**NAME: HARSHIT GHOSH**

**ROLLNO: 22052982**

**SUBJECT: DESIGN ANALYSIS AND ALGORITHMS**

# Lab Day 1: Revision of Data Structures

***1.1 Aim of the program:*** *Write a program to find out the second smallest and second largest*

*element stored in an array of n integers.*

*Input: Size of the array is ‘n’ and read ‘n’ number of elements from a disc file.*

*Output: Second smallest, Second largest*

//Find second largest and second smallest element in an array

#include<stdio.h>

#include<limits.h>

int i;

void secondLargest( int arr[] , int n){

    int first,second;

    first = second = INT\_MIN;

    if(n<2){

        printf("Invalid input");

        return;

    }

    for(i=0; i<n; i++){

        if(arr[i] > first){

            second = first;

            first = arr[i];

        }

        else if(arr[i] > second && arr[i]!=first)

            second = arr[i];

    }

    if(second==INT\_MIN)

        printf("There is no second largest element");

    else

        printf("Second Largest Element: %d\n",second);

}

void secondSmallest( int arr[] , int n){

    int first,second;

    first = second = INT\_MAX;

    if(n<2){

        printf("Invalid input");

        return;

    }

    for(i=0; i<n; i++){

        if(arr[i] < first){

            second = first;

            first = arr[i];

        }

        else if(arr[i] < second && arr[i]!=first)

            second = arr[i];

    }

    if(second==INT\_MAX)

        printf("There is no second largest element");

    else

        printf("Second Smallest Element: %d",second);

}

int main(){

    int n;

    printf("Enter size of array: ");

    scanf("%d",&n);

    int arr[n];

    printf("Enter %d elements in array: ",n);

    for(i=0; i<n; i++){

        scanf("%d",&arr[i]);

    }

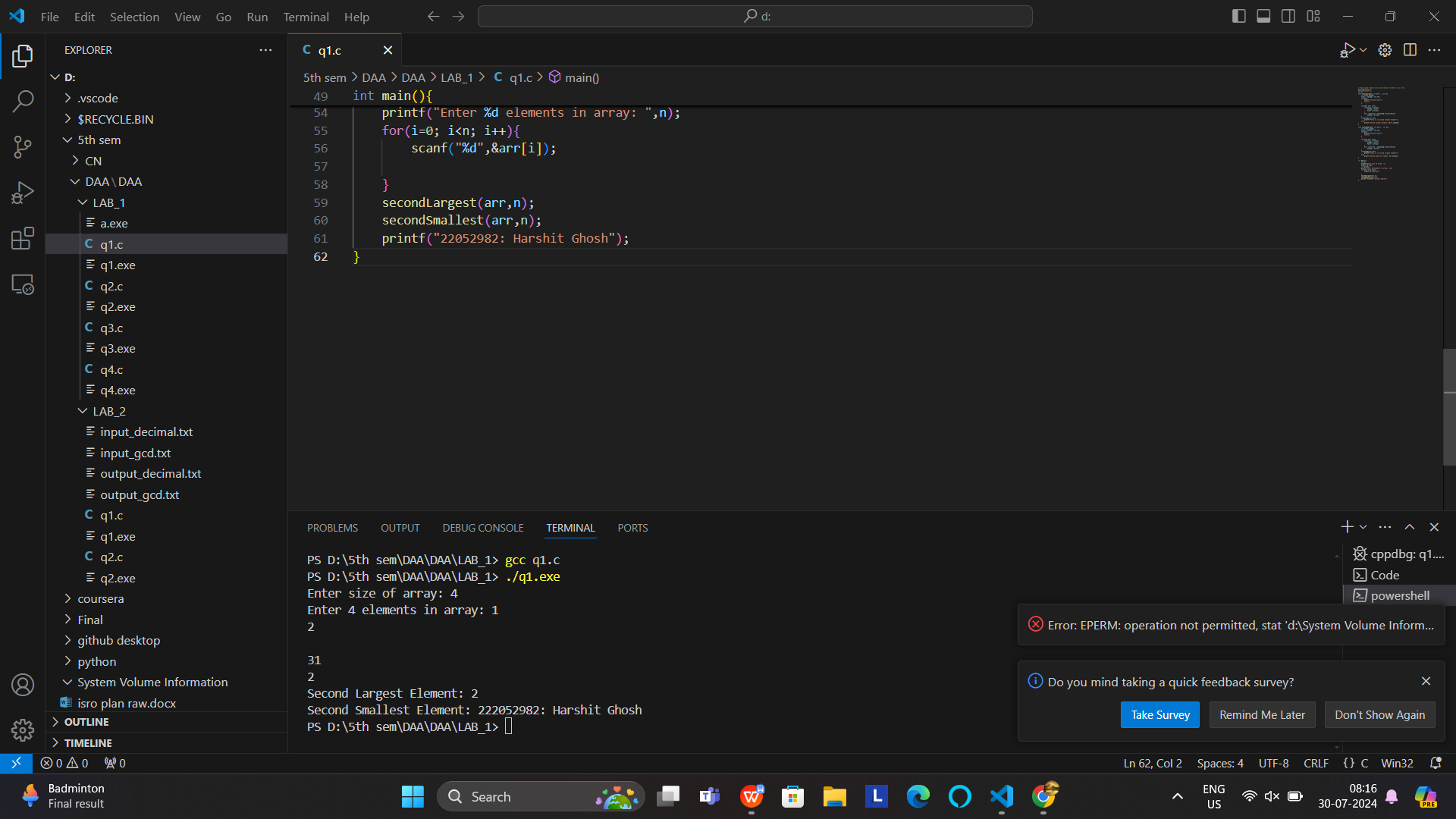
    secondLargest(arr,n);

    secondSmallest(arr,n);

    printf("22052982: Harshit Ghosh");

}

***INPUT/OUTPUT***



**1.2 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].*

//PrefixSum array

#include<stdio.h>

int i;

void prefixSum(int arr[], int n){

    int sum = 0;

    int prefixSum[n];

    for(i=0; i<n; i++){

        sum += arr[i];

        prefixSum[i] = sum;

    }

    printf("Prefix Sum Array: ");

    for(i=0; i<n; i++)

        printf("%d ",prefixSum[i]);

}

int main(){

    int n;

    printf("Enter size of array: ");

    scanf("%d",&n);

    int arr[n];

    printf("Enter %d elements in array: ",n);

    for(i=0; i<n; i++){

        scanf("%d",&arr[i]);

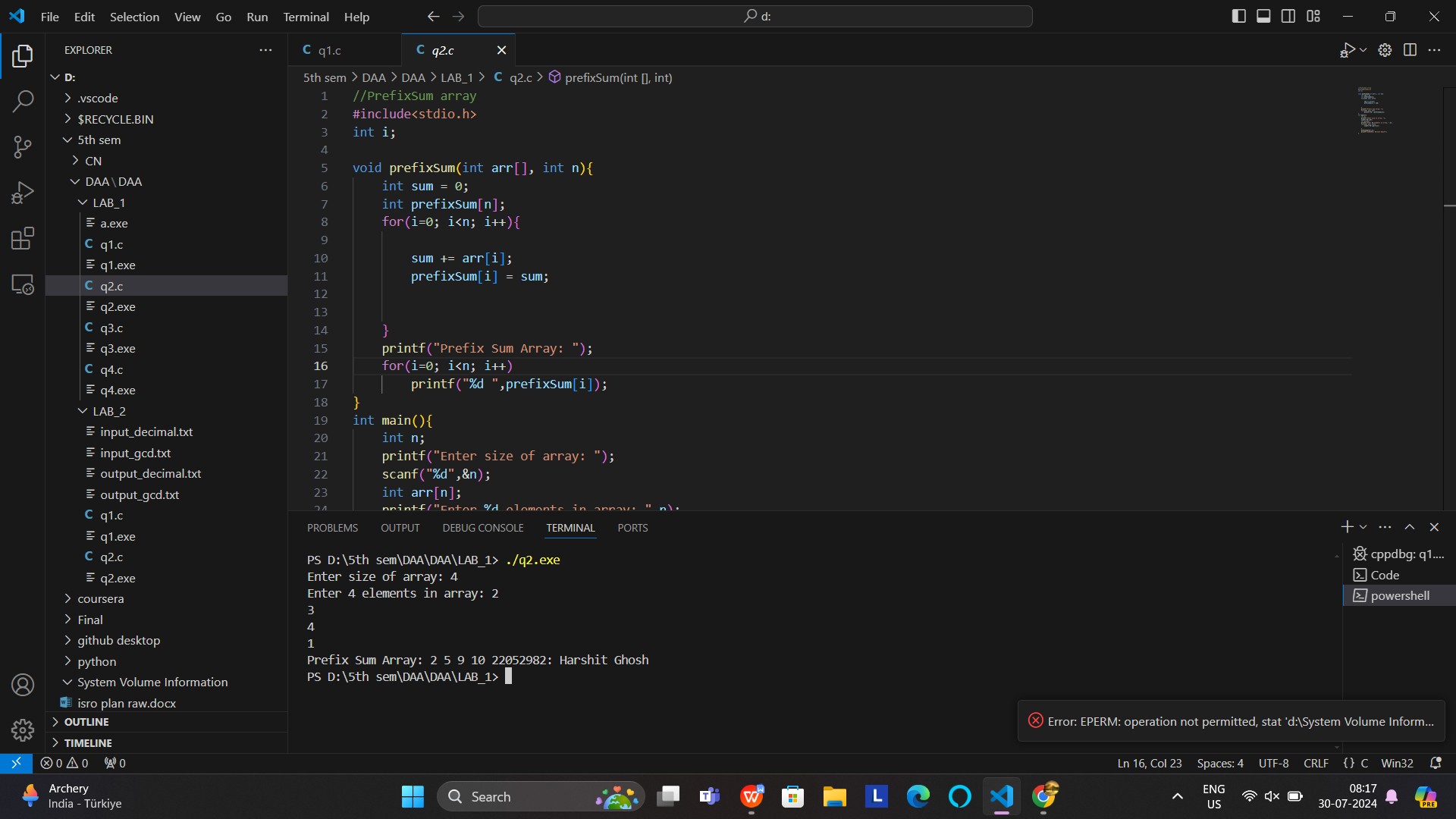
    }

    prefixSum(arr,n);

    printf("22052982: Harshit Ghosh");

}

***INPUT/OUTPUT***



***1.3 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.*

*a) Find out the total number of duplicate elements.*

*b) Find out the most repeating element in the array.*

//Find number of elements that have duplicates and element with max repetition

#include<stdio.h>

#include<stdlib.h>

int i;

#define MAX\_SIZE 1000

int countDuplicates(int arr[], int n, int count[]){

    int duplicates = 0;

    for(i=0; i<n; i++){

        count[arr[i]]++;

        if(count[arr[i]] == 2)

            duplicates++;

    }

    return duplicates;

}

int elementFrequency(int count[] , int size){

    int maxCount = 0 , maxElement = 0;

    for(i=0; i<size; i++){

        if(count[i] > maxCount){

        maxCount = count[i];

        maxElement = i;

        }

    }

    return maxElement;

