***Lab 1***

**Question 1.1**

Write a program to find out the second smallest and second largest element stored in an array of n integers.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#define NUMS 10

int main() {

printf("Developed by Soudeep Ghoshal | 2205421\n");

FILE \*file = fopen("input0.txt", "r");

if (file == NULL) {

perror("File does not exist");

return EXIT\_FAILURE;

}

int num[NUMS];

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

if (fscanf(file, "%d", &num[i]) != 1) {

perror("Error reading integer from file");

fclose(file);

return EXIT\_FAILURE;

}

}

fclose(file);

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

int smallest = i;

for (int j = i + 1; j < NUMS; j++) {

if (num[j] < num[smallest])

smallest = j;

}

if (num[smallest] == num[i])

continue;

num[smallest] += num[i];

num[i] = num[smallest] - num[i];

num[smallest] -= num[i];

}

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

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

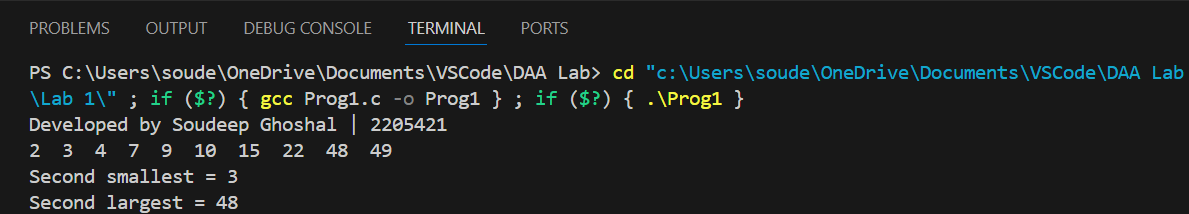
}

printf("\nSecond smallest = %d\n", num[1]);

printf("Second largest = %d\n", num[NUMS - 2]);

return 0;

}

**Output**

**Question 1.2**

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].

**Source Code**

#include <stdio.h>

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

int N = 0;

printf("Enter number of elements: ");

scanf("%d", &N);

int array[N];

printf("Enter %d integers:\n", N);

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

{

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

}

// Prefix Sum Array

int prefixSum[N];

prefixSum[0] = array[0];

for (int i = 1; i < N; i++)

{

prefixSum[i] = prefixSum[i - 1] + array[i];

}

printf("Prefix Sum Array:\n");

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

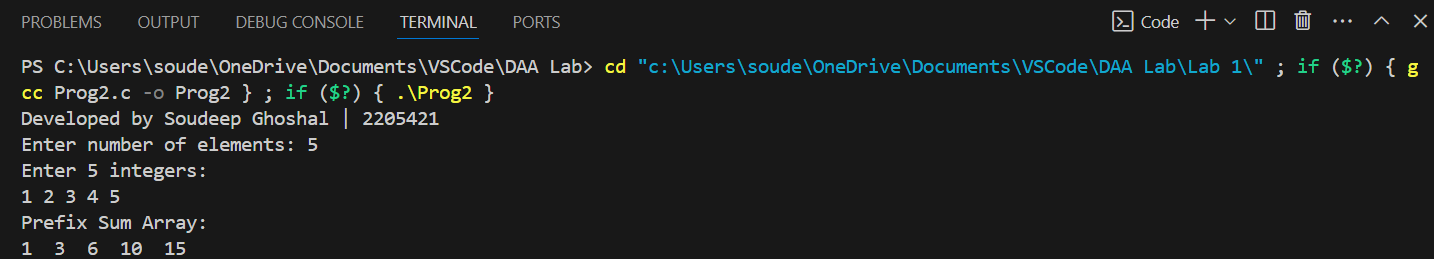
{

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

}

}

**Output**

****

**Question 1.3**

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.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#define NUMS 30

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

FILE \*file = fopen("input0.txt", "r");

if (file == NULL)

{

perror("File does not exist");

return EXIT\_FAILURE;

}

int num[NUMS];

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

{

if (fscanf(file, "%d", &num[i]) != 1)

{

perror("Error reading integer from file");

fclose(file);

return EXIT\_FAILURE;

}

}

fclose(file);

int freq[NUMS][2];

int k = 0;

int flag = 0;

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

{

flag = 0;

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

{

if (freq[j][0] == num[i])

{

freq[j][1]++;

flag = 1;

break;

}

}

if (flag == 0)

{

freq[k][0] = num[i];

freq[k][1] = 1;

k++;

}

}

int dupli\_count = 0;

int max\_dupli\_index = 0;

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

{

if (freq[i][1] > 1)

{

dupli\_count++;

if (freq[i][1] > freq[max\_dupli\_index][1])

max\_dupli\_index = i;

}

}

printf("The content of the array: ");

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

{

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

}

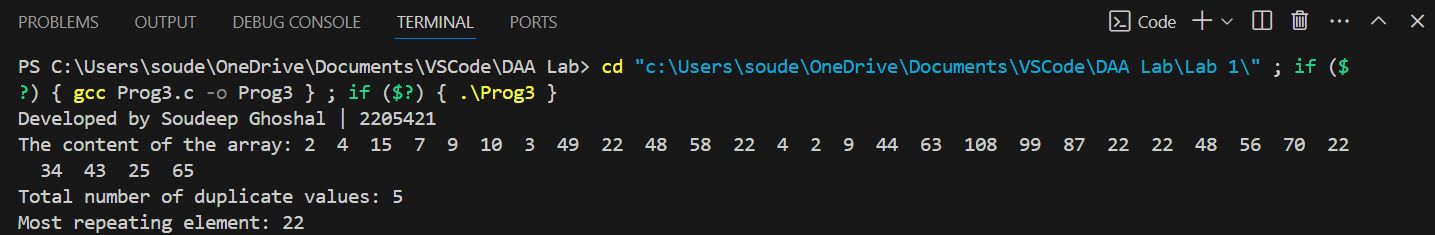
printf("\nTotal number of duplicate values: %d", dupli\_count);

printf("\nMost repeating element: %d", freq[max\_dupli\_index][0]);

return 0;

}

**Output**

****

**Question 1.4**

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.

**Source Code**

#include <stdio.h>

void ROTATE\_RIGHT(int \*A, int rot)

{

int temp = A[rot - 1];

for (int i = rot - 1; i > 0; i--)

{

A[i] = A[i - 1];

}

A[0] = temp;

}

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

int N = 0;

printf("Enter number of elements: ");

scanf("%d", &N);

int array[N];

printf("Enter %d integers:\n", N);

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

{

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

}

printf("Enter rotation value: ");

int rot = 0;

scanf("%d", &rot);

printf("Original Array:\n");

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

{

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

}

ROTATE\_RIGHT(array, rot);

printf("\nRotated Array:\n");

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

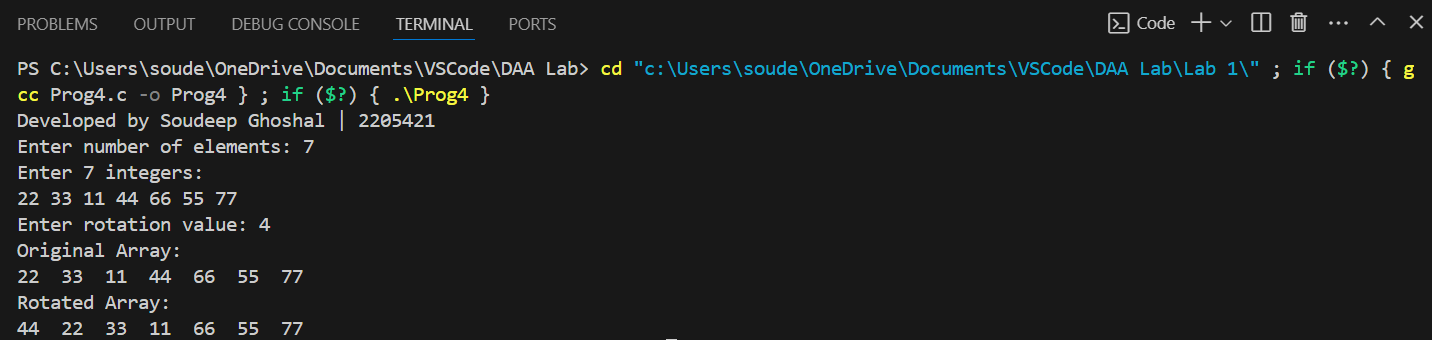
{

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

}

}

**Output**

****

***Lab 2***

**Question 2.1**

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.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

void convert(FILE \*outputFile, int n)

{

if (n > 1)

{

convert(outputFile, n / 2);

}

fprintf(outputFile, "%d", n % 2);

