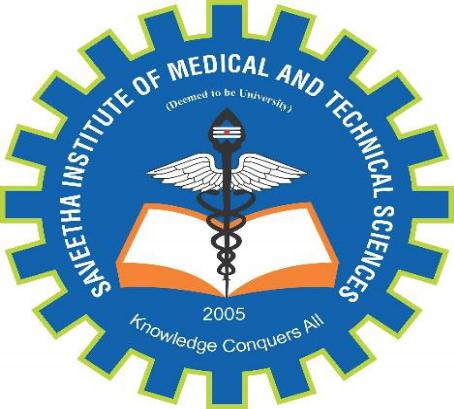
CAPSTONE PROJECT

**Minimum Number of Groups to Create a Valid Assignment**

**CSA0650-** DESIGN ANALYSIS AND ALGORITHMS FOR AMORTIZED ANALYSIS

SAVEETHA SCHOOL OF ENGINEERING

SIMATS ENGINEERING



Supervisor

Dr. R. Dhanalakshmi

Done by

Dhanush Vel Nithi.M (192211010)

1

**Minimum Number of Groups to Create a Valid Assignment**

**PROBLEM STATEMENT:**

Determine the minimum number of groups required where indices in each group have identical values. Additionally, the difference in size between any two groups must not exceed 1. For example, with nums = [3, 2, 3, 2, 3], the output is 2, as grouping can be done into two groups with acceptable size differences. The challenge involves efficiently grouping while maintaining balance in group sizes.

**ABSTRACT:**

The project aims to devise an algorithm to group indices of an array such that all indices in a group have the same value, and the size difference between any two groups is minimal. The solution involves counting occurrences, determining the maximum frequency, and calculating the number of groups needed. This approach is applicable to balancing and clustering problems in various fields, ensuring optimal group sizes.

**INTRODUCTION:**

The problem requires dividing array indices into groups where values are identical and the group sizes are nearly balanced. This ensures that the difference in sizes between groups is at most 1. The significance of this problem lies in its applications in data clustering, scheduling, and resource allocation where balanced grouping is crucial. The solution needs to be both efficient and straightforward.

**CODING:**

To calculates the minimum number of groups required by. Counting the frequency of each value in the array. Determining the maximum frequency. Calculating the number of groups based on these frequencies. Using ceiling division to ensure balanced groups. The code includes memory management and error handling to ensure robustness, printing results along with user information.

This dynamic programming solution is implemented as follows:

**C-programming**

#include <stdio.h>

#include <stdlib.h>

int minGroups(int nums[], int n) {

int maxVal = 0;

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

if (nums[i] > maxVal) {

maxVal = nums[i];

}

}

int \*count = (int \*)calloc(maxVal + 1, sizeof(int));

if (count == NULL) {

fprintf(stderr, "Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

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

count[nums[i]]++;

}

int maxFreq = 0;

for (int i = 0; i <= maxVal; i++) {

if (count[i] > maxFreq) {

maxFreq = count[i];

}

}

int groups = 0;

for (int i = 0; i <= maxVal; i++) {

if (count[i] > 0) {

groups += (count[i] + maxFreq - 1) / maxFreq; // Ceiling division

}

}

free(count);

return groups;

}

int main() {

int nums[] = {3, 2, 3, 2, 3};

int n = sizeof(nums) / sizeof(nums[0]);

int result = minGroups(nums, n);

printf("Name: Dhanush Vel Nithi\n");

printf("Registration Number: 192211010\n");

