**Student Name:-Ritam Hudait**

**Student Roll No.:- 1905797**

**Algorithm Lab. Class Assignment-7**

**CSE Group 1**

**Date: - 27th August 2021**

1. **Write a program to sort a given set of elements using the Merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted, and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

**Program**

**#include <stdio.h>**

**#include <time.h>**

**#include <stdlib.h>**

**void merge(int arr[], int l, int m, int r)**

**{**

**int i, j, k;**

**int n1 = m - l + 1;**

**int n2 = r - m;**

**int L[n1], R[n2];**

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

**L[i] = arr[l + i];**

**}**

**for(int i=0;i<n2;i++) {**

**R[i] = arr[m + 1 + i];**

**}**

**i = 0;**

**j = 0;**

**k = l;**

**while (i < n1 && j < n2) {**

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

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

**i++;**

**}**

**else {**

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

**j++;**

**}**

**k++;**

**}**

**while (i < n1) {**

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

**i++;**

**k++;**

**}**

**while (j < n2) {**

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

**j++;**

**k++;**

**}**

**}**

**void mergeSort(int arr[], int l, int r) {**

**if (l < r) {**

**int m = l + (r - l) / 2;**

**mergeSort(arr, l, m);**

**mergeSort(arr, m + 1, r);**

**merge(arr, l, m, r);**

**}**

**}**

**int main() {**

**printf("n\t\t|\tbest\t\tavg\t\t\tworst\n\_\_|\_\_\_\_\n");**

**int sizes;**

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

