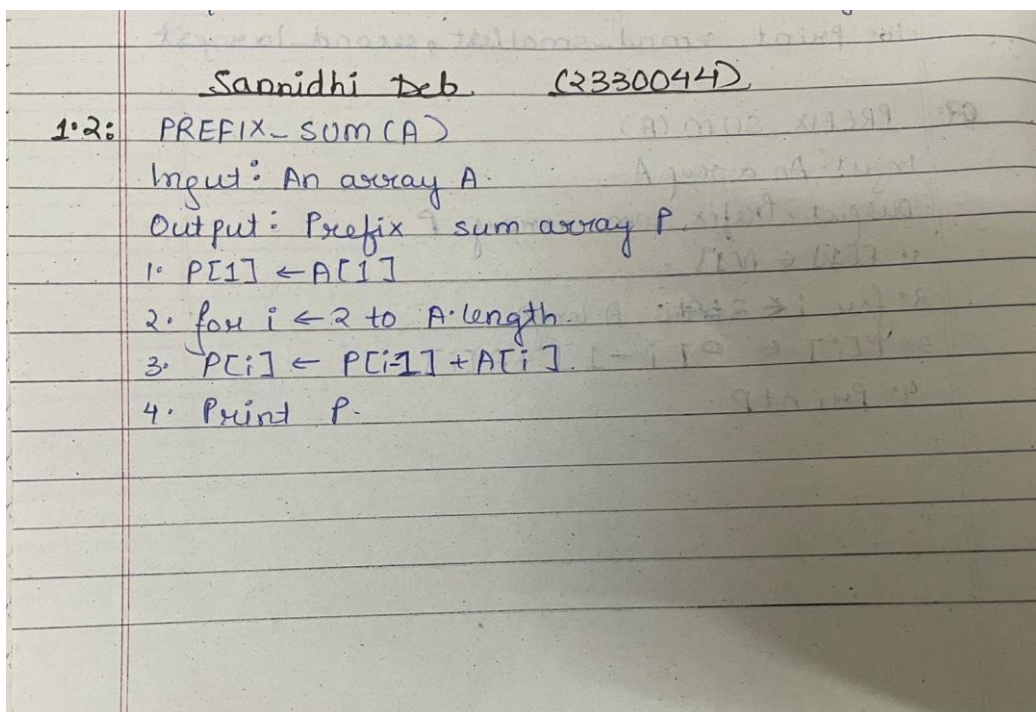
---

(Name of the FIC)



<b>Experiment Number</b>	1.2
<b>Experiment Title</b>	<p>Aim of the program: Given an array arr[] of size N, find the prefix sum of the array. A prefix sum array is another array prefixSum[] of the same size, such that the value of prefixSum[i] is arr[0] + arr[1] + arr[2] . . . arr[i].</p> <p>Input Array: 3 4 5 1 2</p> <p>Output Array: 3 7 12 13 15</p>
<b>Date of Experiment</b>	24/07/2025
<b>Date of Submission</b>	06/08/2025

## 1. Algorithm:-



## 2. Code:-

```
#include <stdio.h> void
prefixSum(int arr[], int n) {
    int prefix[n];
    prefix[0] = arr[0];

    for(int i = 1; i < n; i++) {
        prefix[i] = prefix[i - 1] + arr[i];
    }
    printf("\nSannidhi Deb\n 2330044\n\n ");
    printf("Input Array: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    printf("\nOutput (Prefix Sum) Array: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", prefix[i]);
    }
    printf("\n");
}

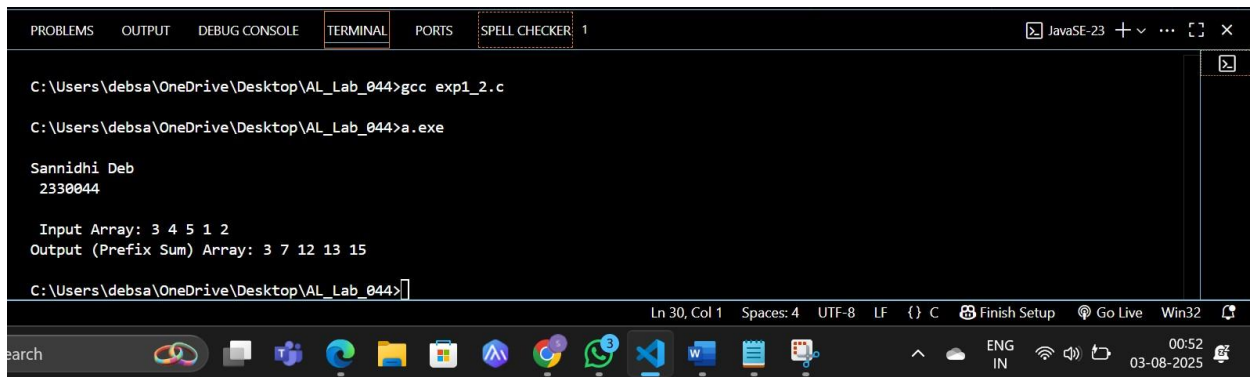
int main() {    int arr[] =
{3, 4, 5, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);

    prefixSum(arr, n);

    return 0;
}
```