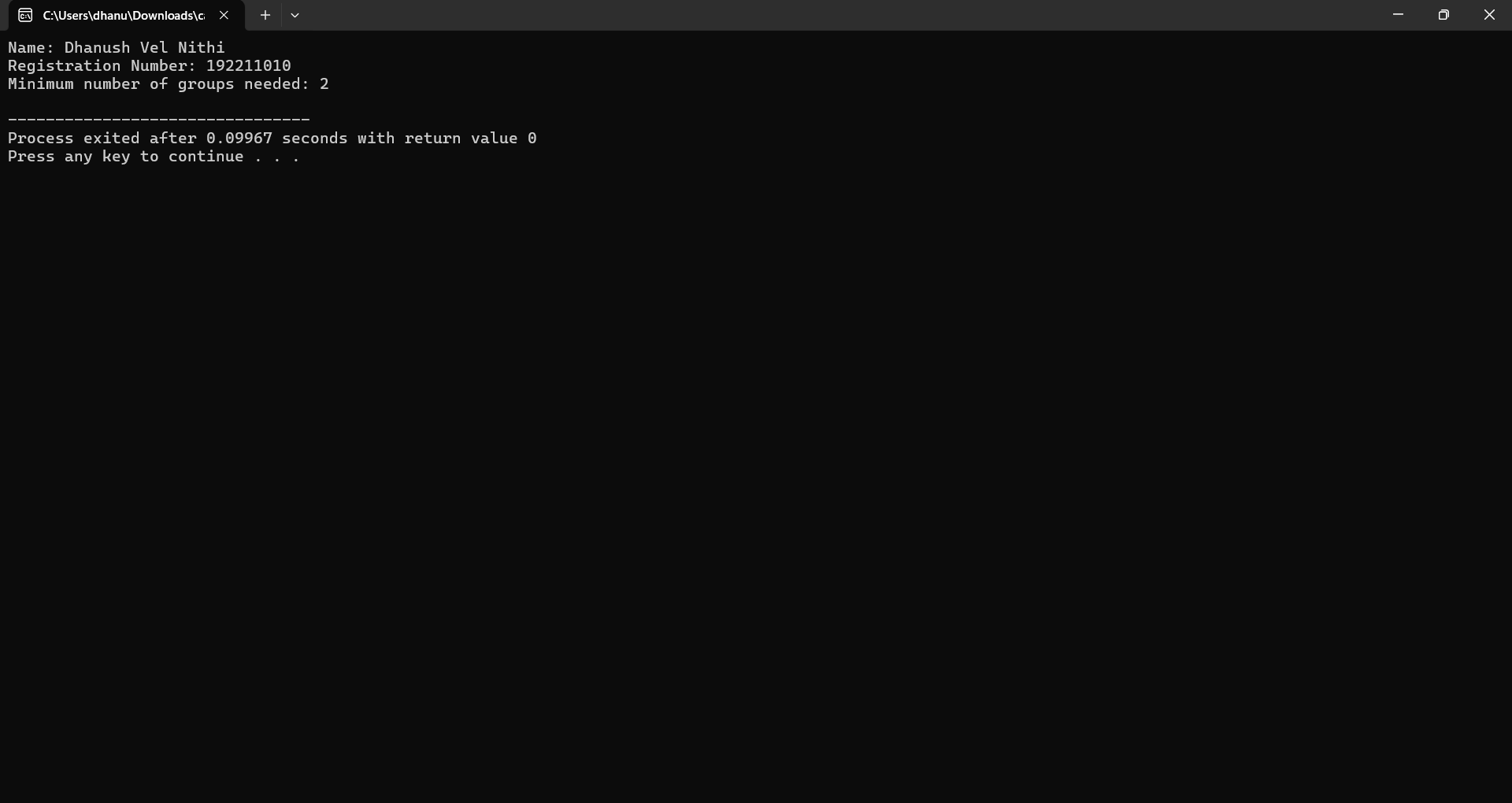
printf("Minimum number of groups needed: %d\n", result);