}

int main(){

    int n;

    int count[MAX\_SIZE] = {0};

    printf("Enter size of array: ");

    scanf("%d",&n);

    int arr[n];

    printf("Enter %d elements in array: ",n);

    for(i=0; i<n; i++){

        scanf("%d",&arr[i]);

    }

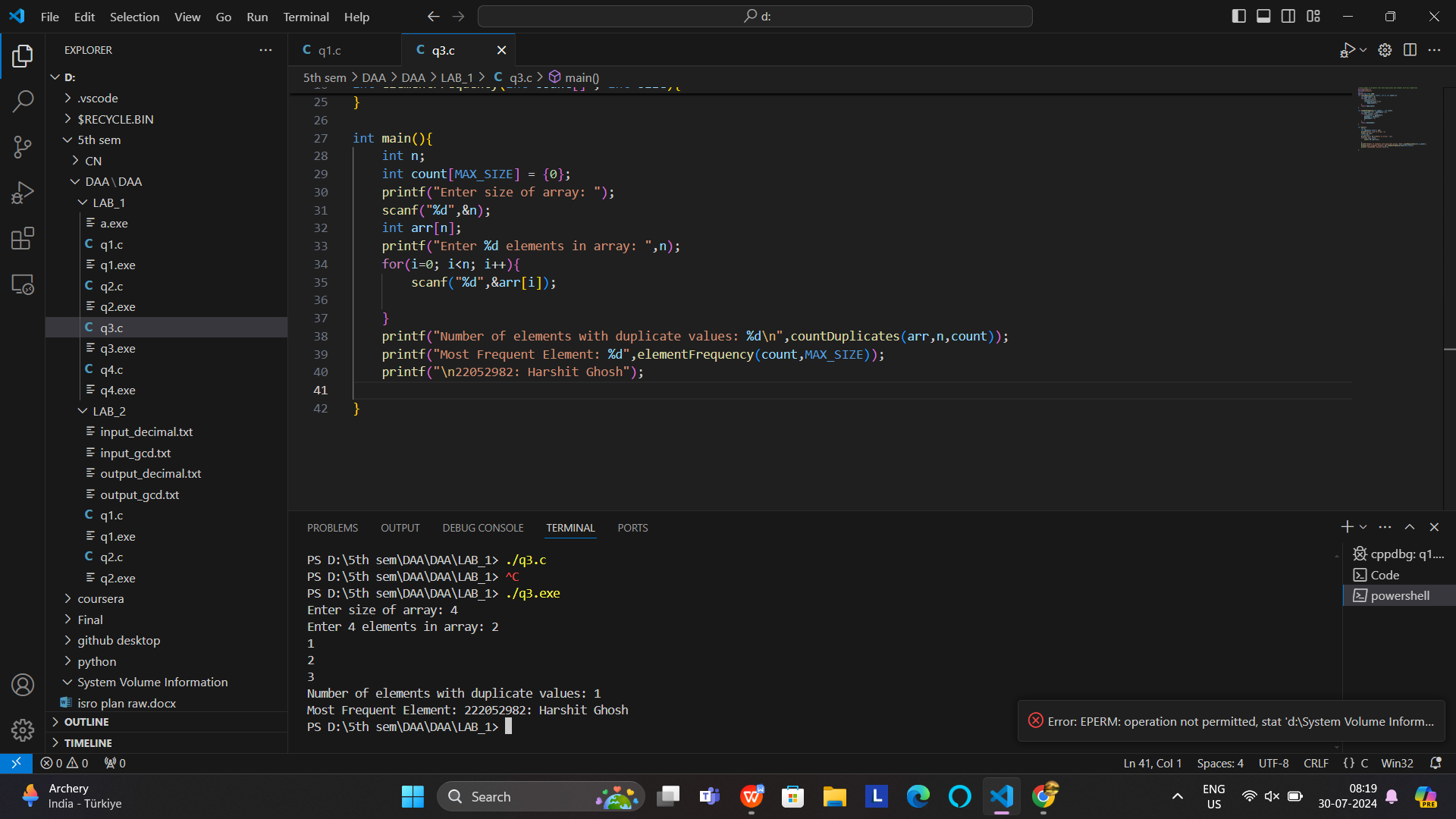
    printf("Number of elements with duplicate values: %d\n",countDuplicates(arr,n,count));

    printf("Most Frequent Element: %d",elementFrequency(count,MAX\_SIZE));

    printf("\n22052982: Harshit Ghosh");

}

***INPUT/OUTPUT***



***1.4 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 &*

*q. Parameter p1 be the starting address of the array and p2 be the number of elements to be*

*rotated.*

#include <stdio.h>

void exchange(int \*p, int \*q) {

    int temp = \*p;

    \*p = \*q;

    \*q = temp;

}

void rotateRight(int \*p1, int p2) {

    for (int i = p2 - 1; i > 0; i--) {

        exchange(&p1[i], &p1[i-1]);

    }

}

void printArray(int \*arr, int size) {

    for (int i = 0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

int main() {

    int size, rotations;

    // User input for array size

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

    scanf("%d", &size);

    int A[size];

    // User input for array elements

    printf("Enter %d elements of the array: ", size);

    for (int i = 0; i < size; i++) {

        scanf("%d", &A[i]);

    }

    // User input for the number of rotations

    printf("Enter the number of rotations: ");

    scanf("%d", &rotations);

    printf("Before rotate: ");

    printArray(A, size);

    // Perform the rotation

    rotateRight(A, rotations);

    printf("After rotate:  ");

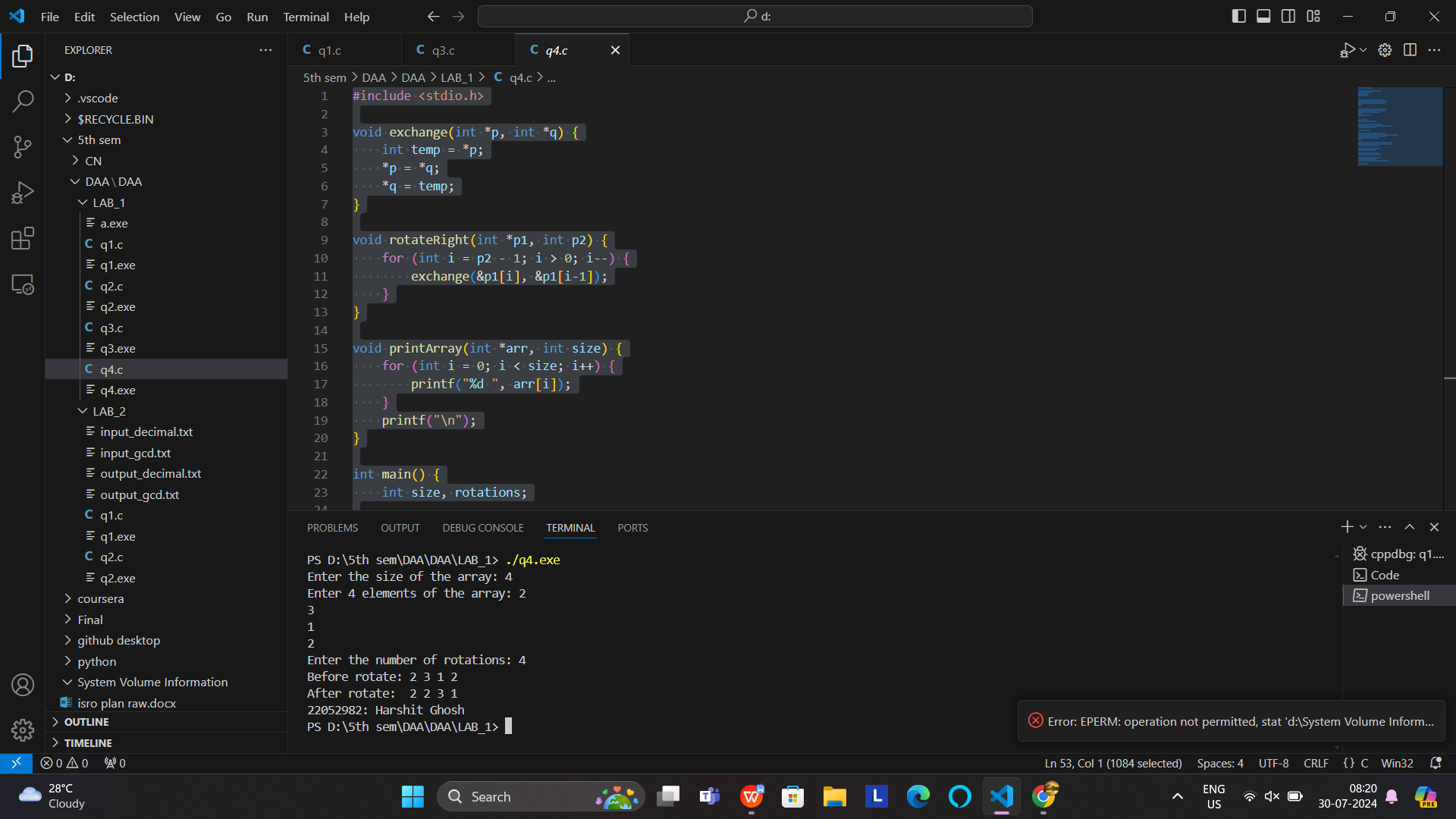
    printArray(A, size);

    printf("22052982: Harshit Ghosh\n");

    return 0;

}

***INPUT/OUTPUT***



# Lab Day 2: Fundamentals of Algorithmic Problem Solving

***2.1 Aim of the program:*** *Write a program in C to convert the first ‘n’ decimal numbers of a disc*

*file to binary using recursion. Store the binary value in a separate disc file.*

#include<stdio.h>

#define LENGTH 100

int i;

long int decimal\_binary(int dec\_num){

    if(dec\_num==0)

        return 0;

    else

        return(dec\_num % 2 + 10 \* decimal\_binary(dec\_num / 2));

}

int main(int argc, char \*argv[]){

    printf("\n22052982: Harshit Ghosh\n");

    char \*inputFile = argv[1];

    int arr[LENGTH];

    int n;

    FILE \*fp = fopen(inputFile,"r");

    if(fp==NULL){

        printf("Cannot open file");

        return 0;

    }

    printf("Enter array length: ");

    scanf("%d",&n);

    for(i=0; i<n; i++){

        fscanf(fp,"%d",&arr[i]);

    }

    fclose(fp);

    long int bin\_arr[LENGTH];

    for(i=0; i<n; i++){

        bin\_arr[i] = decimal\_binary(arr[i]);

    }

    char \*outputFile = argv[2];

    FILE \*fp1 = fopen(outputFile,"w");

    if(fp1==NULL){

        printf("Cannot open file to write");

        return 0;

    }

    for(i=0; i<n; i++){

        fprintf(fp1,"The binary equivalent of %d is %ld\n",arr[i],bin\_arr[i]);

    }

    fclose(fp1);

    FILE \*fp2 = fopen(outputFile,"r");

    char c = fgetc(fp2);

    while(c!=EOF){

        printf("%c",c);

        c=fgetc(fp2);