}

void displayContent(FILE \*inputFile, FILE \*outputFile, int n)

{

char c1[100], c2[100];

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

{

if (fscanf(inputFile, "%s", c1) != 1 || fscanf(outputFile, "%s", c2) != 1)

{

break;

}

printf("Binary equivalent of %s is %s\n", c1, c2);

}

}

void decToBin(const char \*inputFileName, const char \*outputFileName, int n)

{

FILE \*inputFile = fopen(inputFileName, "r");

if (inputFile == NULL)

{

perror("Error opening file.");

exit(EXIT\_FAILURE);

}

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

if (outputFile == NULL)

{

perror("Error opening output file.");

fclose(inputFile);

exit(EXIT\_FAILURE);

}

int num;

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

{

if (fscanf(inputFile, "%d", &num) != 1)

break;

convert(outputFile, num);

fprintf(outputFile, "\n");

}

printf("Conversions:\n");

rewind(inputFile);

rewind(outputFile);

displayContent(inputFile, outputFile, n);

fclose(inputFile);

fclose(outputFile);

}

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

char input[15];

char output[15];

int n;

printf("Enter number of Decimals to convert: ");

scanf("%d", &n);

printf("Enter name of input file: ");

scanf("%s", input);

printf("Enter name of output file: ");

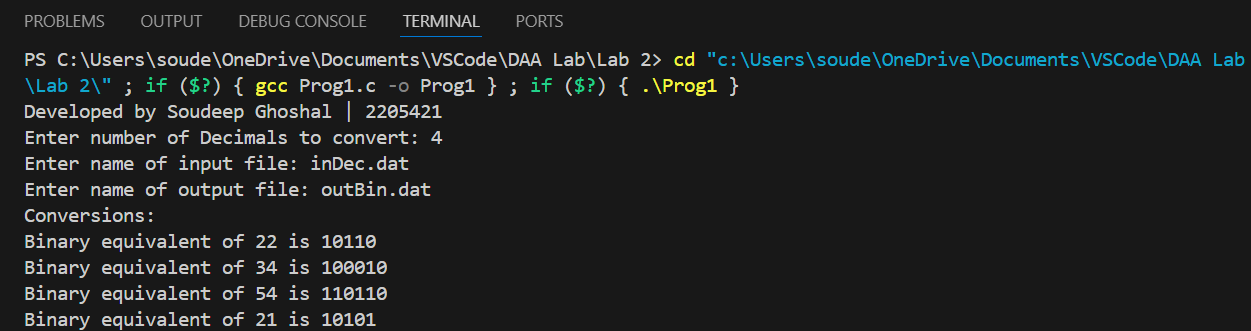
scanf("%s", output);

decToBin(input, output, n);

return 0;

}

**Output**

****

**Question 2.2**

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.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

void find(FILE \*outputFile, int n1, int n2)

{

if (n2 == 0)

fprintf(outputFile, "%d", n1);

else

find(outputFile, n2, n1 % n2);

}

void displayContent(FILE \*inputFile, FILE \*outputFile, int n)

{

char c1[100], c2[100], c3[100];

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

{

if (fscanf(inputFile, "%s", c1) != 1 || fscanf(inputFile, "%s", c2) != 1 || fscanf(outputFile, "%s", c3) != 1)

{

break;

}

printf("GCD of %s and %s is %s\n", c1, c2, c3);

}

}

void findGcd(const char \*inputFileName, const char \*outputFileName, int n)

{

FILE \*inputFile = fopen(inputFileName, "r");

if (inputFile == NULL)

{

perror("Error opening file.");

exit(EXIT\_FAILURE);

}

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

if (outputFile == NULL)

{

perror("Error opening output file.");

fclose(inputFile);

exit(EXIT\_FAILURE);

}

int num1, num2;

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

{

if (fscanf(inputFile, "%d", &num1) != 1 || fscanf(inputFile, "%d", &num2) != 1)

break;

find(outputFile, num1, num2);

fprintf(outputFile, "\n");

}

printf("Conversions:\n");

rewind(inputFile);

rewind(outputFile);

displayContent(inputFile, outputFile, n);

fclose(inputFile);

fclose(outputFile);

}

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

char input[15];

char output[15];

int n;

printf("Enter number of pairs of numbers to process: ");

scanf("%d", &n);

printf("Enter name of input file: ");

scanf("%s", input);

printf("Enter name of output file: ");

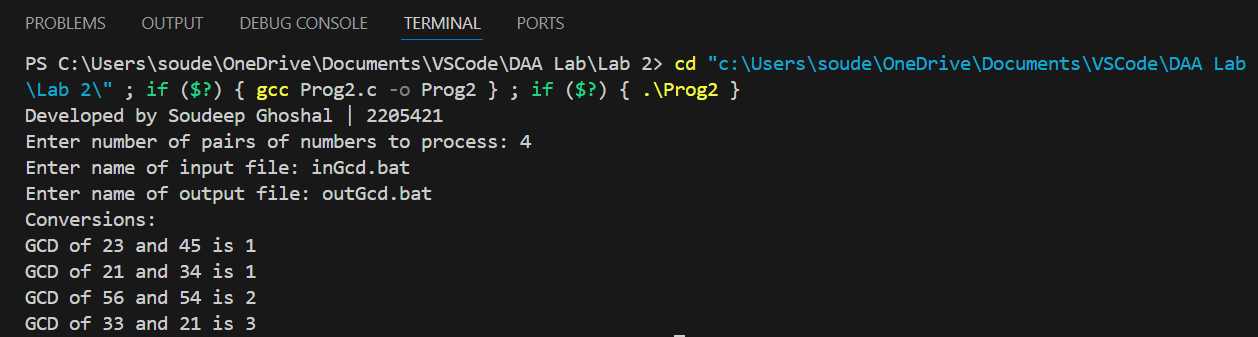
scanf("%s", output);

findGcd(input, output, n);

return 0;

}

**Output**

****

***Lab 3***

**Question 3.1**

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.

**Source Code**

/\* 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. \*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

void mergeAsc(int arr[], int p, int q, int r, int \*comp)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

R[j] = arr[q + 1 + j];

int i = 0, j = 0, k = p;

while (i < n1 && j < n2)

{

(\*comp)++;

if (L[i] <= R[j])

{

arr[k++] = L[i];

i++;

}

else

{

arr[k++] = R[j];

j++;

}

}

while (i < n1)

{

arr[k++] = L[i];

i++;

}

while (j < n2)

{

arr[k++] = R[j];

j++;

}

}

void mergeSortAsc(int a[], int p, int r, int \*comp)

{

if (p < r)

{

int q = (p + r) / 2;

mergeSortAsc(a, p, q, comp);

mergeSortAsc(a, q + 1, r, comp);

mergeAsc(a, p, q, r, comp);

}

}

void mergeDesc(int arr[], int p, int q, int r, int \*comp)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

R[j] = arr[q + 1 + j];

int i = 0, j = 0, k = p;

while (i < n1 && j < n2)

{

(\*comp)++;

if (L[i] >= R[j])

{

arr[k++] = L[i];

i++;

}

else

{

arr[k++] = R[j];

j++;

}

}

while (i < n1)

{

arr[k++] = L[i];

i++;

}

while (j < n2)

{

arr[k++] = R[j];

j++;

}

}

void mergeSortDesc(int a[], int p, int r, int \*comp)

{

if (p < r)

{

int q = (p + r) / 2;

mergeSortDesc(a, p, q, comp);

mergeSortDesc(a, q + 1, r, comp);

mergeDesc(a, p, q, r, comp);

}

}

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

clock\_t start = clock(), end = clock();

double cpu\_time\_used;

int ch = 0;

printf("MAIN MENU (MERGE SORT)\n");

printf("1. Ascending Order\n");

printf("2. Descending Order\n");

printf("3. ERROR (EXIT)\n");

printf("Enter choice: ");

scanf("%d", &ch);

start = clock();

if (ch == 3)

{

printf("Exiting program.\n");

exit(EXIT\_SUCCESS);

}

int n;

printf("Enter number of terms: ");

scanf("%d", &n);

int num[n];

int comp = 0;

FILE \*inputFile = fopen("inRand.dat", "r");

if (inputFile == NULL)

{

perror("Error opening file\n");

exit(EXIT\_FAILURE);

}

FILE \*outputFile = fopen("outMergeSort.dat", "w");

if (outputFile == NULL)

{

perror("Error opening output file.");

fclose(inputFile);

exit(EXIT\_FAILURE);

}

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

{

if (fscanf(inputFile, "%d", &num[i]) != 1)

break;

}

printf("Before Sorting:\n");

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

{

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

}

printf("\n");

switch (ch)

{

case 1:

mergeSortAsc(num, 0, n - 1, &comp);

break;

case 2:

mergeSortDesc(num, 0, n - 1, &comp);

break;

default:

printf("Wrong Choice.");

}

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

{

fprintf(outputFile, "%d\n", num[i]);

}

end = clock();

fclose(inputFile);

fclose(outputFile);

printf("After Sorting:\n");

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

{

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

}

printf("\n");

printf("Number of comparisons: %d\n", comp);

cpu\_time\_used = (((double)(end - start)) / CLOCKS\_PER\_SEC) \* pow(10, 9);

printf("Execution time: %lf ns", cpu\_time\_used);

return 0;

}

**Output**



**Question 3.2**

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.