### 3. Results/Output:- Entire Screen Shot including Date & Time:-



```
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>gcc exp1_2.c
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>a.exe

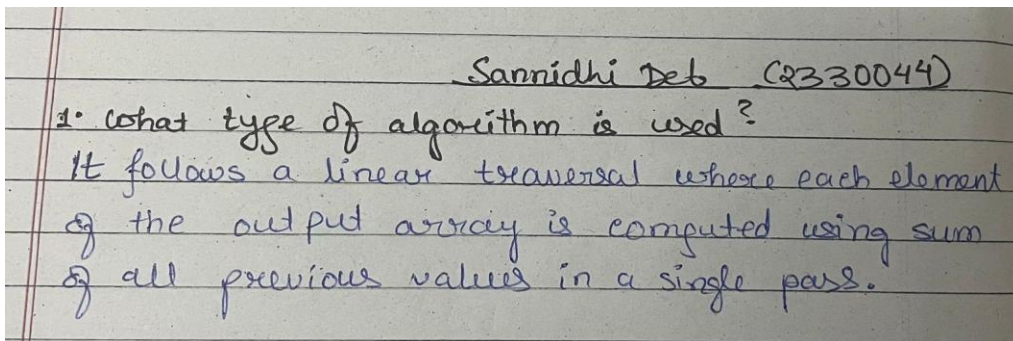
Sannidhi Deb
2330044

Input Array: 3 4 5 1 2
Output (Prefix Sum) Array: 3 7 12 13 15

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>
```

### 4. Remarks:-

1. What type of algorithm is used?

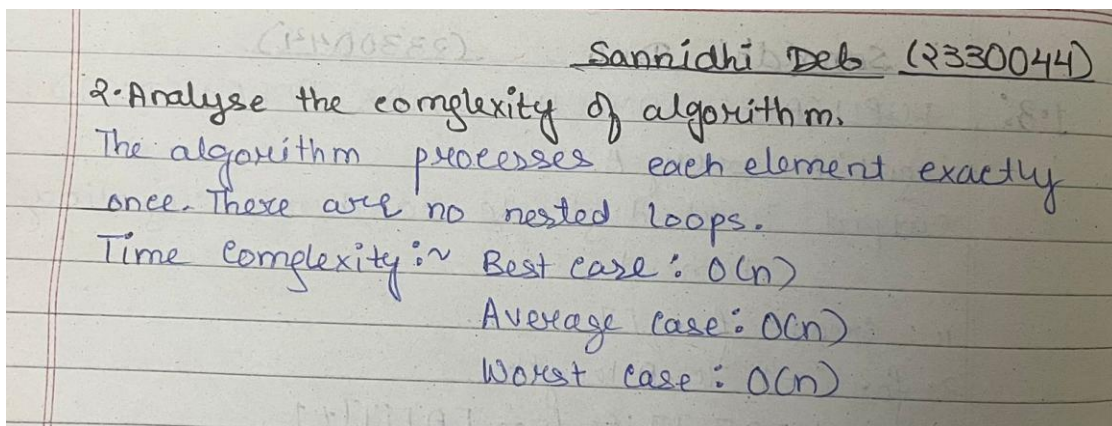


Sannidhi Deb (2330044)

1. What type of algorithm is used?

It follows a linear traversal where each element of the output array is computed using sum of all previous values in a single pass.

2. Analyze the complexity of your algorithm.



Sannidhi Deb (2330044)

2. Analyse the complexity of algorithm.

The algorithm processes each element exactly once. There are no nested loops.