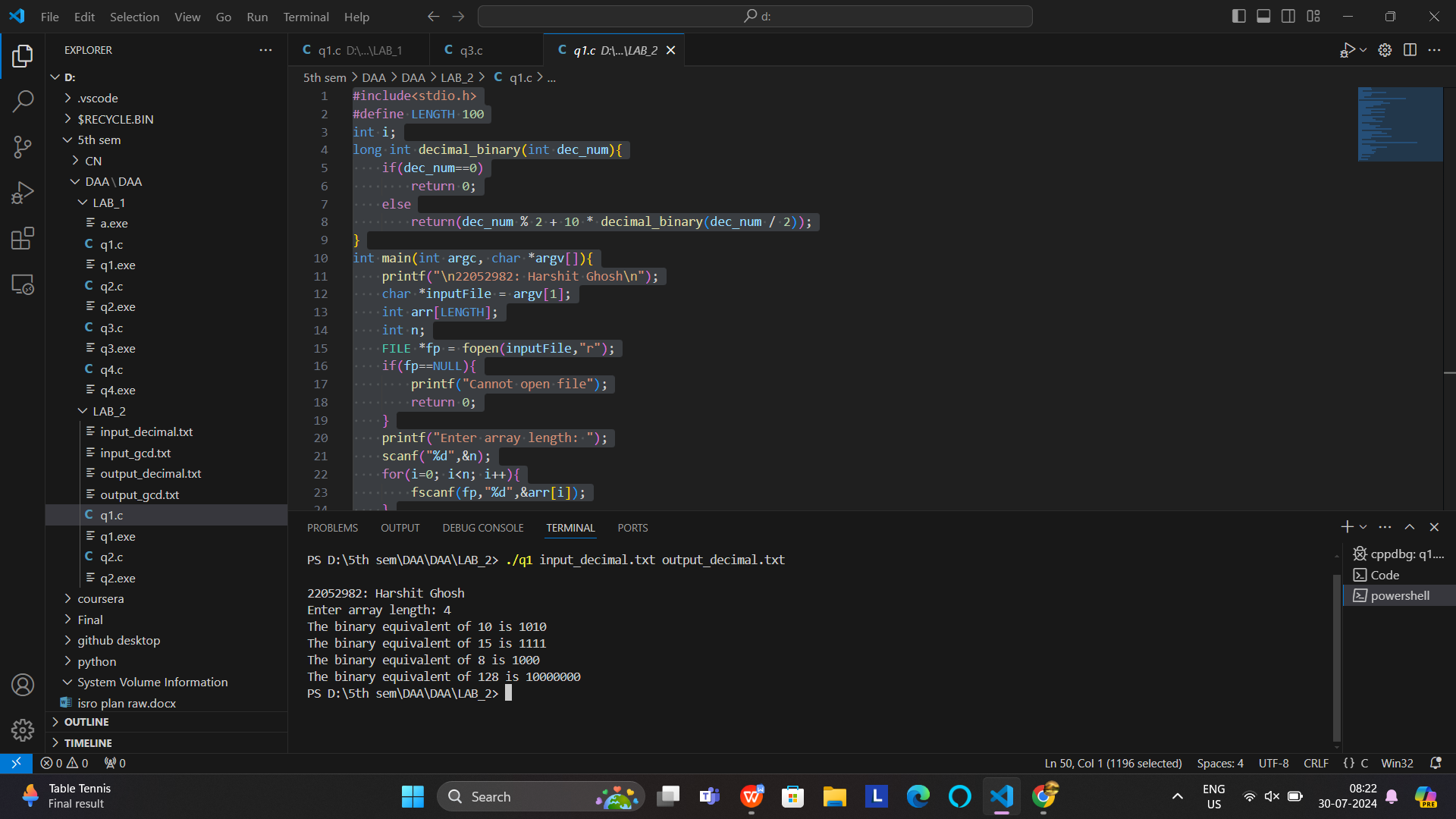
    }

    fclose(fp2);

    return 0;

}

***INPUT/OUTPUT***



***2.3 Aim of the program:*** *Write a program in C to find GCD of two numbers using recursion.*

*Read all pair of numbers from a file and store the result in a separate file.*

#include<stdio.h>

#include<stdlib.h>

int gcd(int a, int b){

    if(b == 0)

        return a;

    return gcd(b, a % b);

}

int main(int argc, char \*argv[]){

    FILE \*infile = fopen(argv[1], "r");

    FILE \*outfile = fopen(argv[2], "w");

    if(infile == NULL || outfile == NULL){

        printf("Cannot Open File");

        return 0;

    }

    int n1, n2;

    while(fscanf(infile, "%d %d", &n1, &n2) == 2){

        int res = gcd(n1, n2);

        fprintf(outfile, "The GCD of %d and %d is %d\n", n1, n2, res);

    }

    fclose(infile);

    fclose(outfile);

    outfile = fopen(argv[2], "r");

    char c = getc(outfile);

    while(c != EOF){

        printf("%c", c);

        c = getc(outfile);

    }

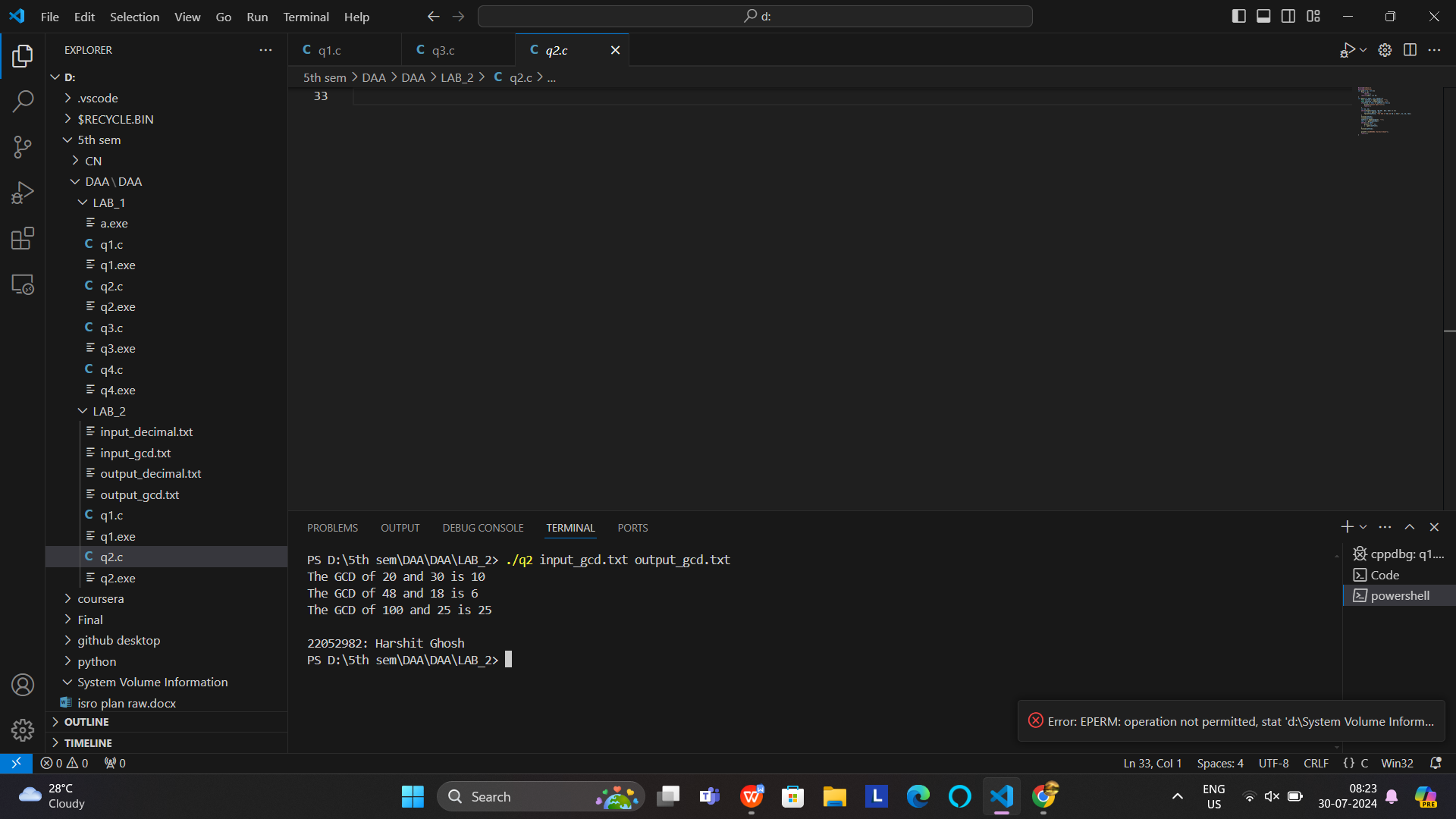
    fclose(outfile);

    printf("\n22052982: Harshit Ghosh");

    return 0;

}

***INPUT/OUTPUT***



# Lab Day 3: Divide and Conquer Method

**3.1 Aim of the program:** *Write a menu driven program to sort list of array elements using Merge*

*Sort technique and calculate the execution time only to sort the elements. Count the number of*

*comparisons.*

*Note#*

*● To calculate execution time, assume that single program is under execution in the CPU.*

*● Number of elements in each input file should vary from 300 to 500 entries.*

*● For ascending order: Read data from a file “inAsce.dat” having content 10 20 30 40.....,*

*Store the result in “outMergeAsce.dat”.*

*● For descending order: Read data from a file “inDesc.dat” having content 90 80 70 60....,*

*Store the result in “outMergeDesc.dat”.*

*● For random data: Read data from a file “inRand.dat” having content 55 66 33 11 44 ...,*

*Store the result in “outMergeRand.dat”*

*Sample Input from file:*

*MAIN MENU (MERGE SORT)*

*1. Ascending Data*

*2. Descending Data*

*3. Random Data*

*4. ERROR (EXIT)*

*Output:*

*Enter option: 1*

*Before Sorting: Content of the input file*

*After Sorting: Content of the output file*

*Number of Comparisons: Actual*

*Execution Time: lapse time in nanosecond*

#include <stdio.h>

#include <stdlib.h>

#ifdef \_WIN32

#include <windows.h>

#else

#include <time.h>

#endif

// Function prototypes

void mergesort(int a[], int low, int high, int \*comparison);

void merge(int a[], int low, int mid, int high, int \*comparison);

void process\_file(const char \*input\_filename, const char \*output\_filename);

#ifdef \_WIN32

// Function to get time in nanoseconds on Windows

long long get\_time\_in\_nanoseconds() {

    LARGE\_INTEGER frequency, start, end;

    QueryPerformanceFrequency(&frequency);

    QueryPerformanceCounter(&start);

    // The time measurement will be done in the function where it's used

    return start.QuadPart;

}

long long get\_elapsed\_time(long long start) {

    LARGE\_INTEGER end;

    QueryPerformanceCounter(&end);

    return (end.QuadPart - start) \* 1000000000LL / get\_time\_in\_nanoseconds();

}

#else

// Function to get time in nanoseconds on Linux

long long get\_time\_in\_nanoseconds() {

    struct timespec ts;

    clock\_gettime(CLOCK\_MONOTONIC, &ts);

    return ts.tv\_sec \* 1e9 + ts.tv\_nsec;