**Source Code**

/\* 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. \*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partitionAsc(int arr[], int low, int high, int \*comp)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

(\*comp)++;

if (arr[j] < pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSortAsc(int arr[], int low, int high, int \*comp, int depth)

{

if (low < high)

{

int pi = partitionAsc(arr, low, high, comp);

quickSortAsc(arr, low, pi - 1, comp, depth + 1);

quickSortAsc(arr, pi + 1, high, comp, depth + 1);

}

}

int partitionDesc(int arr[], int low, int high, int \*comp)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

(\*comp)++;

if (arr[j] > pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSortDesc(int arr[], int low, int high, int \*comp, int depth)

{

if (low < high)

{

int pi = partitionDesc(arr, low, high, comp);

quickSortDesc(arr, low, pi - 1, comp, depth + 1);

quickSortDesc(arr, pi + 1, high, comp, depth + 1);

}

}

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

clock\_t start, end;

double cpu\_time\_used;

int ch = 0;

printf("MAIN MENU (QUICK SORT)\n");

printf("1. Ascending Order\n");

printf("2. Descending Order\n");

printf("3. ERROR (EXIT)\n");

printf("Enter choice: ");

scanf("%d", &ch);

start = clock();

if (ch == 3)

{

printf("Exiting program.\n");

exit(EXIT\_SUCCESS);

}

int n;

printf("Enter number of terms: ");

scanf("%d", &n);

int num[n];

int comp = 0;

FILE \*inputFile = fopen("inRand.dat", "r");

if (inputFile == NULL)

{

perror("Error opening file\n");

exit(EXIT\_FAILURE);

}

FILE \*outputFile = fopen("outQuickSort.dat", "w");

if (outputFile == NULL)

{

perror("Error opening output file.");

fclose(inputFile);

exit(EXIT\_FAILURE);

}

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

{

if (fscanf(inputFile, "%d", &num[i]) != 1)

break;

}

printf("Before Sorting:\n");

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

{

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

}

printf("\n");

switch (ch)

{

case 1:

quickSortAsc(num, 0, n - 1, &comp, 0);

break;

case 2:

quickSortDesc(num, 0, n - 1, &comp, 0);

break;

default:

printf("Wrong Choice.");

}

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

{

fprintf(outputFile, "%d\n", num[i]);

}

fclose(inputFile);

fclose(outputFile);

end = clock();

printf("After Sorting:\n");

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

{

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

}

printf("\n");

printf("Number of comparisons: %d\n", comp);

cpu\_time\_used = (((double)(end - start)) / CLOCKS\_PER\_SEC) \* pow(10, 9);

printf("Execution time: %lf", cpu\_time\_used);

return 0;

}

**Output**

****

***Lab 4***

**Question 4.1**

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

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_NAME 50

struct person

{

int id;

char \*name;

int age;

int height;

int weight;

};

struct heap

{

struct person \*array;

int size;

int capacity;

int (\*compare)(struct person \*, struct person \*);

};

struct heap \*create\_heap(int capacity, int (\*compare)(struct person \*, struct person \*));

void read\_data(struct heap \*h, const char \*filename);

void print\_data(struct heap \*h);

void heapify(struct heap \*h, int i);

void build\_heap(struct heap \*h);

void insert\_person(struct heap \*h, struct person p);

void display\_weight\_youngest(struct heap \*h);

int compare\_age\_min(struct person \*a, struct person \*b);

int compare\_weight\_max(struct person \*a, struct person \*b);

void free\_heap(struct heap \*h);

struct person delete\_oldest\_person(struct heap \*h);

void heapify(struct heap \*h, int i)

{

int smallest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < h->size && h->compare(&h->array[left], &h->array[smallest]) < 0)

smallest = left;

if (right < h->size && h->compare(&h->array[right], &h->array[smallest]) < 0)

smallest = right;

if (smallest != i)

{

struct person temp = h->array[i];

h->array[i] = h->array[smallest];

h->array[smallest] = temp;

heapify(h, smallest);

}

}

struct heap \*create\_heap(int capacity, int (\*compare)(struct person \*, struct person \*))

{

struct heap \*h = (struct heap \*)malloc(sizeof(struct heap));

h->array = (struct person \*)malloc(capacity \* sizeof(struct person));

h->capacity = capacity;

h->size = 0;

h->compare = compare;

return h;

}

void read\_data(struct heap \*h, const char \*filename)

{

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

if (file == NULL)

{

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

return;

}

h->size = 0;

char name\_buffer[MAX\_NAME];

while (fscanf(file, "%d %49s %d %d %d", &h->array[h->size].id, name\_buffer,

&h->array[h->size].age, &h->array[h->size].height, &h->array[h->size].weight) == 5)

{

h->array[h->size].name = strdup(name\_buffer);

if (h->array[h->size].name == NULL)

{

printf("Memory allocation error for name.\n");

fclose(file);

return;

}

h->size++;

if (h->size >= h->capacity)

{

h->capacity \*= 2;

h->array = (struct person \*)realloc(h->array, h->capacity \* sizeof(struct person));

if (h->array == NULL)

{

printf("Memory reallocation error.\n");

fclose(file);

return;

}

}

}

fclose(file);

build\_heap(h);

printf("Data read successfully. Total records: %d\n", h->size);

}

void print\_data(struct heap \*h)

{

printf("ID\tName\t\tAge\tHeight(cm)\tWeight(kg)\n");

for (int i = 0; i < h->size; i++)

{

printf("%d\t%10s\t%d\t%d\t\t%d\n", h->array[i].id, h->array[i].name, h->array[i].age,

h->array[i].height, h->array[i].weight);

}

}

void build\_heap(struct heap \*h)

{

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

heapify(h, i);

}

void insert\_person(struct heap \*h, struct person p)

{

if (h->size >= h->capacity)

{

h->capacity \*= 2;

h->array = (struct person \*)realloc(h->array, h->capacity \* sizeof(struct person));

if (h->array == NULL)

{

printf("Memory reallocation failed\n");

return;

}

}

int i = h->size;

h->array[i] = p;

h->size++;

while (i > 0 && h->compare(&h->array[i], &h->array[(i - 1) / 2]) < 0)

{

struct person temp = h->array[i];

h->array[i] = h->array[(i - 1) / 2];

h->array[(i - 1) / 2] = temp;

i = (i - 1) / 2;

}

}

struct person delete\_oldest\_person(struct heap \*h)

{

if (h->size <= 0)

{

printf("Heap is empty\n");

struct person dummy = {-1, "", -1, -1, -1};

return dummy;

}

int oldest\_index = 0;

for (int i = 1; i < h->size; i++)

{

if (h->array[i].age > h->array[oldest\_index].age)

{

oldest\_index = i;

}

}

struct person oldest = h->array[oldest\_index];

h->array[oldest\_index] = h->array[h->size - 1];

h->size--;

heapify(h, oldest\_index);

return oldest;

}

void display\_weight\_youngest(struct heap \*h)

{

if (h->size > 0)

{

printf("Weight of youngest person: %d kg\n", h->array[0].weight);

}

else

{

printf("Heap is empty\n");

}

}

int compare\_age\_min(struct person \*a, struct person \*b)

{

return a->age - b->age;

}

int compare\_weight\_max(struct person \*a, struct person \*b)

{

return b->weight - a->weight;

}

void free\_heap(struct heap \*h)

{

for (int i = 0; i < h->size; i++)

{

free(h->array[i].name);

}

free(h->array);

free(h);

}

int main()

{

struct heap \*h = NULL;

int choice;

char name\_buffer[MAX\_NAME];

printf("MAIN 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");

do

{

printf("Enter option: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

if (h != NULL)

{

free\_heap(h);

}

h = create\_heap(100, compare\_age\_min);

read\_data(h, "people.dat");

print\_data(h);

break;

case 2:

if (h != NULL)

{

h->compare = compare\_age\_min;

build\_heap(h);

printf("Min-heap based on age created.\n");

}

else

{

printf("Please read data first.\n");

}

break;

case 3:

if (h != NULL)

{

h->compare = compare\_weight\_max;

build\_heap(h);

printf("Max-heap based on weight created.\n");