Time complexity: ~ Best case:  $O(n)$   
Average case:  $O(n)$   
Worst case:  $O(n)$

3. Any other observations?

Sannidhi Deb (2330044)

3. Any other observation.

The prefix sum algorithm computes cumulative sums in a single pass, which is useful for solving problems quickly.

Sannidhi Deb  
(2330044)

---

Sannidhi Deb

Signature of the FIC

---

(Name of the FIC)

<b>Experiment Number</b>	1.3
<b>Experiment Title</b>	<p>Aim of the program: Write a program to read 'n' integers from a disc file that must contain some duplicate values and store them into an array. Perform the following operations on the array.</p> <p>a) Find out the total number of duplicate elements.</p> <p>b) Find out the most repeating element in the array. Input: Enter how many numbers you want to read from file: 15 Output: The content of the array: 10 40 35 47 68 22 40 10 98 10 50 35 68 40 10 Total number of duplicate values = 4 The most repeating element in the array = 10</p>
<b>Date of Experiment</b>	31/07/2025
<b>Date of Submission</b>	06/08/2025

## 1. Algorithm:-

Sannidhi Deb (2330044)

1.3: DUPLICATE-COUNT-AND-MAX-REPEATED(A)

Input: An array A

Output: Total duplicate count, most repeating element.

1. Create freq[]
2. for  $i \leftarrow 1$  to A.length.
3.  $\text{freq}[A[i]] \leftarrow \text{freq}[A[i]] + 1$ .
4. duplicate-count  $\leftarrow 0$
5. max\_freq  $\leftarrow 0$
6. most-repeated  $\leftarrow A[1]$
7. for each key in freq[]
8. if  $\text{freq}[\text{key}] > 1$  then.
9. duplicate-count  $\leftarrow \text{duplicate-count} + 1$
10. if  $\text{freq}[\text{key}] > \text{max\_freq}$  then.
11. max\_freq  $\leftarrow \text{freq}[\text{key}]$ .
12. most-repeated  $\leftarrow \text{key}$ .
13. print duplicate-count, most-repeated.

## 2. Code:-

```
#include <stdio.h>
#define MAX 1000
int
main() {
    int A[100], freq[MAX] = {0};
    int n;
    printf("\nSannidhi Deb\n 2330044\n\n");
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &A[i]);
        freq[A[i]]++;
    }
    int duplicate_count = 0;
    int max_freq = 0;
    int most_repeated = A[0];

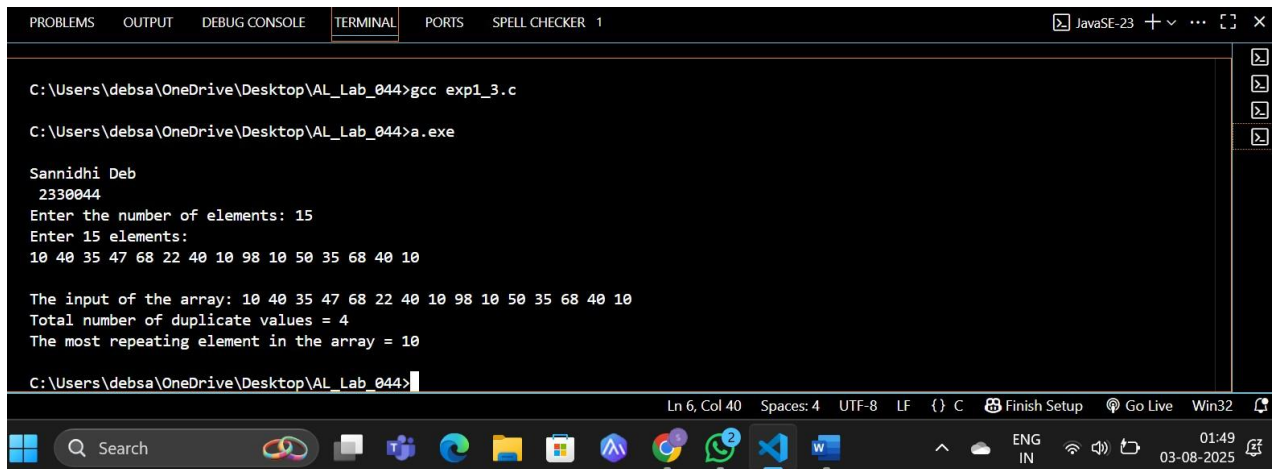
    for (int key = 0; key < MAX; key++) {
        if (freq[key] > 1) {
            duplicate_count++;
        }
        if (freq[key] > max_freq) {
            max_freq = freq[key];
            most_repeated = key;
        }
    }
    printf("\nThe input of the array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", A[i]);
    }

    printf("\nTotal number of duplicate values = %d", duplicate_count);
    printf("\nThe most repeating element in the array = %d\n", most_repeated);

    return 0;
}
```