}

#endif

int main() {

    int choice;

    while (1) {

        printf("22052982: HARSHIT GHOSH");

        printf("MAIN MENU:\n1. Ascending Data\n2. Descending Data\n3. Random Data\n4. EXIT\nEnter option: ");

        if (scanf("%d", &choice) != 1) {

            fprintf(stderr, "Invalid input. Please enter an integer.\n");

            while (getchar() != '\n'); // Clear invalid input

            continue;

        }

        switch (choice) {

            case 1: process\_file("inAsce.txt", "outMergeAsce.txt"); break;

            case 2: process\_file("inDesc.txt", "outMergeDesc.txt"); break;

            case 3: process\_file("inRand.txt", "outMergeRand.txt"); break;

            case 4: printf("Exiting...\n"); exit(0);

            default: printf("Invalid option. Please try again.\n");

        }

    }

    return 0;

}

void merge(int a[], int low, int mid, int high, int \*comparison) {

    int n1 = mid - low + 1, n2 = high - mid;

    int \*L = (int \*)malloc(n1 \* sizeof(int));

    int \*R = (int \*)malloc(n2 \* sizeof(int));

    if (!L || !R) { fprintf(stderr, "Memory allocation failed\n"); exit(EXIT\_FAILURE); }

    for (int i = 0; i < n1; i++) L[i] = a[low + i];

    for (int i = 0; i < n2; i++) R[i] = a[mid + i + 1];

    int i = 0, j = 0, k = low;

    while (i < n1 && j < n2) {

        a[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

        (\*comparison)++;

    }

    while (i < n1) a[k++] = L[i++];

    while (j < n2) a[k++] = R[j++];

    free(L);

    free(R);

}

void mergesort(int a[], int low, int high, int \*comparison) {

    if (low < high) {

        int mid = (low + high) / 2;

        mergesort(a, low, mid, comparison);

        mergesort(a, mid + 1, high, comparison);

        merge(a, low, mid, high, comparison);

    }

}

void process\_file(const char \*input\_filename, const char \*output\_filename) {

    FILE \*input = fopen(input\_filename, "r");

    FILE \*output = fopen(output\_filename, "w");

    if (!input || !output) { perror("Error opening file"); return; }

    int n;

    if (fscanf(input, "%d", &n) != 1) { fprintf(stderr, "Error reading array size.\n"); fclose(input); fclose(output); return; }

    int \*arr = (int \*)malloc(n \* sizeof(int));

    if (!arr) { fprintf(stderr, "Memory allocation failed\n"); fclose(input); fclose(output); return; }

    for (int i = 0; i < n; i++) {

        if (fscanf(input, "%d", &arr[i]) != 1) { fprintf(stderr, "Error reading array element.\n"); free(arr); fclose(input); fclose(output); return; }

    }

    fprintf(output, "Before sorting:\n");

    for (int i = 0; i < n; i++) fprintf(output, "%d ", arr[i]);

    fprintf(output, "\n");

    int comparison = 0;

    long long start\_time = get\_time\_in\_nanoseconds();

    mergesort(arr, 0, n - 1, &comparison);

    long long end\_time = get\_time\_in\_nanoseconds();

    long long time\_taken = end\_time - start\_time;

    fprintf(output, "After sorting:\n");

    for (int i = 0; i < n; i++) fprintf(output, "%d ", arr[i]);

    fprintf(output, "\nNumber of Comparisons: %d\nExecution Time: %lld nanoseconds\n", comparison, time\_taken);

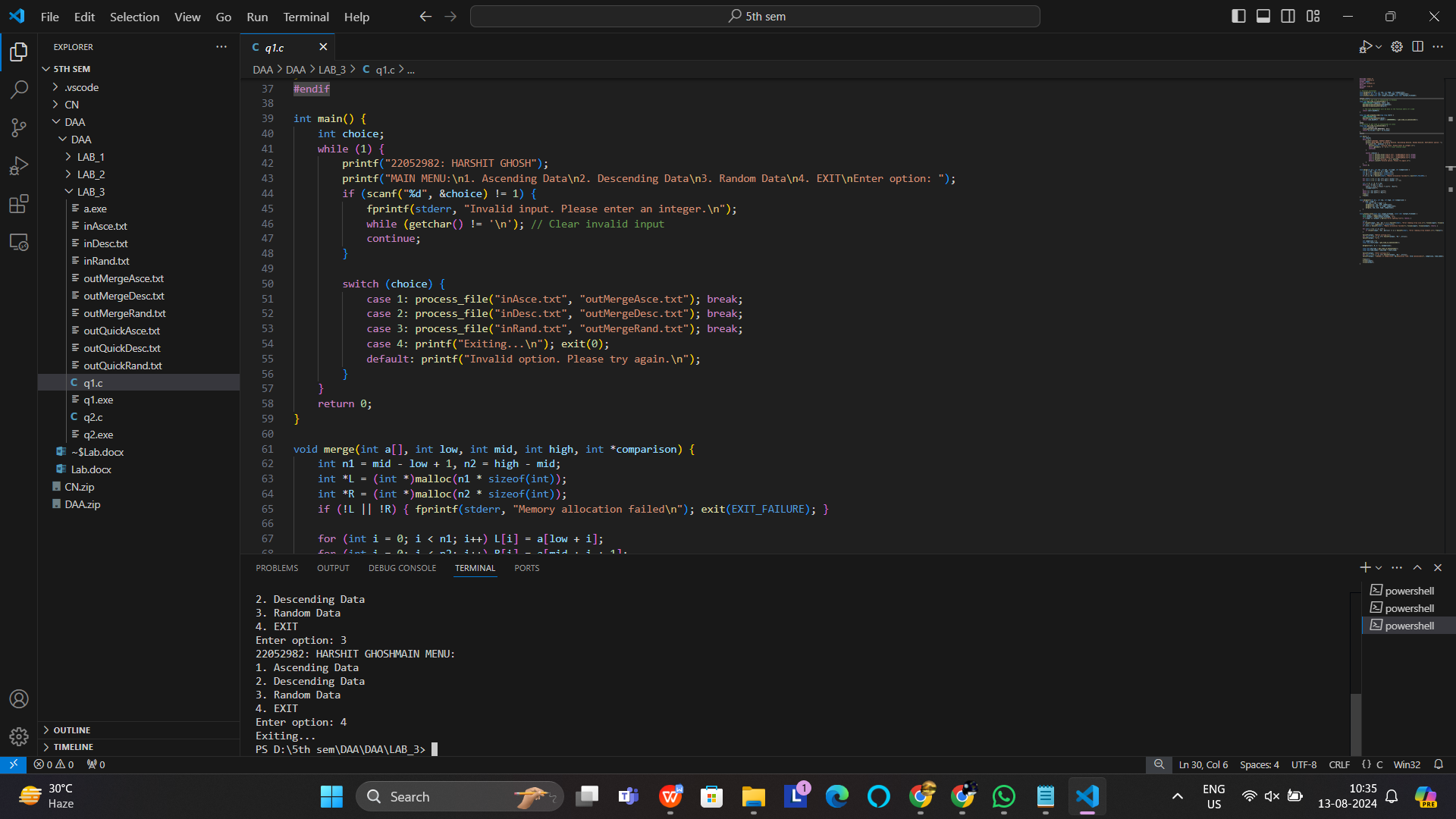
    free(arr);

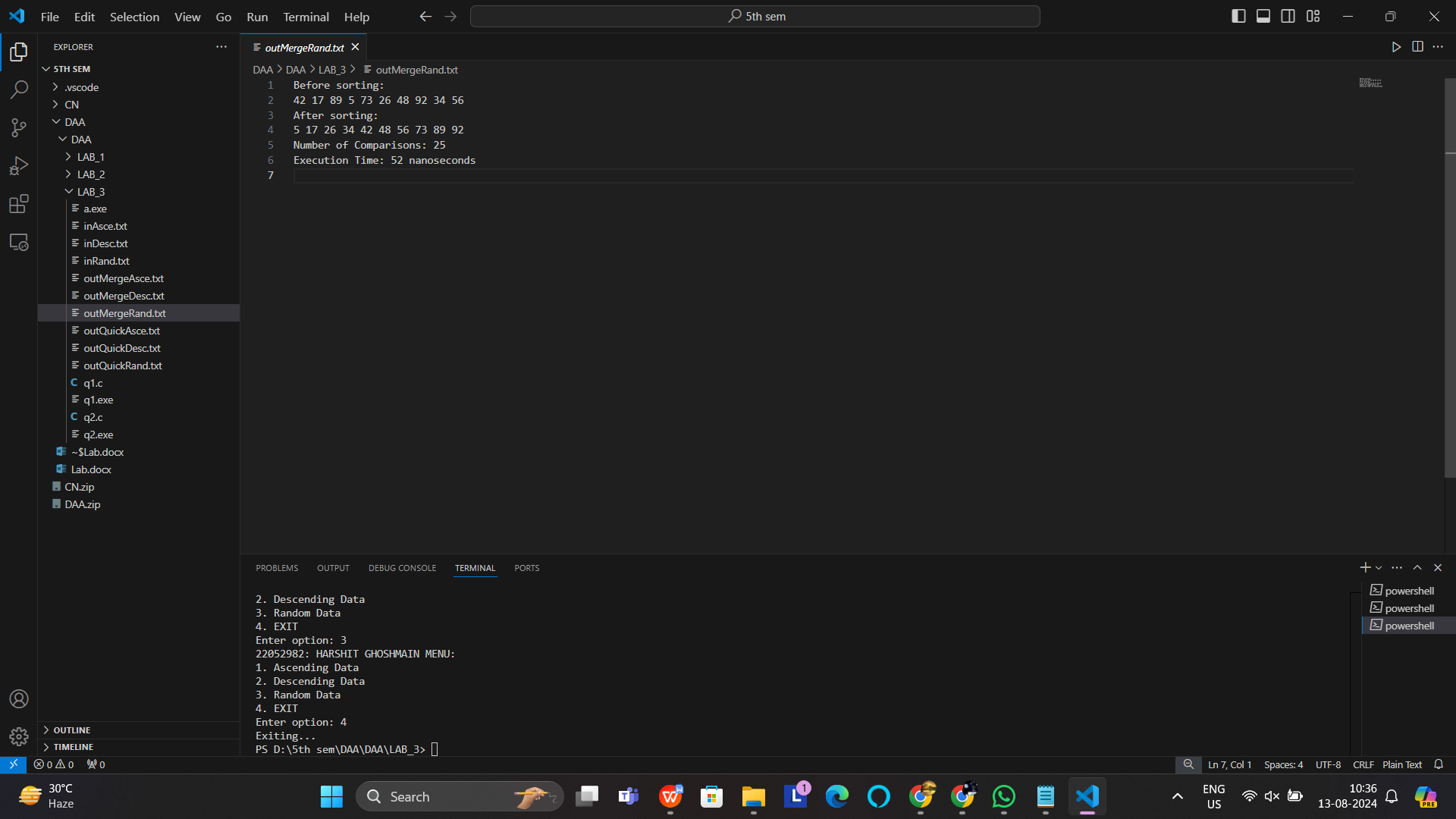
    fclose(input);

    fclose(output);