}

else

{

printf("Please read data first.\n");

}

break;

case 4:

if (h != NULL && h->compare == compare\_age\_min)

{

display\_weight\_youngest(h);

}

else

{

printf("Please create a min-heap based on age first.\n");

}

break;

case 5:

if (h != NULL && h->compare == compare\_age\_min)

{

struct person p;

printf("Enter id, name, age, height, weight: ");

scanf("%d %s %d %d %d", &p.id, name\_buffer, &p.age, &p.height, &p.weight);

p.name = strdup(name\_buffer);

insert\_person(h, p);

printf("Person inserted into the min-heap.\n");

}

else

{

printf("Please create a min-heap based on age first.\n");

}

break;

case 6:

if (h != NULL && h->compare == compare\_age\_min)

{

struct person p = delete\_oldest\_person(h);

if (p.id != -1)

{

printf("Oldest person removed: %s (Age: %d)\n", p.name, p.age);

free(p.name);

}

}

else

{

printf("Please create a min-heap based on age first.\n");

}

break;

case 7:

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

break;

default:

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

}

} while (choice != 7);

if (h != NULL)

{

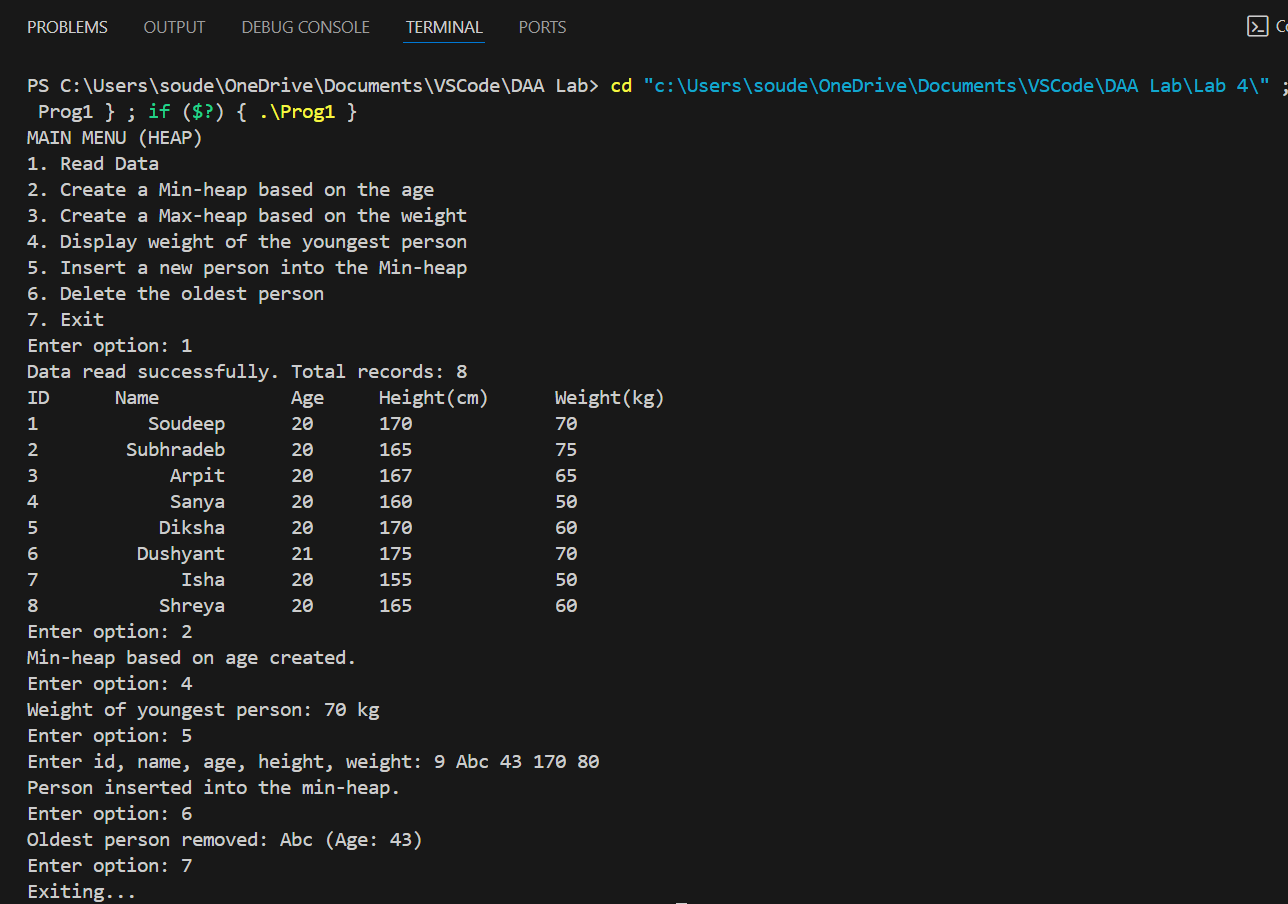
free\_heap(h);

}

return 0;

}

**Output**

****

***Lab 5***

**Question 5.1**

Write a program to find the maximum profit nearest to but not exceeding the given knapsack capacity using the Fractional Knapsack algorithm.

Note: Declare a structure ITEM having data members item\_id, item\_profit, item\_weight and profit\_weight\_ratio. Apply heap sort technique to sort the items in non-increasing order, according to their profit/weight.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

typedef struct

{

int item\_id;

int item\_profit;

int item\_weight;

float profit\_weight\_ratio;

} ITEM;

void heapify(ITEM \*arr, int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left].profit\_weight\_ratio < arr[largest].profit\_weight\_ratio)

largest = left;

if (right < n && arr[right].profit\_weight\_ratio < arr[largest].profit\_weight\_ratio)

largest = right;

if (largest != i)

{

ITEM temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

void heap\_sort(ITEM \*arr, int n)

{

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

heapify(arr, n, i);

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

{

ITEM temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

float fractional\_knapsack(ITEM \*items, int n, int knapsack\_capacity)

{

float total\_profit = 0.0;

int remaining\_capacity = knapsack\_capacity;

heap\_sort(items, n);

printf("\nSorted items:\n");

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

{

printf("Item %d: profit = %d, weight = %d, ratio = %.2f\n",

items[i].item\_id, items[i].item\_profit, items[i].item\_weight, items[i].profit\_weight\_ratio);

}

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

{

if (items[i].item\_weight <= remaining\_capacity)

{

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

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

printf("Adding item %d completely. Profit: %.2f, Remaining capacity: %d\n",

items[i].item\_id, total\_profit, remaining\_capacity);

}

else

{

float fraction = (float)remaining\_capacity / items[i].item\_weight;

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

printf("Adding fraction %.2f of item %d. Profit: %.2f, Remaining capacity: 0\n",

fraction, items[i].item\_id, total\_profit);

remaining\_capacity = 0;

break;

}

}

return total\_profit;

}

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

int n, knapsack\_capacity;

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

scanf("%d", &n);

printf("Enter the knapsack capacity: ");

scanf("%d", &knapsack\_capacity);

ITEM \*items = (ITEM \*)malloc(n \* sizeof(ITEM));

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

{

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

printf("Enter the profit and weight of item %d: ", items[i].item\_id);

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

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

}

float max\_profit = fractional\_knapsack(items, n, knapsack\_capacity);

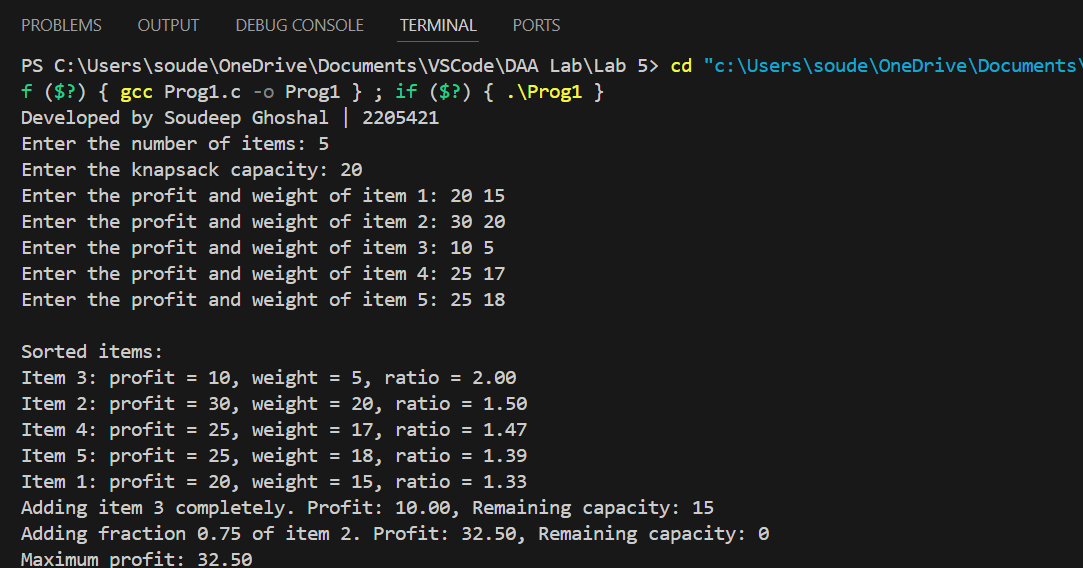
printf("Maximum profit: %.2f\n", max\_profit);

free(items);

return 0;

}

**Output**

****

**Question 5.2**

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.