### 3. Results/Output:- Entire Screen Shot including Date & Time:-



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 1 JavaSE-23 + ... [ ] X

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>gcc exp1_3.c

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>a.exe

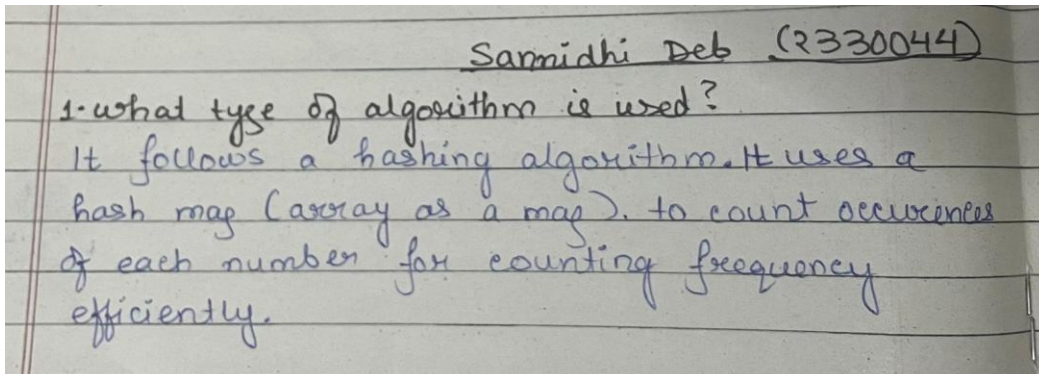
Sannidhi Deb
2330044
Enter the number of elements: 15
Enter 15 elements:
10 40 35 47 68 22 40 10 98 10 50 35 68 40 10

The input of the array: 10 40 35 47 68 22 40 10 98 10 50 35 68 40 10
Total number of duplicate values = 4
The most repeating element in the array = 10

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>
```

### 4. Remarks:-

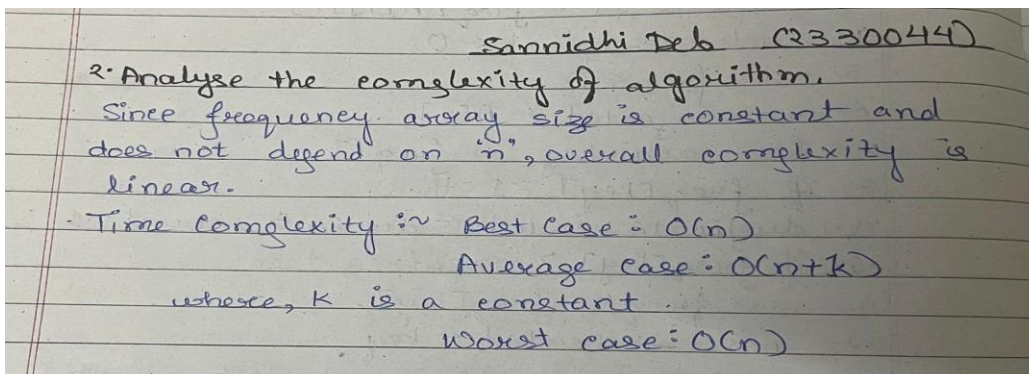
1. What type of algorithm is used?



Sannidhi Deb (2330044)

1. what type of algorithm is used?  
It follows a hashing algorithm. It uses a hash map (array as a map), to count occurrences of each number for counting frequency efficiently.

2. Analyze the complexity of your algorithm.



Sannidhi Deb (2330044)

2. Analyse the complexity of algorithm.  
Since frequency array size is constant and does not depend on  $n$ , overall complexity is linear.

Time Complexity: ~ Best case:  $O(n)$   
Average case:  $O(n+k)$   
Worst case:  $O(n)$   
where,  $k$  is a constant.

### 3. Any other observations?

Sannidhi Deb (2330044)

3. Any other observation.

By using frequency map, algorithm accurately identifies both number of duplicate values and most frequently occurring element in linear time.

Sannidhi Deb  
(2330044)

---

Sannidhi Deb

Signature of the FIC

---

(Name of the FIC)

<b>Experiment Number</b>	1.4
<b>Experiment Title</b>	<p>Aim of the program: Write a function to ROTATE_RIGHT (p1, p2) right an array for first p2 elements by 1 position using EXCHANGE (p, q) function that swaps/exchanges the numbers p &amp; q. Parameter p1 be the starting address of the array and p2 be the number of elements to be rotated.</p> <p>Input:  Enter an array A of size N (9): 11 22 33 44 55 66 77 88 99  Call the function ROTATE_RIGHT (A, 5)  Output:  Before ROTATE: 11 22 33 44 55 66 77 88 99  After ROTATE: 55 11 22 33 44 66 77 88 99</p>
<b>Date of Experiment</b>	01/07/2025
<b>Date of Submission</b>	06/08/2025

## 1. Algorithm:-

Sannidhi Deb (2330044)

1.4: ROTATE\_RIGHT (A, p2)

Input: Array A of size N, and number of elements to rotate = p2.

Output: Modified array after right rotation of first p2 elements.

1. temp  $\leftarrow A[p2]$
2. for  $i \leftarrow p2$  down to 1.
3.  $A[i] \leftarrow A[i-1]$
4.  $A[i] \leftarrow temp$
5. print A.



## 2. Code:-