}

***INPUT/OUTPUT***





**3.2 Aim of the program:** *Write a menu driven program to sort a list of elements in ascending*

*order using Quick Sort technique. Each choice for the input data has its own disc file. A separate*

*output file can be used for sorted elements. After sorting display the content of the output file*

*along with number of comparisons. Based on the partitioning position for each recursive call,*

*conclude the input scenario is either best-case partitioning or worst-case partitioning.*

*Note#*

*● The worst-case behavior for quicksort occurs when the partitioning routine produces one*

*subproblem with n-1 elements and one with 0 elements. The best-case behaviour*

*occurred in most even possible split, PARTITION produces two subproblems, each of*

*size no more than n/2.*

*● Number of elements in each input file should vary from 300 to 500 entries.*

*● For ascending order: Read data from a file “inAsce.dat” having content 10 20 30 40.....,*

*Store the result in “outQuickAsce.dat”.*

*● For descending order: Read data from a file “inDesc.dat” having content 90 80 70 60....,*

*Store the result in “outQuickDesc.dat”.*

*● For random data: Read data from a file “inRand.dat” having content 55 66 33 11 44 ...,*

*Store the result in “outQuickRand.dat”*

*Sample Input from file:*

*MAIN MENU (QUICK SORT)*

*1. Ascending Data*

*2. Descending Data*

*3. Random Data*

*4. ERROR (EXIT)*

*Output:*

*Enter option: 1*

*Before Sorting: Content of the input file*

*After Sorting: Content of the output file*

*Number of Comparisons: Actual*

*Scenario: Best or Worst-case*

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#ifdef \_WIN32

#include <windows.h>

#else

#include <sys/time.h>

#endif

void swap(int \*x, int \*y);

int partition(int arr[], int low, int high, int \*comparison);

void quicksort(int arr[], int low, int high, int \*comparison);

void driver(FILE \*input, int arr[], FILE \*output);

void print\_array(FILE \*output, int arr[], int n);

#ifdef \_WIN32

double get\_time\_in\_seconds() {

    LARGE\_INTEGER frequency;

    LARGE\_INTEGER start;

    QueryPerformanceFrequency(&frequency);

    QueryPerformanceCounter(&start);

    return (double)start.QuadPart / frequency.QuadPart;

}

#else

double get\_time\_in\_seconds() {

    struct timeval time;

    gettimeofday(&time, NULL);

    return (double)time.tv\_sec + (double)time.tv\_usec \* .000001;

}

#endif

int main() {

    FILE \*inputfile, \*outputfile;

    int choice = 0;

    int arr[500];

    do {

        printf("22052982: HARSHIT GHOSH");

        printf("MAIN MENU:\n");

        printf("1. Ascending Data\n2. Descending Data\n3. Random Data\n4. EXIT\n");

        printf("Enter option: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                inputfile = fopen("inAsce.txt", "r");

                outputfile = fopen("outQuickAsce.txt", "w");

                if (inputfile && outputfile) {

                    driver(inputfile, arr, outputfile);

                    fclose(inputfile);

                    fclose(outputfile);

                } else {

                    printf("Error opening file.\n");

                }

                break;

            case 2:

                inputfile = fopen("inDesc.txt", "r");

                outputfile = fopen("outQuickDesc.txt", "w");

                if (inputfile && outputfile) {

                    driver(inputfile, arr, outputfile);

                    fclose(inputfile);

                    fclose(outputfile);

                } else {

                    printf("Error opening file.\n");

                }

                break;

            case 3:

                inputfile = fopen("inRand.txt", "r");

                outputfile = fopen("outQuickRand.txt", "w");

                if (inputfile && outputfile) {

                    driver(inputfile, arr, outputfile);

                    fclose(inputfile);

                    fclose(outputfile);

                } else {

                    printf("Error opening file.\n");

                }

                break;

            case 4:

                printf("Exiting...\n");

                exit(0);

                break;

            default:

                printf("Invalid option. Please try again.\n");

                break;

        }

    } while (choice != 4);

    return 0;

}

void swap(int \*x, int \*y){

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

int partition(int arr[], int low, int high, int \*comparison){

    int i = low, j = high;

    int pivot = arr[low]; // starting element as pivot element

    while (i < j) {

        while (arr[i] <= pivot && i <= high - 1) {

            i++;

            (\*comparison)++;

        }

        while (arr[j] > pivot && j >= low + 1) {

            j--;

            (\*comparison)++;

        }

        if (i < j) {

            swap(&arr[i], &arr[j]);

            (\*comparison)++;

        }

    }

    swap(&arr[low], &arr[j]);

    (\*comparison)++;

    return j;

}

void quicksort(int arr[], int low, int high, int \*comparison){

    if (low < high) {

        int partitionIndex = partition(arr, low, high, comparison);

        quicksort(arr, low, partitionIndex - 1, comparison);

        quicksort(arr, partitionIndex + 1, high, comparison);

    }

}

void print\_array(FILE \*output, int arr[], int n) {

    for (int i = 0; i < n; i++) {

        fprintf(output, "%d ", arr[i]);

    }

    fprintf(output, "\n");

}

void driver(FILE \*input, int arr[], FILE \*output) {

    int n;

    int comparison = 0; // Initialize comparison counter locally

    // Read size of array

    fscanf(input, "%d", &n);

    // Read array

    for (int i = 0; i < n; i++) {

        fscanf(input, "%d", &arr[i]);

    }

    // Printing array before sorting

    fprintf(output, "Before sorting:\n");

    print\_array(output, arr, n);

    // Sorting and calculating execution time

    double start = get\_time\_in\_seconds();

    quicksort(arr, 0, n - 1, &comparison);

    double end = get\_time\_in\_seconds();

    long time\_taken = (long)((end - start) \* 1e9);

    // Printing array after sorting

    fprintf(output, "After sorting:\n");

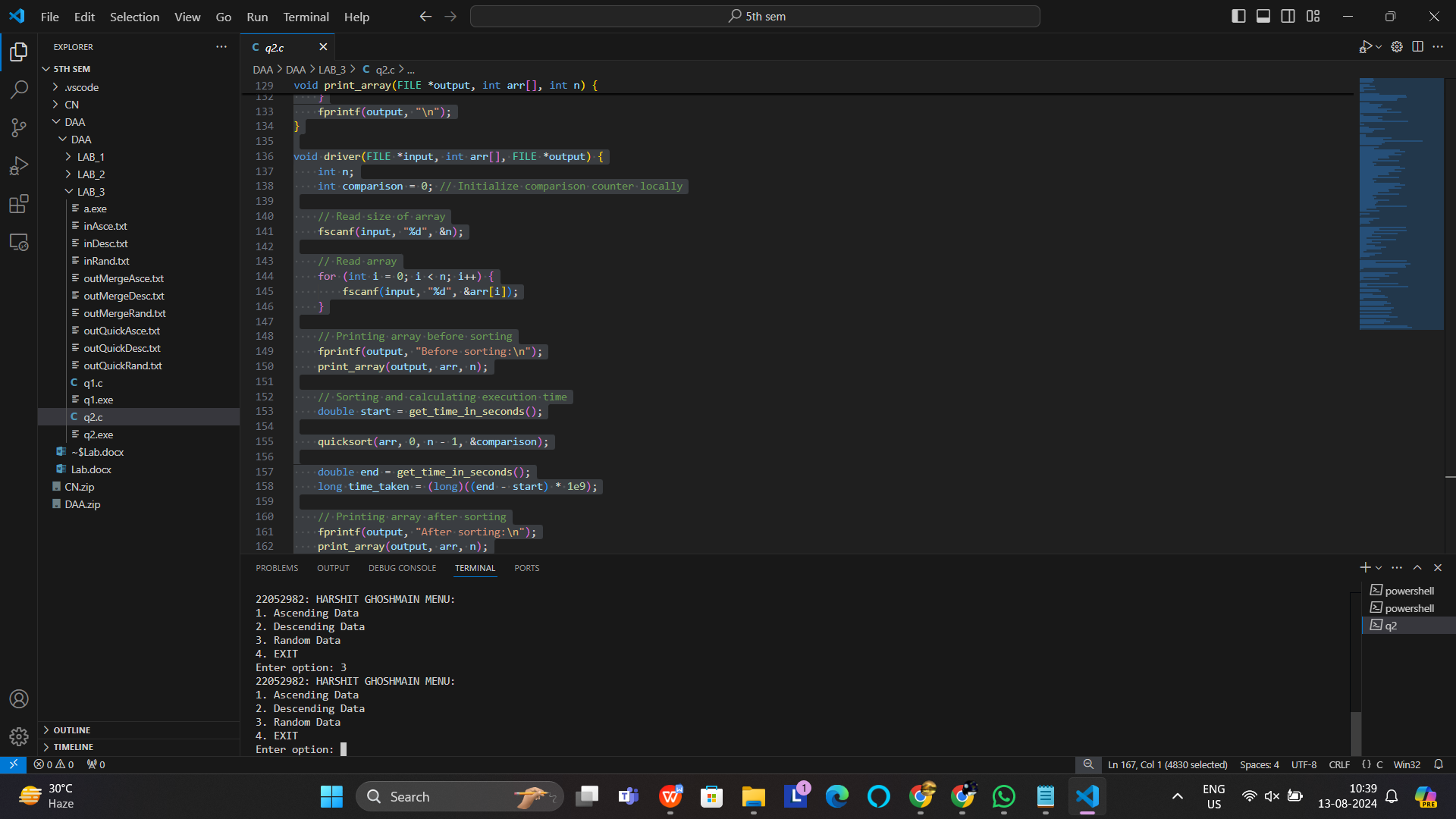
    print\_array(output, arr, n);