Note#

● Declare a structure SYMBOL having members alphabet and frequency. Create a Min-Priority Queue, keyed on frequency attributes.

● Create an array of structures where size=number of alphabets.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

char alphabet;

int frequency;

struct Node \*left, \*right;

} Node;

typedef struct

{

char alphabet;

int frequency;

} SYMBOL;

typedef struct

{

int size;

int capacity;

Node \*\*array;

} MinHeap;

Node \*createNode(char alphabet, int frequency)

{

Node \*newNode = (Node \*)malloc(sizeof(Node));

newNode->alphabet = alphabet;

newNode->frequency = frequency;

newNode->left = newNode->right = NULL;

return newNode;

}

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

{

Node \*temp = \*a;

\*a = \*b;

\*b = temp;

}

MinHeap \*createMinHeap(int capacity)

{

MinHeap \*minHeap = (MinHeap \*)malloc(sizeof(MinHeap));

minHeap->size = 0;

minHeap->capacity = capacity;

minHeap->array = (Node \*\*)malloc(capacity \* sizeof(Node \*));

return minHeap;

}

void heapify(MinHeap \*minHeap, int index)

{

int smallest = index;

int left = 2 \* index + 1;

int right = 2 \* index + 2;

if (left < minHeap->size && minHeap->array[left]->frequency < minHeap->array[smallest]->frequency)

{

smallest = left;

}

if (right < minHeap->size && minHeap->array[right]->frequency < minHeap->array[smallest]->frequency)

{

smallest = right;

}

if (smallest != index)

{

swap(&minHeap->array[smallest], &minHeap->array[index]);

heapify(minHeap, smallest);

}

}

Node \*extractMin(MinHeap \*minHeap)

{

Node \*temp = minHeap->array[0];

minHeap->array[0] = minHeap->array[minHeap->size - 1];

minHeap->size--;

heapify(minHeap, 0);

return temp;

}

void insertMinHeap(MinHeap \*minHeap, Node \*node)

{

minHeap->size++;

int i = minHeap->size - 1;

while (i && node->frequency < minHeap->array[(i - 1) / 2]->frequency)

{

minHeap->array[i] = minHeap->array[(i - 1) / 2];

i = (i - 1) / 2;

}

minHeap->array[i] = node;

}

MinHeap \*buildMinHeap(SYMBOL symbols[], int size)

{

MinHeap \*minHeap = createMinHeap(size);

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

{

minHeap->array[i] = createNode(symbols[i].alphabet, symbols[i].frequency);

}

minHeap->size = size;

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

{

heapify(minHeap, i);

}

return minHeap;

}

Node \*buildHuffmanTree(SYMBOL symbols[], int size)

{

Node \*left, \*right, \*top;

MinHeap \*minHeap = buildMinHeap(symbols, size);

while (minHeap->size != 1)

{

left = extractMin(minHeap);

right = extractMin(minHeap);

top = createNode('$', left->frequency + right->frequency);

top->left = left;

top->right = right;

insertMinHeap(minHeap, top);

}

return extractMin(minHeap);

}

void inorderTraversal(Node \*root)

{

if (root == NULL)

return;

inorderTraversal(root->left);

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

{

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

}

inorderTraversal(root->right);

}

int main()

{

printf("Developed by Soudeep Ghoshal | 2205421\n");

int n;

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

scanf("%d", &n);

SYMBOL symbols[n];

printf("Enter the alphabets: ");

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

{

scanf(" %c", &symbols[i].alphabet);

}

printf("Enter its frequencies: ");

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

{

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

}

Node \*root = buildHuffmanTree(symbols, n);

printf("In-order traversal of the tree (Huffman): ");

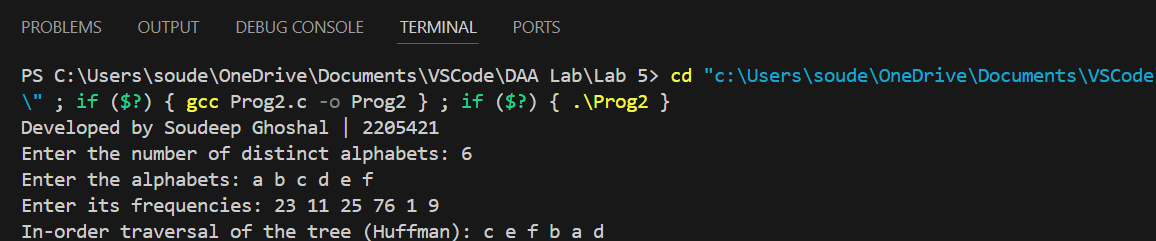
inorderTraversal(root);

printf("\n");

return 0;

}

**Output**

****

***Lab 6***

**Question 6.1**

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.

**Source Code**

#include <stdio.h>

#include <limits.h>

int n = 0;

void print\_optimal\_parens(int s[n][n], int i, int j) {

if (i == j)

printf("A%d ", i);

else {

printf("( ");

print\_optimal\_parens(s, i, s[i][j]);

print\_optimal\_parens(s, s[i][j] + 1, j);

printf(") ");

}

}

int matrix\_chain\_order(int \*p, int n) {

int m[n+1][n+1];

int s[n][n];

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

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

m[i][j] = 0;

if (i <= n && j <= n)

s[i][j] = 0;

}

}

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

m[i][i] = 0;

for (int l = 2; l <= n; l++) {

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

int j = i + l - 1;

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

for (int k = i; k <= j - 1; k++) {

int 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;

}

}

}

}

printf("M-table:\n");

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

for (int j = 1; j < n; j++) {

printf("%d\t", m[i][j]);

}

printf("\n");

}

printf("S-table:\n");

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

for (int j = 1; j < n; j++) {

printf("%d\t", s[i][j]);

}

printf("\n");

}

printf("Optimal parenthesization: ");

print\_optimal\_parens(s, 1, n);

printf("\n");

return m[1][n - 1];

}

int main() {

printf("Enter size of array: ");

scanf("%d", &n);

int p[n];

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

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

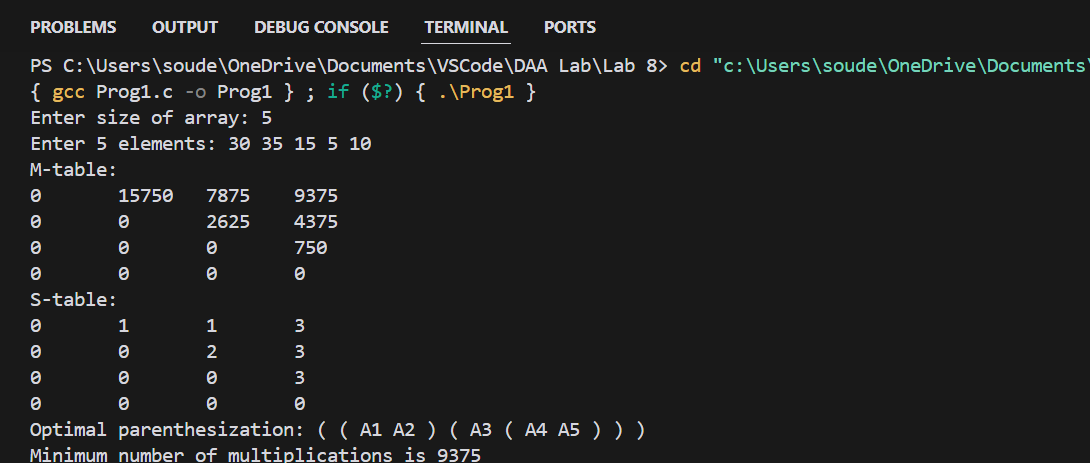
scanf("%d", &p[i]);

}

printf("Minimum number of multiplications is %d\n", matrix\_chain\_order(p, n));

return 0;

}

**Output**

***Lab 7***

**Question 7.1**

Write a program to find out the Longest Common Subsequence of two given strings. Calculate length of the LCS.