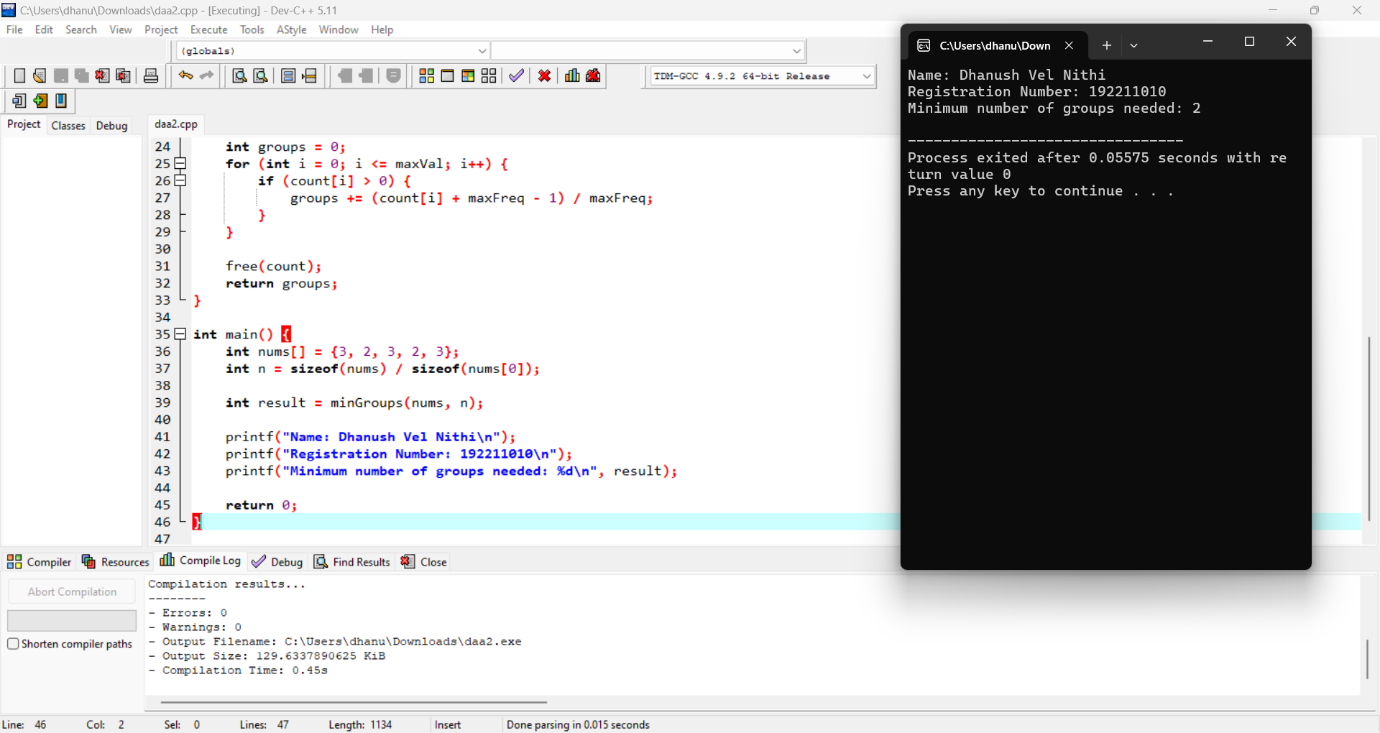
return 0;

}

4

**OUTPUT:**





**COMPLEXITY ANALYSIS:**

**Time Complexity**: The time complexity is O(n), where n is the length of the nums array. This is due to the single pass through the array to compute frequencies. If sorting were required, the complexity would increase to O(n log n), but in this case, we only use counting.

**Space Complexity**: The space complexity is O(1) because the frequency array is of constant size (fixed at 1001 to accommodate possible values of nums[i]), independent of the input size n.

**BEST CASE:**

The best-case scenario occurs when all elements in the array are identical. In this case, only one group is required, and the time complexity remains O(n), as we still need to count the frequencies of the elements.

**WORST CASE:**

The worst case arises when all elements in the array are distinct. In this scenario, the number of groups equals the length of the array (i.e., n), and the program still runs in O(n) time.

**AVERAGE CASE:**

In an average case, the array contains some repeated values but also some distinct ones. The number of groups will vary depending on the distribution of elements, but the time complexity remains O(n) due to the frequency counting process.

**Future Scope:**

**This grouping algorithm can be applied to various domains such as:**

**Task Scheduling: Distributing tasks among workers where tasks of the same type should be grouped together.**

**Load Balancing: Ensuring servers in a distributed system receive similar workloads.**

**Data Clustering: Grouping similar data points in machine learning applications, where balancing cluster sizes is a priority.**

**Conclusion:**

**This project presents a solution to the problem of grouping array indices such that group sizes are balanced and group members have identical values. The C program provides an efficient implementation with linear time complexity. We have analyzed the algorithm's behavior in best, worst, and average case scenarios. With potential applications in various fields like scheduling and clustering, future work could explore optimizing the grouping process further or adapting the solution for dynamic or real-time data streams.**