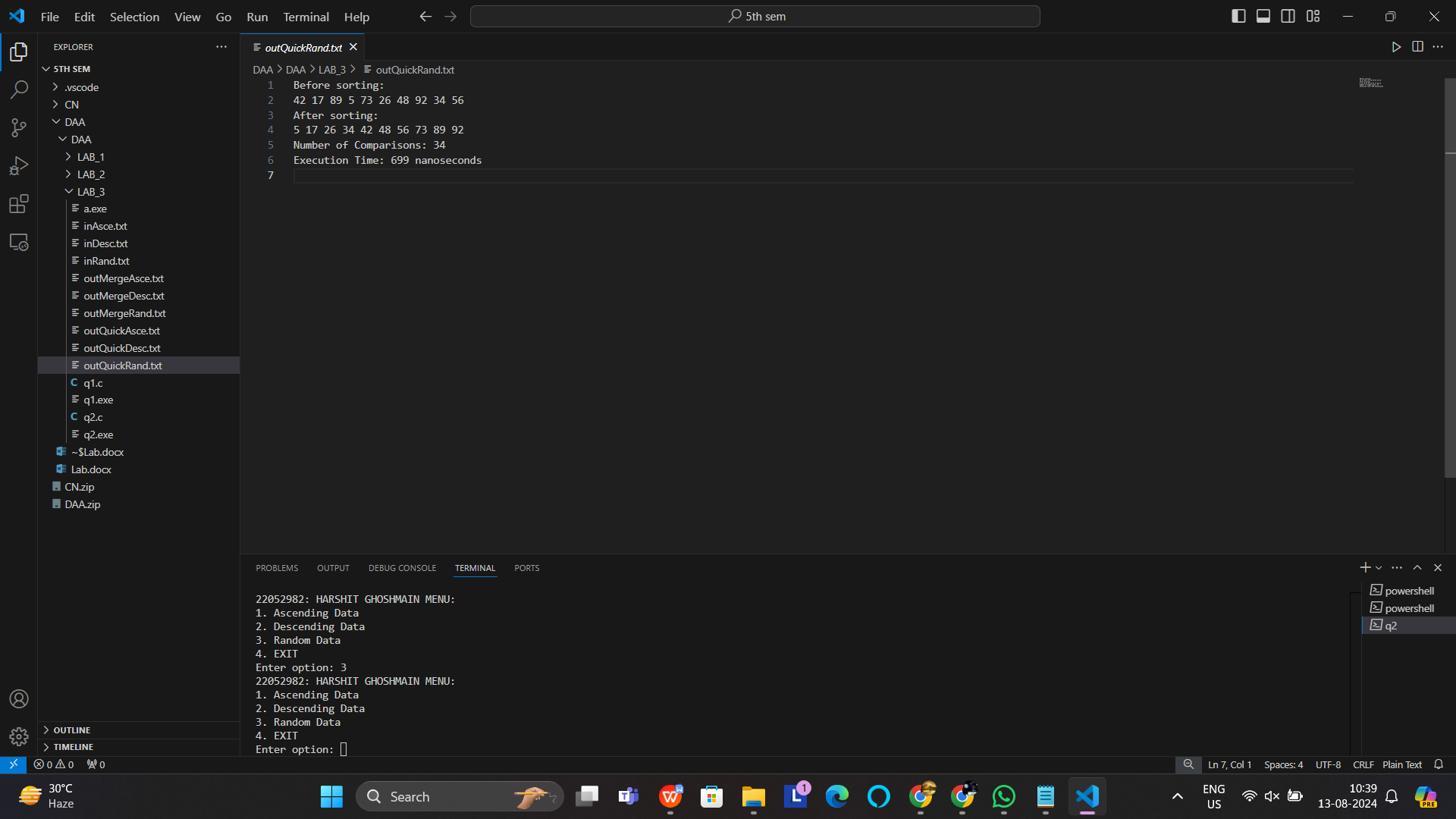
    fprintf(output, "Number of Comparisons: %d\n", comparison);

    fprintf(output, "Execution Time: %ld nanoseconds\n", time\_taken);

}

***INPUT/OUTPUT***





**Lab Day 4: Heap**

4.1 Aim of the program:

Define a struct person as follows:

struct person

{

int id;

char \*name;

int age;

int height;

int weight;

};

Write a menu driven program to read the data of ‘n’ students from a file and store them in a

dynamically allocated array of struct person. Implement the min-heap or max-heap and its

operations based on the menu options.

Sample Input/Output:

MAIN MENU (HEAP)

1. Read Data

2. Create a Min-heap based on the age

3. Create a Max-heap based on the weight

4. Display weight of the youngest person

5. Insert a new person into the Min-heap

6. Delete the oldest person

7. Exit

#*include* <stdio.h>

#*include* <stdlib.h>

#*include* <string.h>

#*define* *MAX* 100

typedef struct {

int id;

char \*name;

int age;

int height;

int weight;

} Person;

Person \*persons;

int size = 0;

void *swap*(Person \*a, Person \*b) {

Person temp = \*a;

\*a = \*b;

\*b = temp;

}

int *leftChild*(int i) {

*return* 2 \* i + 1;

}

int *rightChild*(int i) {

*return* 2 \* i + 2;

}

void *minHeapify*(int n, int i) {

int smallest = i;

*if* (*leftChild*(i) < n && persons[*leftChild*(i)].age < persons[smallest].age)

smallest = *leftChild*(i);

*if* (*rightChild*(i) < n && persons[*rightChild*(i)].age < persons[smallest].age)

smallest = *rightChild*(i);

*if* (smallest != i) {

*swap*(&persons[i], &persons[smallest]);

*minHeapify*(n, smallest);

}

}

void *buildMinHeap*() {

*for* (int i = size / 2 - 1; i >= 0; i--)

*minHeapify*(size, i);

}

void *maxHeapify*(int n, int i) {

int largest = i;

*if* (*leftChild*(i) < n && persons[*leftChild*(i)].weight > persons[largest].weight)

largest = *leftChild*(i);

*if* (*rightChild*(i) < n && persons[*rightChild*(i)].weight > persons[largest].weight)

largest = *rightChild*(i);

*if* (largest != i) {

*swap*(&persons[i], &persons[largest]);

*maxHeapify*(n, largest);

}

}

void *buildMaxHeap*() {

*for* (int i = size / 2 - 1; i >= 0; i--)

*maxHeapify*(size, i);

}

void *readData*(*const* char \*filename) {

FILE \*file = *fopen*(filename, "r");

*if* (file == *NULL*) {

*printf*("Error opening file!\n");

*return*;

}

*fscanf*(file, "%d", &size);

persons = (Person \*)*malloc*(size \* sizeof(Person));

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

persons[i].name = (char \*)*malloc*(50 \* sizeof(char));

*fscanf*(file, "%d %s %d %d %d", &persons[i].id, persons[i].name, &persons[i].age, &persons[i].height, &persons[i].weight);

}

*fclose*(file);

*printf*("Data read from file:\n");

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

*printf*("ID: %d, Name: %s, Age: %d, Height: %d, Weight: %d\n",

persons[i].id, persons[i].name, persons[i].age, persons[i].height, persons[i].weight);

}

}

void *writeToFile*(*const* char \*filename) {

FILE \*file = *fopen*(filename, "w");

*if* (file == *NULL*) {

*printf*("Error opening file for writing!\n");

*return*;

}

*fprintf*(file, "ID Name Age Height Weight\n");

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

*fprintf*(file, "%d %s %d %d %d\n",

persons[i].id, persons[i].name, persons[i].age, persons[i].height, persons[i].weight);

}

*fclose*(file);

*printf*("Heap written to file: %s\n", filename);

}

void *displayWeightOfYoungest*() {

*buildMinHeap*();

*printf*("Weight of the youngest person: %d\n", persons[0].weight);

}

void *insertPersonToMinHeap*() {

*if* (size >= *MAX*) {

*printf*("Heap overflow\n");

*return*;

}

Person newPerson;

newPerson.name = (char \*)*malloc*(50 \* sizeof(char));

*printf*("Enter ID, Name, Age, Height, Weight for the new person: ");

*scanf*("%d %s %d %d %d", &newPerson.id, newPerson.name, &newPerson.age, &newPerson.height, &newPerson.weight);

persons = (Person \*)*realloc*(persons, (size + 1) \* sizeof(Person));

persons[size] = newPerson;

size++;

*buildMinHeap*();

*printf*("New person inserted into the Min-Heap.\n");

}

void *deleteOldestPerson*() {

*if* (size <= 0) {

*printf*("Heap underflow\n");

*return*;

}

*buildMinHeap*();

*printf*("Deleted oldest person: ID: %d, Name: %s, Age: %d, Height: %d, Weight: %d\n",

persons[size - 1].id, persons[size - 1].name, persons[size - 1].age, persons[size - 1].height, persons[size - 1].weight);

size--;

persons = (Person \*)*realloc*(persons, size \* sizeof(Person));

*buildMinHeap*();

}

int *main*() {

int choice;

*do* {

*printf*("\n 22052982: Harshit Ghosh\n");

*printf*("\nMAIN MENU (HEAP)\n");

*printf*("1. Read Data\n");

*printf*("2. Create a Min-heap based on the age\n");

*printf*("3. Create a Max-heap based on the weight\n");

*printf*("4. Display weight of the youngest person\n");

*printf*("5. Insert a new person into the Min-heap\n");

*printf*("6. Delete the oldest person\n");

*printf*("7. Exit\n");

*printf*("Enter your choice: ");

*scanf*("%d", &choice);

*switch* (choice) {

*case* 1:

*readData*("data.txt");

*writeToFile*("data\_output.txt");

*break*;

*case* 2:

*buildMinHeap*();

*writeToFile*("min\_heap\_output.txt");

*break*;

*case* 3:

*buildMaxHeap*();

*writeToFile*("max\_heap\_output.txt");

*break*;

*case* 4:

*displayWeightOfYoungest*();

*break*;

*case* 5:

*insertPersonToMinHeap*();

*writeToFile*("min\_heap\_after\_insertion.txt");

*break*;

*case* 6:

*deleteOldestPerson*();

*writeToFile*("min\_heap\_after\_deletion.txt");

*break*;

*case* 7:

*printf*("Exiting.\n");

*break*;

*default*:

*printf*("Invalid choice! Please try again.\n");

}

} *while* (choice != 7);

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

*free*(persons[i].name);

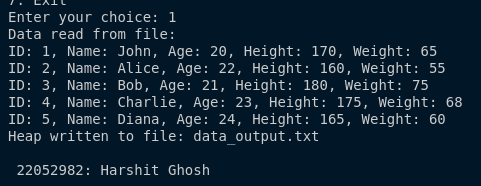
}

*free*(persons);

*return* 0;

}

**INPUT/OUTPUT**

****

**Lab Day 5: Greedy Techniques**

5.1 Aim of the program: Write a program to find the maximum profit nearest to but not

exceeding the given knapsack capacity using the Fractional Knapsack algorithm.