**Source Code**

#include <stdio.h>

#include <string.h>

void print\_LCS(int m, int n, int B[m+1][n+1], char \*X, int i, int j);

int LCS\_Length(char \*X, char \*Y, int m, int n) {

int B[m+1][n+1];

int C[m+1][n+1];

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

C[i][0] = 0;

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

C[0][j] = 0;

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

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

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

B[i][j] = 1;

} else if (C[i-1][j] >= C[i][j-1]) {

C[i][j] = C[i-1][j];

B[i][j] = 2;

} else {

C[i][j] = C[i][j-1];

B[i][j] = 3;

}

}

}

// Print the LCS

printf("LCS: ");

print\_LCS(m, n, B, X, m, n);

printf("\n");

return C[m][n];

}

void print\_LCS(int m, int n, int B[m+1][n+1], char \*X, int i, int j) {

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

return;

if (B[i][j] == 1) {

print\_LCS(m, n, B, X, i-1, j-1);

printf("%c", X[i-1]);

} else if (B[i][j] == 2) {

print\_LCS(m, n, B, X, i-1, j);

} else {

print\_LCS(m, n, B, X, i, j-1);

}

}

int main() {

char X[50], Y[50];

printf("Enter first String: ");

scanf("%s", X);

printf("Enter second String: ");

scanf("%s", Y);

int m = strlen(X);

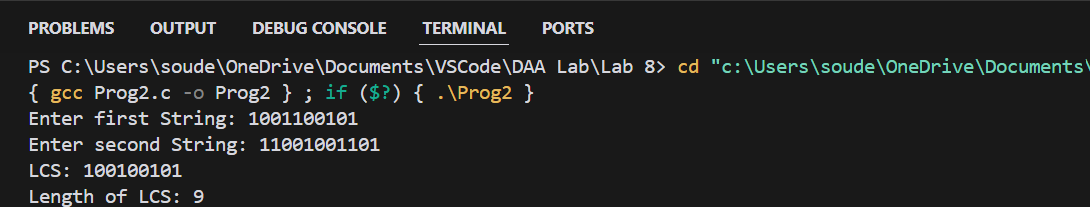
int n = strlen(Y);

printf("Length of LCS: %d\n", LCS\_Length(X, Y, m, n));

return 0;

}

**Output**

****

**Question 7.2**

Given a directed weighted graph G (V, E) where weight indicates distance. Vertices will be numbered consecutively from 1 to n (user input), and edges will have varying distances or lengths.

● Determine the length of the shortest path between every pair of vertices using Floyd-Warshall’s algorithm.

● Display the intermediate vertices on the shortest-path from the given pair of vertices (u,v).

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define INF 99999

#define MAX 100

void floydWarshall(int dist[][MAX], int next[][MAX], int n) {

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

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

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

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

dist[i][j] = dist[i][k] + dist[k][j];

next[i][j] = next[i][k];

}

}

}

}

}

void printPath(int next[][MAX], int u, int v) {

if (next[u][v] == -1) {

printf("No path");

return;

}

printf("%d", u + 1);

while (u != v) {

u = next[u][v];

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

}

printf("\n");

}

int main() {

int n;

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

if (!file) {

printf("Error: File not found.\n");

return 1;

}

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

scanf("%d", &n);

int dist[MAX][MAX], next[MAX][MAX];

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

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

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

if (i != j && dist[i][j] == 0)

dist[i][j] = INF;

if (dist[i][j] != INF)

next[i][j] = j;

else

next[i][j] = -1;

}

}

fclose(file);

floydWarshall(dist, next, n);

printf("\nShortest Path Weight Matrix:\n");

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

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

if (dist[i][j] == INF)

printf("INF\t");

else

printf("%d\t", dist[i][j]);

}

printf("\n");

}

int u, v;

printf("Enter the source and destination vertices: ");

scanf("%d %d", &u, &v);

u--; v--;

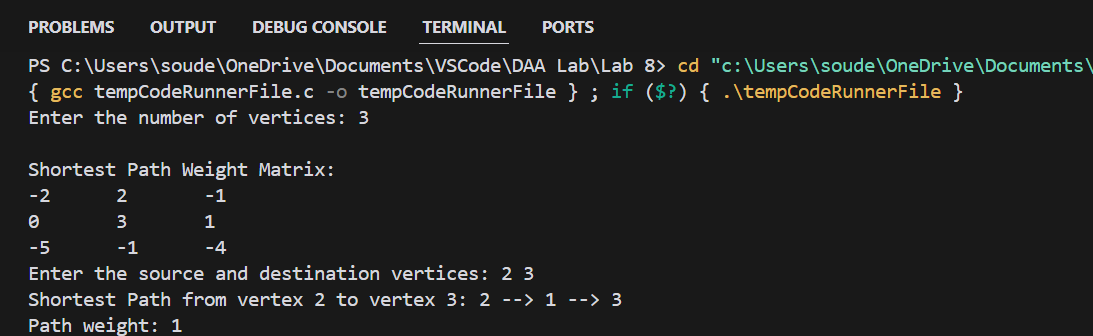
printf("Shortest Path from vertex %d to vertex %d: ", u + 1, v + 1);

printPath(next, u, v);

printf("Path weight: %d\n", dist[u][v]);

return 0;

}

**Out****put**

***Lab 8***

**Question 8.1**

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.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX 100

#define INF 99999

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

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

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

if (visited[v] == 0 && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

}

return min\_index;

}

void printPath(int parent[], int j) {

if (parent[j] == -1)

return;

printPath(parent, parent[j]);

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

}

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

int dist[MAX];

int visited[MAX] = {0};

int parent[MAX];

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

dist[i] = INF;

parent[i] = -1;

}

dist[src] = 0;

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

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

visited[u] = 1;

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

if (!visited[v] && graph[u][v] && dist[u] != INF &&

dist[u] + graph[u][v] < dist[v]) {

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

parent[v] = u;

}

}

}

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

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

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

if (dist[i] == INF)

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

else {

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

printPath(parent, i);

printf("\n");

}

}

}

int main() {

int n;

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

if (!file) {

printf("Error: File not found.\n");

return 1;

}

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

scanf("%d", &n);

int graph[MAX][MAX];

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

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

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

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

graph[i][j] = INF;

}

}

fclose(file);

int src;

printf("Enter the source vertex: ");

scanf("%d", &src);

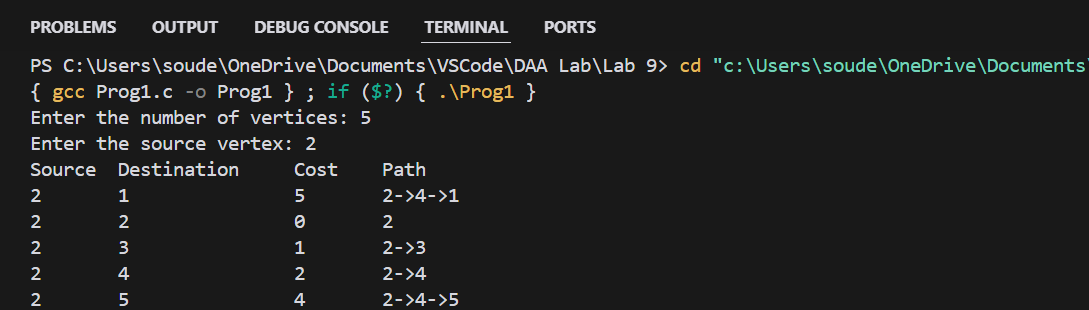
src--;

dijkstra(graph, n, src);

return 0;

}

**Output**

****

***Lab 9***

**Question 9.1**

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

**Source Code**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#include <stdlib.h>

#define INF INT\_MAX

int minKey(int key[], bool mstSet[], int n) {

int min = INF, min\_index;

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

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

void printMST(int parent[], int n, int \*\*graph) {

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

int totalCost = 0;

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

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

mstMatrix[i] = (int \*)malloc(n \* sizeof(int));

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

mstMatrix[i][j] = 0;

}

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

int u = parent[i];

int v = i;

mstMatrix[u][v] = mstMatrix[v][u] = graph[u][v];

totalCost += graph[u][v];

}

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

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

if (mstMatrix[i][j] == INF)

printf("INF ");

else

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

}

printf("\n");

}

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

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

free(mstMatrix[i]);

free(mstMatrix);

}

void primMST(int \*\*graph, int n) {

int parent[n];

int key[n];

bool mstSet[n];

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

key[i] = INF;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

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

int u = minKey(key, mstSet, n);

mstSet[u] = true;

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

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

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

}

printMST(parent, n, graph);

}

int main() {

int n;

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

scanf("%d", &n);

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

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

graph[i] = (int \*)malloc(n \* sizeof(int));