```
#include <stdio.h> void
EXCHANGE(int *p, int *q) {
    int temp = *p;    *p = *q;
    *q = temp;
}

void ROTATE_RIGHT(int *p1, int p2) {
    for (int i = p2 - 1; i > 0; i--) {
        EXCHANGE(&p1[i], &p1[i - 1]);
    }
}

int main() {
    int A[9] = {11, 22, 33, 44, 55, 66, 77, 88, 99};
    int N = 9;

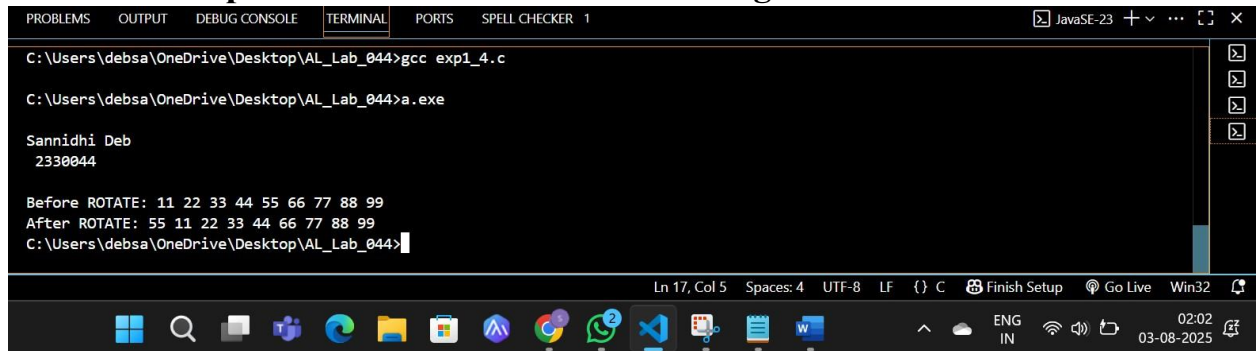
    printf("\nSannidhi Deb\n 2330044\n\n");
    printf("Before ROTATE: ");    for (int i = 0;
    i < N; i++) {
        printf("%d ", A[i]);
    }

    ROTATE_RIGHT(A, 5);

    printf("\nAfter ROTATE: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", A[i]);
    }

    return 0;
}
```

### 3. Results/Output:- Entire Screen Shot including Date & Time:-