#*include* <stdio.h>

#*include* <stdlib.h>

struct ITEM {

int item\_id;

float item\_profit;

float item\_weight;

float profit\_weight\_ratio;

};

void *swap*(struct ITEM\* a, struct ITEM\* b) {

struct ITEM temp = \*a;

\*a = \*b;

\*b = temp;

}

void *heapify*(struct ITEM items*[]*, int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

*if* (left < n && items[left].profit\_weight\_ratio > items[largest].profit\_weight\_ratio)

largest = left;

*if* (right < n && items[right].profit\_weight\_ratio > items[largest].profit\_weight\_ratio)

largest = right;

*if* (largest != i) {

*swap*(&items[i], &items[largest]);

*heapify*(items, n, largest);

}

}

void *heapSort*(struct ITEM items*[]*, int n) {

*for* (int i = n / 2 - 1; i >= 0; i--)

*heapify*(items, n, i);

*for* (int i = n - 1; i >= 0; i--) {

*swap*(&items[0], &items[i]);

*heapify*(items, i, 0);

}

}

float *fractionalKnapsack*(struct ITEM items*[]*, int n, float knapsack\_capacity) {

float total\_profit = 0.0;

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

*if* (knapsack\_capacity >= items[i].item\_weight) {

knapsack\_capacity -= items[i].item\_weight;

total\_profit += items[i].item\_profit;

*printf*("Item No: %d\tProfit: %.6f\tWeight: %.6f\tAmount to be taken: %.6f\n", items[i].item\_id, items[i].item\_profit, items[i].item\_weight, 1.0);

} *else* {

float fraction = knapsack\_capacity / items[i].item\_weight;

total\_profit += items[i].item\_profit \* fraction;

*printf*("Item No: %d\tProfit: %.6f\tWeight: %.6f\tAmount to be taken: %.6f\n", items[i].item\_id, items[i].item\_profit, items[i].item\_weight, fraction);

*break*;

}

}

*return* total\_profit;

}

int *main*() {

int n;

float knapsack\_capacity;

*printf*("Enter the number of items: ");

*scanf*("%d", &n);

struct ITEM items[n];

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

items[i].item\_id = i + 1;

*printf*("Enter the profit and weight of item no %d: ", i + 1);

*scanf*("%f %f", &items[i].item\_profit, &items[i].item\_weight);

items[i].profit\_weight\_ratio = items[i].item\_profit / items[i].item\_weight;

}

*printf*("Enter the capacity of knapsack: ");

*scanf*("%f", &knapsack\_capacity);

*heapSort*(items, n);

float max\_profit = *fractionalKnapsack*(items, n, knapsack\_capacity);

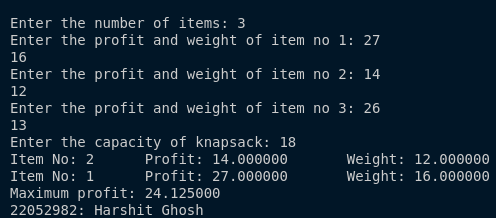
*printf*("Maximum profit: %.6f\n", max\_profit);

*printf*("22052982: Harshit Ghosh\n");

*return* 0;

}

***INPUT/OUTPUT***

******

5.2 Aim of the program: Huffman coding assigns variable length code words to fixed length

input characters based on their frequencies or probabilities of occurrence. Given a set of

characters along with their frequency of occurrences, write a c program to construct a Huffman

tree.

#*include* <stdio.h>

#*include* <stdlib.h>

typedef struct SYMBOL

{

char alphabet;

int frequency;

struct SYMBOL \*l, \*r;

} S;

typedef struct MinHeap

{

int size;

int capacity;

S \*\*arr;

} Min;

S \**createNode*(char alphabet, int frequency)

{

S \*node = (S \*)*malloc*(sizeof(S));

node->alphabet = alphabet;

node->frequency = frequency;

node->l = node->r = *NULL*;

*return* node;

}

Min \**createHeap*(int cap)

{

Min \*m = (Min \*)*malloc*(sizeof(Min));

m->size = 0;

m->capacity = cap;

m->arr = (S \*\*)*malloc*(m->capacity \* sizeof(S \*));

*return* m;

}

void *swap*(S \*\*a, S \*\*b)

{

S \*t = \*a;

\*a = \*b;

\*b = t;

}

void *minHeapify*(Min \*m, int idx)

{

int smallest = idx;

int l = 2 \* idx + 1;

int r = 2 \* idx + 2;

*if* (l < m->size && m->arr[l]->frequency < m->arr[smallest]->frequency)

smallest = l;

*if* (r < m->size && m->arr[r]->frequency < m->arr[smallest]->frequency)

smallest = r;

*if* (smallest != idx)

{

*swap*(&m->arr[smallest], &m->arr[idx]);

*minHeapify*(m, smallest);

}

}

int *issize*(Min \*m)

{

*return* (m->size == 1);

}

S \**extractMin*(Min \*m)

{

S \*temp = m->arr[0];

m->arr[0] = m->arr[m->size - 1];

--m->size;

*minHeapify*(m, 0);

*return* temp;

}

void *insert*(Min \*m, S \*node)

{

++m->size;

int i = m->size - 1;

*while* (i && node->frequency < m->arr[(i - 1) / 2]->frequency)

{

m->arr[i] = m->arr[(i - 1) / 2];

i = (i - 1) / 2;

}

m->arr[i] = node;

}

void *buildMinHeap*(Min \*m)

{

int n = m->size - 1;

*for* (int i = (n - 1) / 2; i >= 0; --i)

*minHeapify*(m, i);

}

Min \**createMinHeap*(char alphabet*[]*, int frequency*[]*, int size)

{

Min \*m = *createHeap*(size);

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

m->arr[i] = *createNode*(alphabet[i], frequency[i]);

m->size = size;

*buildMinHeap*(m);

*return* m;

}

S \**huff*(char alphabet*[]*, int frequency*[]*, int size)

{

S \*l, \*r, \*t;

Min \*m = *createMinHeap*(alphabet, frequency, size);

*while* (!*issize*(m))

{

l = *extractMin*(m);

r = *extractMin*(m);

t = *createNode*('$', l->frequency + r->frequency);

t->l = l;

t->r = r;

*insert*(m, t);

}

*return* *extractMin*(m);

}

void *inorder*(S \*root)

{

*if* (root)

{

*inorder*(root->l);

*if* (root->alphabet != '$')

*printf*("%c ", root->alphabet);

*inorder*(root->r);

}

}

void *printCodes*(S \*root, int arr*[]*, int top)

{

*if* (root->l)

{

arr[top] = 0;

*printCodes*(root->l, arr, top + 1);

}

*if* (root->r)

{

arr[top] = 1;

*printCodes*(root->r, arr, top + 1);

}

*if* (!(root->l) && !(root->r))

{

*printf*("%c: ", root->alphabet);

*for* (int i = 0; i < top; i++)

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

*printf*("\n");

}

}

int *main*()

{

int top = 0;

int arr[100];

int n;

*printf*("Enter the number of alphabets: ");

*scanf*("%d", &n);

char alphabet[n];

int frequency[n];

*printf*("Enter alphabets: ");

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

*scanf*(" %c", &alphabet[i]); // *Ensure space before %c to skip whitespace*

*printf*("Enter frequencies: ");

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

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

S \*root = *huff*(alphabet, frequency, n);

*printf*("Leaf order traversal: ");

*inorder*(root);

*printf*("\n");

*printf*("Huffman codes:\n");

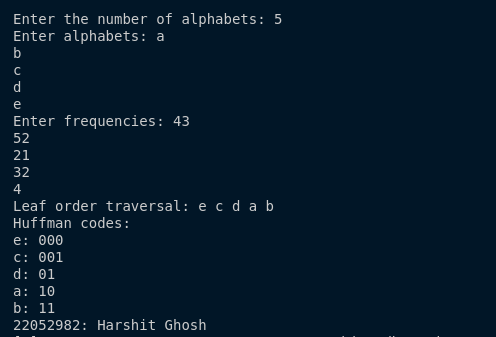
*printCodes*(root, arr, top);

*printf*("22052982: Harshit Ghosh\n");

*return* 0;

}

***INPUT/OUTPUT***

******

***Lab Day 6: Greedy Techniques (Cont…)***

6.1 Aim of the program: Given an undirected weighted connected graph G(V, E) and starring

vertex ‘s’. Maintain a Min-Priority Queue ‘Q’ from the vertex set V and apply Prim’s algorithm

to

● Find the minimum spanning tree T(V, E’). Display the cost adjacency matrix of ‘T’.

● Display total cost of the minimum spanning tree T.

#*include* <stdio.h>

#*include* <stdlib.h>

#*include* <limits.h>

#*include* <stdbool.h>

#*define* *MAX* 100

void *readGraphFromFile*(*const* char \*filename, int graph[*MAX*][*MAX*], int \*V) {

FILE \*file = *fopen*(filename, "r");

*if* (file == *NULL*) {

*printf*("Error opening file.\n");

*exit*(1);

}

*fscanf*(file, "%d", V);

*for* (int i = 0; i < \*V; i++) {

*for* (int j = 0; j < \*V; j++) {

*fscanf*(file, "%d", &graph[i][j]);

}

}

*fclose*(file);

}

int *minKey*(int key*[]*, *bool* mstSet*[]*, int V) {

int min = *INT\_MAX*, min\_index;

*for* (int v = 0; v < V; v++)

*if* (!mstSet[v] && key[v] < min) {

min = key[v];

min\_index = v;

}

*return* min\_index;

}

void *printMSTGraph*(int mstGraph[*MAX*][*MAX*], int V, int totalWeight) {

*printf*("Cost Adjacency Matrix of the Minimum Spanning Tree:\n");

*for* (int i = 0; i < V; i++) {

*for* (int j = 0; j < V; j++) {

*printf*("%d ", mstGraph[i][j]);

}

*printf*("\n");

}

*printf*("Total Weight of the Spanning Tree: %d\n", totalWeight);

}

void *primMST*(int graph[*MAX*][*MAX*], int V, int start) {

int parent[*MAX*];

int key[*MAX*];

*bool* mstSet[*MAX*];

int mstGraph[*MAX*][*MAX*] = {0};

*for* (int i = 0; i < V; i++) {

key[i] = *INT\_MAX*;

mstSet[i] = *false*;

}

key[start] = 0;

parent[start] = -1;

*for* (int count = 0; count < V - 1; count++) {

int u = *minKey*(key, mstSet, V);

mstSet[u] = *true*;

*for* (int v = 0; v < V; v++) {

*if* (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

int totalWeight = 0;

*for* (int i = 1; i < V; i++) {

int u = parent[i];

int v = i;

*if* (u != -1) {

mstGraph[u][v] = graph[u][v];

mstGraph[v][u] = graph[u][v];

totalWeight += graph[u][v];

}

}

*printMSTGraph*(mstGraph, V, totalWeight);

}

int *main*() {

int graph[*MAX*][*MAX*], V;

int startVertex;

*readGraphFromFile*("inUnAdjMat.txt", graph, &V);

*printf*("Enter the starting vertex (1 to %d): ", V);

*scanf*("%d", &startVertex);

startVertex--; // *Convert to 0-based indexing*

*if* (startVertex < 0 || startVertex >= V) {

*printf*("Invalid starting vertex!\n");

*return* 1;

}

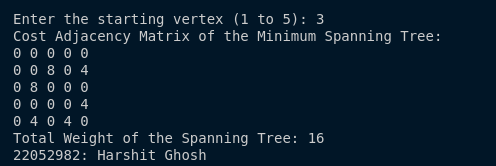
*primMST*(graph, V, startVertex);

*printf*("22052982: Harshit Ghosh\n");

*return* 0;

}

***INPUT/OUTPUT***

******

6.2 Aim of the program: Given an undirected weighted connected graph G(V, E). Apply

Krushkal’s algorithm to

●Find the minimum spanning tree T(V, E’) and Display the selected edges of G.

● Display total cost of the minimum spanning tree T.