}

FILE \*file = fopen("input.dat", "r");

if (file == NULL) {

printf("Error: Could not open file\n");

return 1;

}

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;

}

}

fclose(file);

primMST(graph, n);

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

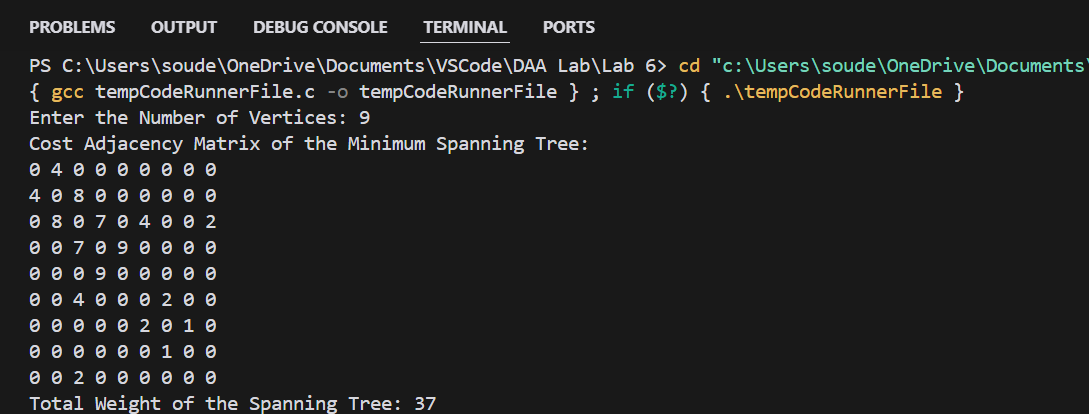
free(graph[i]);

}

free(graph);

return 0;

}

**O****utput**

***A Few LeetCode Questions I have solved***

***Question 1 [Hard]***

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

***Source Code***

class Solution {

public double findMedianSortedArrays(int[] nums1, int[] nums2) {

int merged[] = new int[nums1.length + nums2.length];

int k = -1;

int i, j;

for (i = 0, j = 0; i < nums1.length && j < nums2.length;) {

if (nums1[i] < nums2[j])

merged[++k] = nums1[i++];

else merged[++k] = nums2[j++];

}

while (i < nums1.length)

merged[++k] = nums1[i++];

while (j < nums2.length)

merged[++k] = nums2[j++];

double median = 0;

if ((k + 1) % 2 == 0)

median = (merged[k / 2] + merged[(k / 2) + 1]) / 2.0;

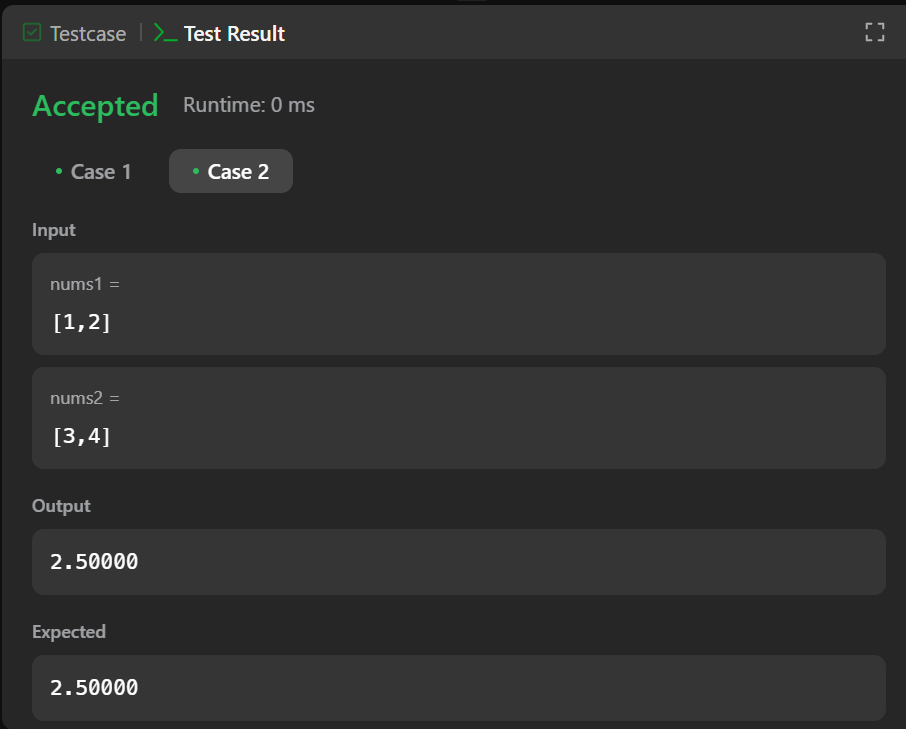
else median = merged[(k + 1) / 2];

return(median);

}

}

***Output***

****

***Question 2 [Hard]***

Given a string s and a dictionary of strings wordDict, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in **any order**.

**Note** that the same word in the dictionary may be reused multiple times in the segmentation.

***Source Code***

class Solution {

public List<String> wordBreak(String s, List<String> wordDict) {

HashSet<String> hs = new HashSet<>(wordDict);

return wordBreakHelper(s, 0, hs);

}

private List<String> wordBreakHelper(String s, int start, HashSet<String> dict) {

List<String> validSubstr = new ArrayList<>();

if (start == s.length())

validSubstr.add("");

for (int end = start + 1; end <= s.length(); end++) {

String prefix = s.substring(start, end);

if (dict.contains(prefix)) {

List<String> suffixes = wordBreakHelper(s, end, dict);

for (String suffix : suffixes) {

validSubstr.add(prefix + (suffix.equals("") ? "" : " ") + suffix);

}

}

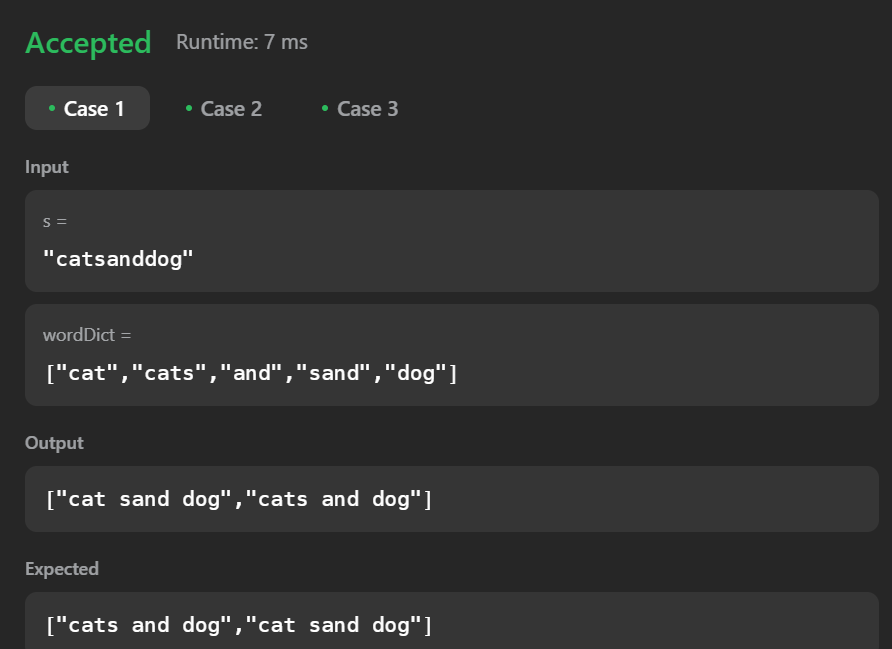
}

return validSubstr;

}

}

***Output***

****

***Question 3 [Medium] [DAA Concept: Bitmasking]***

Given an integer array nums of unique elements, return all possible subsets (the power set).

The solution set must not contain duplicate subsets. Return the solution in any order.