```
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>gcc exp1_4.c

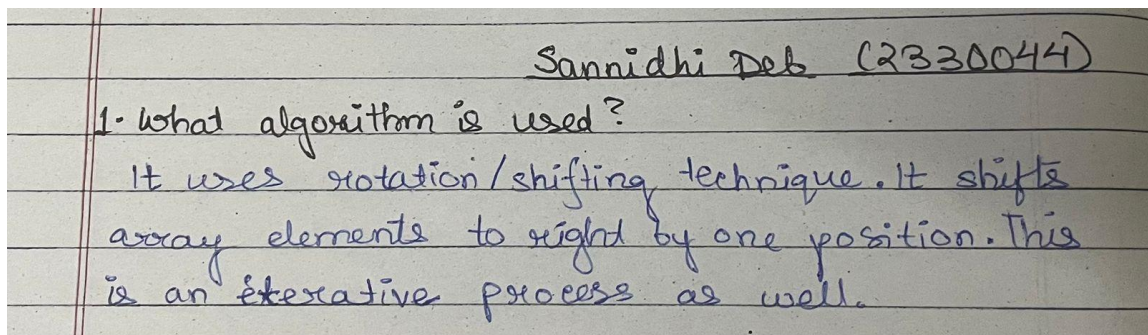
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>a.exe

Sannidhi Deb
2330044

Before ROTATE: 11 22 33 44 55 66 77 88 99
After ROTATE: 55 11 22 33 44 66 77 88 99
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>
```

### 4. Remarks:-

1. What type of algorithm is used?

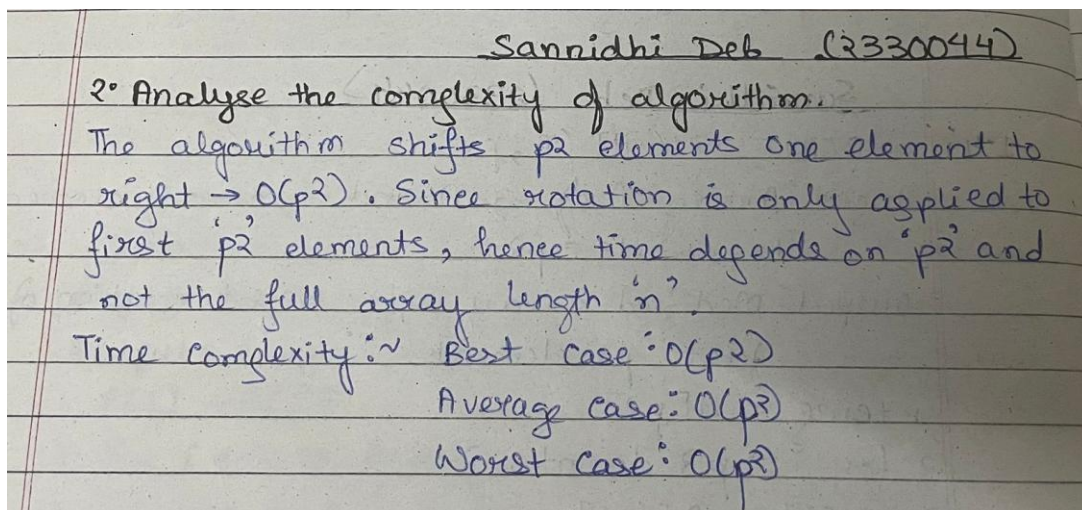


Sannidhi Deb (2330044)

1. What algorithm is used?

It uses rotation/shifting technique. It shifts array elements to right by one position. This is an iterative process as well.

2. Analyze the complexity of your algorithm.



Sannidhi Deb (2330044)

2. Analyze the complexity of algorithm.

The algorithm shifts  $p_2$  elements one element to right  $\rightarrow O(p_2)$ . Since rotation is only applied to first  $p_2$  elements, hence time depends on  $p_2$  and not the full array length  $n$ .

Time complexity: ~ Best case:  $O(p_2)$   
Average case:  $O(p_2)$   
Worst case:  $O(p_2)$

### 3. Any other observations?

Sannidhi Deb (233044)

3. Any other observation.  
The right rotation of the first  $p-2$  elements is performed using temporary variable, that avoids need for additional arrays and minimizes space usage.

### 5. Conclusion:-

In this lab session, we revised essential array-based operations in C programming. We implemented programs to find the second smallest and second largest elements, calculate prefix sums, detect duplicate values and identify the most frequent one, and perform partial array rotation using a swapping function. These exercises reinforced key data structure concepts such as array traversal, in-place updates, and efficient element manipulation — all fundamental for building more complex algorithms.

Sannidhi Deb  
(2330044)

---

Sannidhi Deb

Signature of the FIC

---

(Name of the FIC)