#*include* <stdio.h>

#*include* <stdlib.h>

#*define* *MAX* 100

typedef struct

{

int u, v, w;

} Edge;

int parent[*MAX*], rank[*MAX*];

Edge edges[*MAX*], mst[*MAX*];

int n, m;

int *find\_set*(int i)

{

*if* (parent[i] != i)

parent[i] = *find\_set*(parent[i]);

*return* parent[i];

}

void *union\_sets*(int u, int v)

{

int rootU = *find\_set*(u);

int rootV = *find\_set*(v);

*if* (rank[rootU] > rank[rootV])

parent[rootV] = rootU;

*else* *if* (rank[rootU] < rank[rootV])

parent[rootU] = rootV;

*else*

{

parent[rootV] = rootU;

rank[rootU]++;

}

}

int *compare\_edges*(*const* void \*a, *const* void \*b)

{

Edge \*edgeA = (Edge \*)a;

Edge \*edgeB = (Edge \*)b;

*return* edgeA->w - edgeB->w;

}

void *kruskal*()

{

int mstWeight = 0;

int mstEdges = 0;

*for* (int i = 1; i <= n; i++)

{

parent[i] = i;

rank[i] = 0;

}

*qsort*(edges, m, sizeof(Edge), *compare\_edges*);

*for* (int i = 0; i < m && mstEdges < n - 1; i++)

{

int u = edges[i].u;

int v = edges[i].v;

int w = edges[i].w;

*if* (*find\_set*(u) != *find\_set*(v))

{

mst[mstEdges++] = edges[i];

mstWeight += w;

*union\_sets*(u, v);

}

}

*printf*("Edge \tCost\n");

*for* (int i = 0; i < mstEdges; i++)

{

*printf*("%d--%d \t%d\n", mst[i].u, mst[i].v, mst[i].w);

}

*printf*("Total Weight of the Spanning Tree: %d\n", mstWeight);

}

int *main*()

{

*printf*("Enter the number of vertices and edges: ");

*scanf*("%d %d", &n, &m);

*printf*("Enter the edges (u v w):\n");

*for* (int i = 0; i < m; i++)

{

*scanf*("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);

}

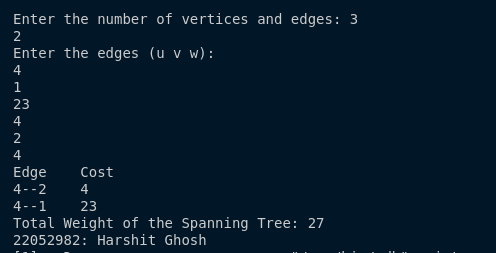
*kruskal*();

*printf*("22052982: Harshit Ghosh\n");

*return* 0;

}

***INPUT/OUTPUT***

******

**Lab Day 7 : Greedy Techniques (Cont...) – Single Source Shortest path**

7.1 Aim of the program: Given a directed graph G (V, E) and a starting vertex ‘s’.

● Determine the lengths of the shortest paths from the starting vertex ‘s’ to all other

vertices in the graph G using Dijkstra’s Algorithm.

● Display the shortest path from the given source ‘s’ to all other vertices.

#*include* <stdio.h>

#*include* <limits.h>

#*define* *INF* *INT\_MAX*

#*define* *MAX* 100

// *Function to print the path from source to destination using the previous node array*

void *printPath*(int prev*[]*, int i) {

*if* (prev[i] == -1)

*return*;

*printPath*(prev, prev[i]);

*printf*("->%d", i + 1);

}

// *Function to find the vertex with the minimum distance value that hasn't been visited*

int *minDistance*(int dist*[]*, int visited*[]*, int n) {

int min = *INF*, min\_index = -1;

*for* (int v = 0; v < n; v++) {

*if* (!visited[v] && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

}

*return* min\_index;

}

// *Function implementing Dijkstra's algorithm*

void *dijkstra*(int graph[*MAX*][*MAX*], int src, int n) {

int dist[*MAX*], visited[*MAX*], prev[*MAX*];

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

dist[i] = *INF*;

visited[i] = 0;

prev[i] = -1;

}

dist[src] = 0;

*for* (int count = 0; count < n - 1; count++) {

int u = *minDistance*(dist, visited, n);

*if* (u == -1)

*break*;

visited[u] = 1;

*for* (int v = 0; v < n; v++) {

*if* (!visited[v] && graph[u][v] != *INF* && dist[u] != *INF* && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

prev[v] = u;

}

}

}

// *Display the results*

*printf*("Source\tDestination\tCost\tPath\n");

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

*printf*("%d\t%d\t", src + 1, i + 1);

*if* (dist[i] == *INF*) {

*printf*("-\t-\n");

} *else* {

*printf*("%d\t%d", dist[i], src + 1);

*printPath*(prev, i);

*printf*("\n");

}

}

}

int *main*() {

FILE \*file = *fopen*("inDiAdjMat1.txt", "r");

*if* (file == *NULL*) {

*printf*("Error opening file!\n");

*return* 1;

}

int n, graph[*MAX*][*MAX*];

*printf*("Enter the Number of Vertices: ");

*scanf*("%d", &n);

*printf*("Enter the Source Vertex (1 to %d): ", n);

int src;

*scanf*("%d", &src);

src--;

*if* (src < 0 || src >= n) {

*printf*("Invalid source vertex!\n");

*return* 1;

}

// *Reading the graph from file*

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

*for* (int j = 0; j < n; j++) {

*fscanf*(file, "%d", &graph[i][j]);

*if* (graph[i][j] == 0 && i != j) {

graph[i][j] = *INF*; // *No edge between different nodes*

}

}

}

*fclose*(file);

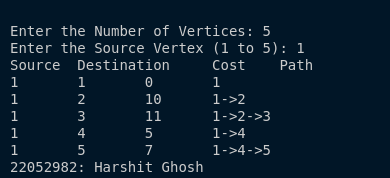
*dijkstra*(graph, src, n);

*printf*("22052982: Harshit Ghosh\n");

*return* 0;

}

***INPUT/OUTPUT***

******

***Lab Day 8: Dynamic Programming***

8.1 Aim of the program: Write a program to implement the matrix chain multiplication problem

using M-table & S-table to find optimal parenthesization of a matrix-chain product. Print the

number of scalar multiplications required for the given input.

#*include* <stdio.h>

#*include* <limits.h>

#*define* *MAX* 100

// *Function to print the optimal parenthesization*

void *printOptimalParens*(int s[*MAX*][*MAX*], int i, int j) {

*if* (i == j) {

*printf*("A%d", i + 1);

} *else* {

*printf*("(");

*printOptimalParens*(s, i, s[i][j]);

*printOptimalParens*(s, s[i][j] + 1, j);

*printf*(")");

}

}

// *Function to implement matrix chain multiplication*

void *matrixChainOrder*(int p*[]*, int n) {

int m[*MAX*][*MAX*]; // *M-table to store the minimum scalar multiplications*

int s[*MAX*][*MAX*]; // *S-table to store the splits*

int i, j, k, L, q;

// *Initialize the M-table to 0 for diagonal elements (base case)*

*for* (i = 1; i < n; i++)

m[i][i] = 0;

// *Fill the M-table using dynamic programming*

*for* (L = 2; L < n; L++) { // *L is the chain length*

*for* (i = 1; i <= n - L; i++) {

j = i + L - 1;

m[i][j] = *INT\_MAX*;

*for* (k = i; k < j; k++) {

q = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];

*if* (q < m[i][j]) {

m[i][j] = q;

s[i][j] = k;

}

}

}

}

// *Print M-table*

*printf*("M Table:\n");

*for* (i = 1; i < n; i++) {

*for* (j = i; j < n; j++) {

*printf*("%7d ", m[i][j]);

}

*printf*("\n");

}

// *Print S-table*

*printf*("\nS Table:\n");

*for* (i = 1; i < n - 1; i++) {

*for* (j = i + 1; j < n; j++) {

*printf*("%7d ", s[i][j]);

}

*printf*("\n");

}

// *Print the optimal parenthesization*

*printf*("\nOptimal parenthesization: ");

*printOptimalParens*(s, 1, n - 1);

*printf*("\nThe optimal ordering of the given matrices requires %d scalar multiplications.\n", m[1][n - 1]);

}

int *main*() {

int n, i;

*printf*("Enter number of matrices: ");

*scanf*("%d", &n);

int p[*MAX*]; // *To store dimensions of matrices*

*printf*("Enter row and column size of A1: ");

*scanf*("%d%d", &p[0], &p[1]);

*for* (i = 2; i <= n; i++) {

*printf*("Enter row and column size of A%d: ", i);

int row, col;

*scanf*("%d%d", &row, &col);

*if* (row != p[i - 1]) {

*printf*("Incompatible dimensions!\n");

*return* 1;

}

p[i] = col;

}

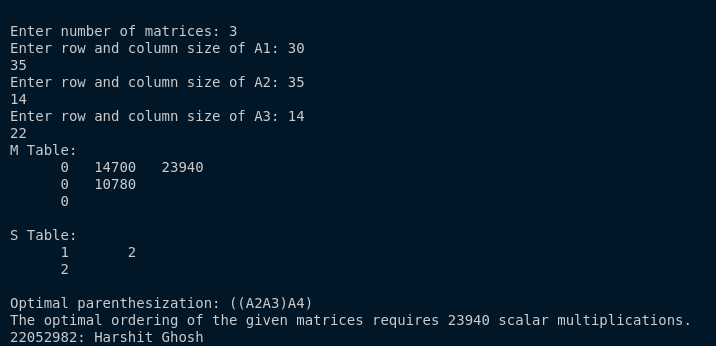
*matrixChainOrder*(p, n + 1);

*printf*("22052982: Harshit Ghosh\n");

*return* 0;

}

***INPUT/OUTPUT***



8.2 Aim of the program: Write a program to find out the Longest Common Subsequence of two

given strings. Calculate length of the LCS.

#*include* <stdio.h>

#*include* <string.h>

#*define* *MAX* 100

// *Function to find the length of LCS*

int *LCS*(char \*X, char \*Y, char \*lcs\_result) {

int m = *strlen*(X);

int n = *strlen*(Y);

int L[*MAX*][*MAX*]; // *DP table to store lengths of LCS*

// *Building the LCS table in bottom-up fashion*

*for* (int i = 0; i <= m; i++) {

*for* (int j = 0; j <= n; j++) {

*if* (i == 0 || j == 0)

L[i][j] = 0;

*else* *if* (X[i - 1] == Y[j - 1])

L[i][j] = L[i - 1][j - 1] + 1;

*else*

L[i][j] = (L[i - 1][j] > L[i][j - 1]) ? L[i - 1][j] : L[i][j - 1];

}

}

// *The length of LCS is L[m][n]*

int index = L[m][n];

int lcs\_length = index;

lcs\_result[index] = '\0';

// *Backtracking to find the LCS string from L[m][n]*

int i = m, j = n;

*while* (i > 0 && j > 0) {

*if* (X[i - 1] == Y[j - 1]) {

lcs\_result[index - 1] = X[i - 1];

i--;

j--;

index--;

}

*else* *if* (L[i - 1][j] > L[i][j - 1])

i--;

*else*

j--;

}

*return* lcs\_length;

}

int *main*() {

char X[*MAX*], Y[*MAX*], lcs\_result[*MAX*];

*printf*("Enter the first string into an array: ");

*scanf*("%s", X);

*printf*("Enter the second string into an array: ");

*scanf*("%s", Y);

int lcs\_length = *LCS*(X, Y, lcs\_result);

*printf*("LCS: %s\n", lcs\_result);

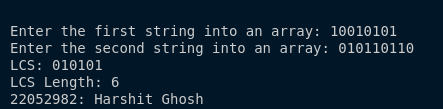
*printf*("LCS Length: %d\n", lcs\_length);

*printf*("22052982: Harshit Ghosh\n");

*return* 0;

}

***INPUT/OUTPUT***

******