***Source Code***

class Solution {

public List<List<Integer>> subsets(int[] nums) {

List<List<Integer>>powerSet = new ArrayList<>();

for (int i = 0; i < (1 << nums.length); i++) {

List<Integer> subSet = new ArrayList<>();

for (int j = 0; j < nums.length; j++) {

if ((i & (1 << j)) != 0)

subSet.add(nums[j]);

}

powerSet.add(subSet);

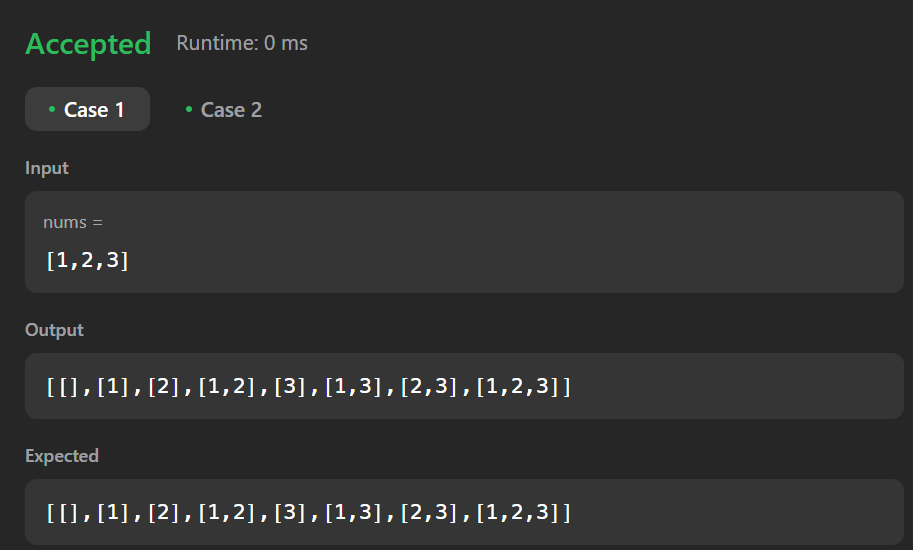
}

return powerSet;

}

}

***Output***

****

***Question 4 [Hard] [DAA Concept: Greedy Approach]***

There are n workers. You are given two integer arrays quality and wage where quality[i] is the quality of the ith worker and wage[i] is the minimum wage expectation for the ith worker.

We want to hire exactly k workers to form a **paid group**. To hire a group of k workers, we must pay them according to the following rules:

1. Every worker in the paid group must be paid at least their minimum wage expectation.
2. In the group, each worker's pay must be directly proportional to their quality. This means if a worker’s quality is double that of another worker in the group, then they must be paid twice as much as the other worker.

Given the integer k, return the least amount of money needed to form a paid group satisfying the above conditions. Answers within 10-5 of the actual answer will be accepted.

***Source Code***

class Solution {

public double mincostToHireWorkers(int[] quality, int[] wage, int k) {

int n = quality.length;

double[][] workers = new double[n][2];

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

workers[i][0] = (double) wage[i] / quality[i]; // ratio

workers[i][1] = (double) quality[i]; // quality

}

Arrays.sort(workers, (a, b) -> Double.compare(a[0], b[0]));

PriorityQueue<Double> maxHeap = new PriorityQueue<>((a, b) -> Double.compare(b, a));

double totalQuality = 0;

double minCost = Double.MAX\_VALUE;

for (double[] worker : workers) {

double ratio = worker[0];

double q = worker[1];

maxHeap.offer(q);

totalQuality += q;

if (maxHeap.size() > k) {

totalQuality -= maxHeap.poll();

}

if (maxHeap.size() == k) {

minCost = Math.min(minCost, totalQuality \* ratio);

}

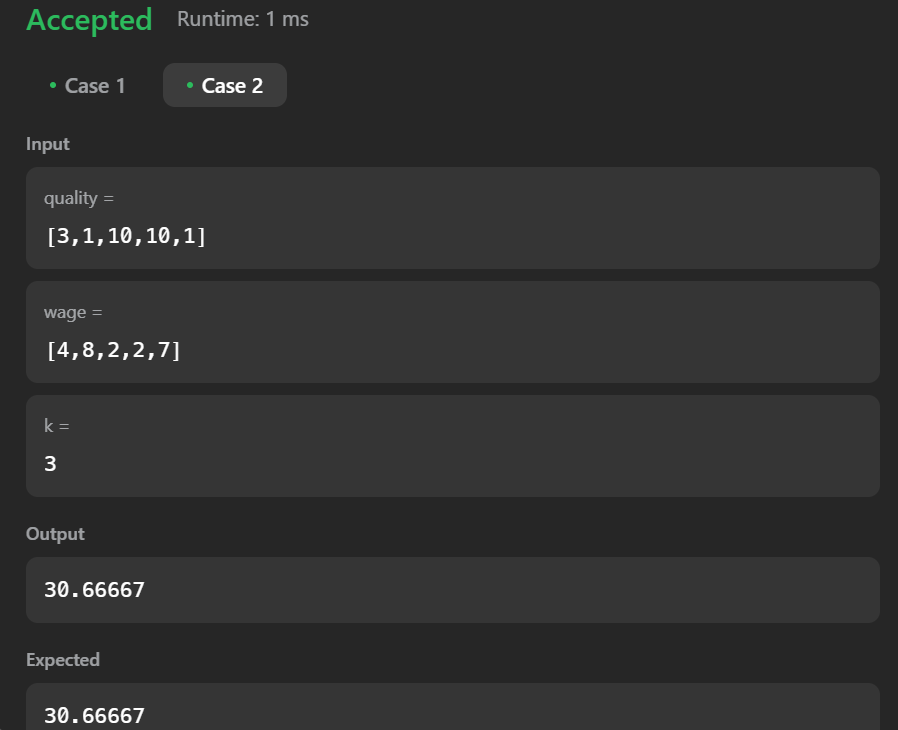
}

return minCost;

}

}

***Output***

****

***Question 5 [Hard]***

There exists an **undirected** tree with n nodes numbered 0 to n - 1. You are given a **0-indexed** 2D integer array edges of length n - 1, where edges[i] = [ui, vi] indicates that there is an edge between nodes ui and vi in the tree. You are also given a **positive** integer k, and a **0-indexed** array of **non-negative** integers nums of length n, where nums[i] represents the **value** of the node numbered i.

Alice wants the sum of values of tree nodes to be **maximum**, for which Alice can perform the following operation **any** number of times (**including zero**) on the tree:

* Choose any edge [u, v] connecting the nodes u and v, and update their values as follows:
  + nums[u] = nums[u] XOR k
  + nums[v] = nums[v] XOR k

Return the **maximum** possible **sum** of the **values** Alice can achieve by performing the operation **any** number of times.

***Source Code***

class Solution {

public long maximumValueSum(int[] nums, int k, int[][] edges) {

long maxSum = 0;

for (int i: nums)

maxSum += i;

long maxDiff = 0;

long maxNegDiff = Long.MAX\_VALUE;

int posCount = 0;

for (int i: nums) {

long diff = (i ^ k) - i;

if (diff > 0) {

maxDiff += diff;

posCount++;

}

maxNegDiff = Math.min(maxNegDiff, Math.abs(diff));

}

maxSum += maxDiff;

if (posCount % 2 == 1)

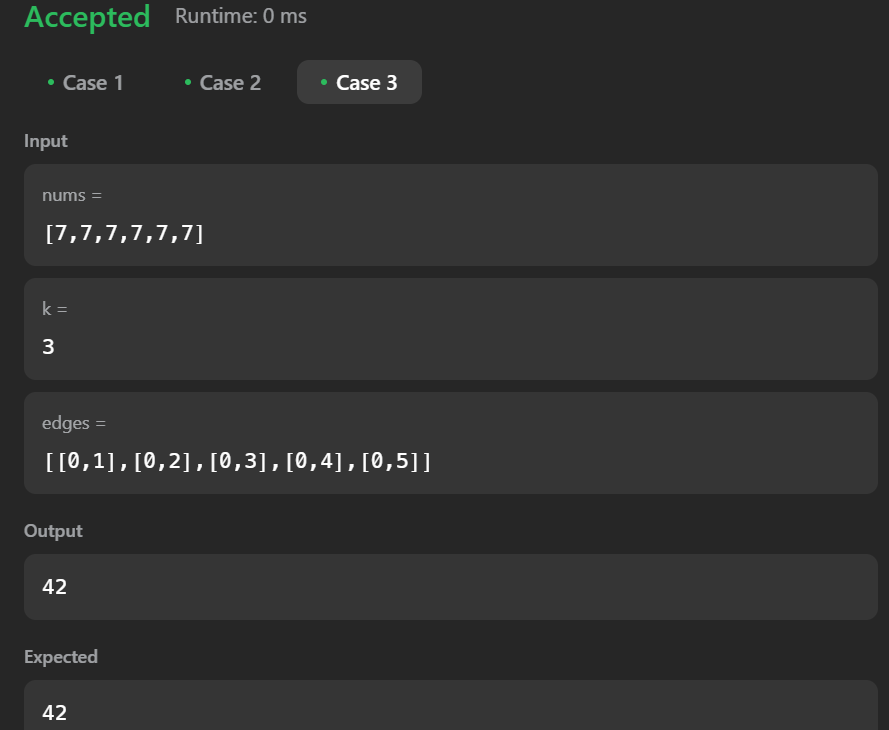
maxSum -= maxNegDiff;

return maxSum;

}

}

***Output***

****