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

**int n;**

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

**printf("%d\t|\t", n);**

**int arr[n];**

**time\_t start, end;**

**double time;**

**// Best**

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

**arr[j] = j + 1;**

**}**

**start = clock();**

**mergeSort(arr, 0, n - 1);**

**end = clock();**

**time = (end - start) \* 1.0 / CLOCKS\_PER\_SEC;**

**printf("%f\t", time);**

**// Avg**

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

**arr[j] = rand() % 10000;**

**}**

**start = clock();**

**mergeSort(arr, 0, n - 1);**

**end = clock();**

**time = (end - start) \* 1.0 / CLOCKS\_PER\_SEC;**

**printf("%f\t", time);**

**// Worst**

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

**arr[j] = rand() % 10000;**

**}**

**start = clock();**

**mergeSort(arr, 0, n - 1);**

**end = clock();**

**time = (end - start) \* 1.0 / CLOCKS\_PER\_SEC;**

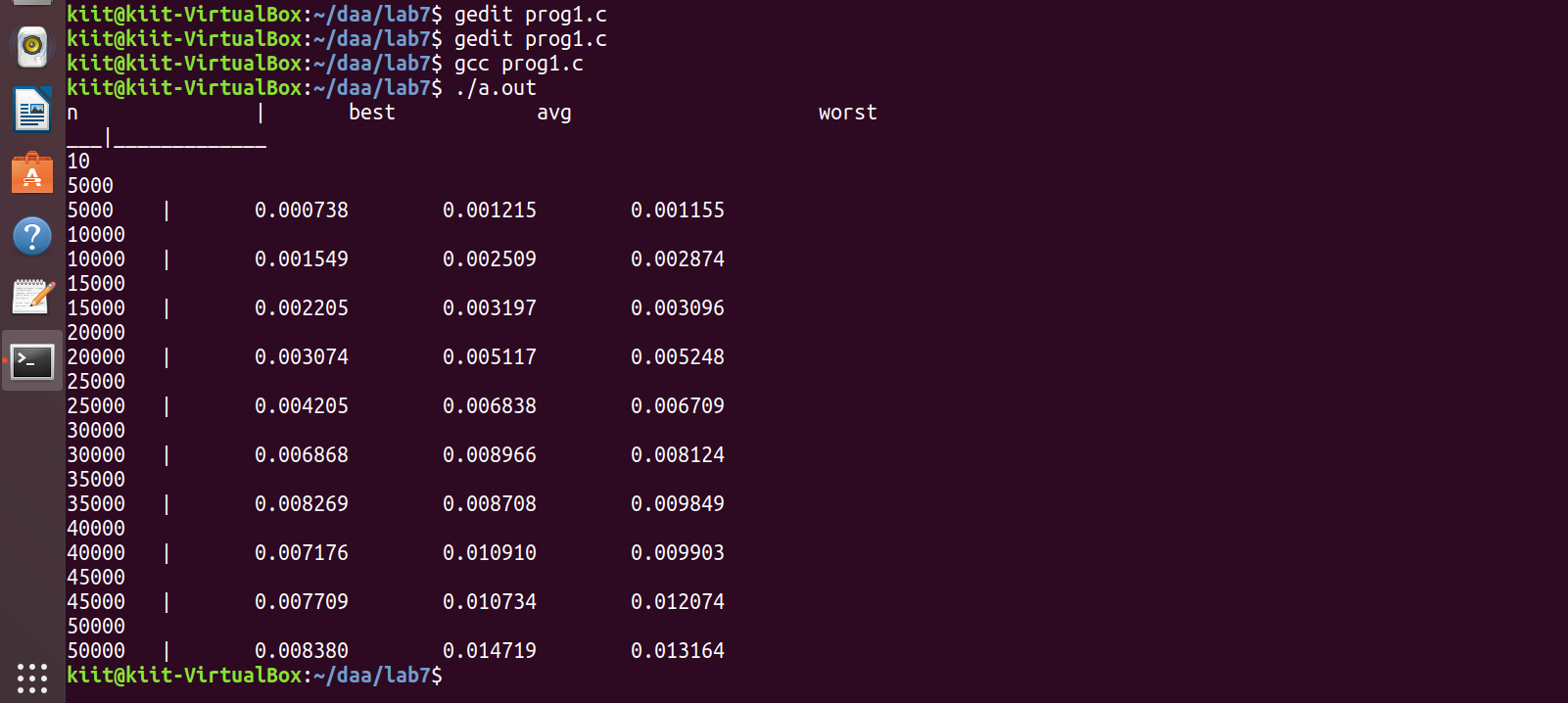
**printf("%f\n", time);**

**}**

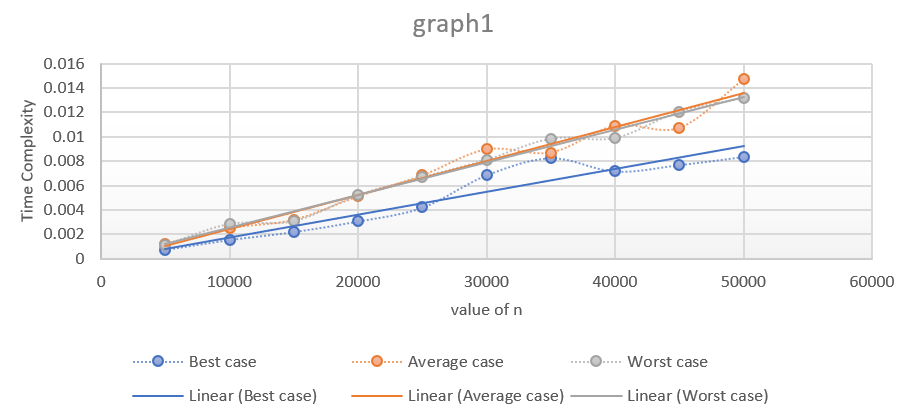
**return 0;**

**}**

**Output**

****

**Graph**

****

1. **Write a C program to implement the Tower of Hanoi problem using the Divide and Conqueror approach.**

**Program**

**#include <stdio.h>**

**#include <time.h>**

**#include <stdlib.h>**

**void towerOfHanoi(int n, char s, char h, char d) {**

**if (n == 1) {**

**printf("Move %d from %c to %c\n", n, s, d);**

**return;**

**}**

**towerOfHanoi(n - 1, s, d, h);**

**printf("Move %d from %c to %c\n", n, s, d);**

**towerOfHanoi(n - 1, h, s, d);**

**}**

**int main() {**

**int n;**

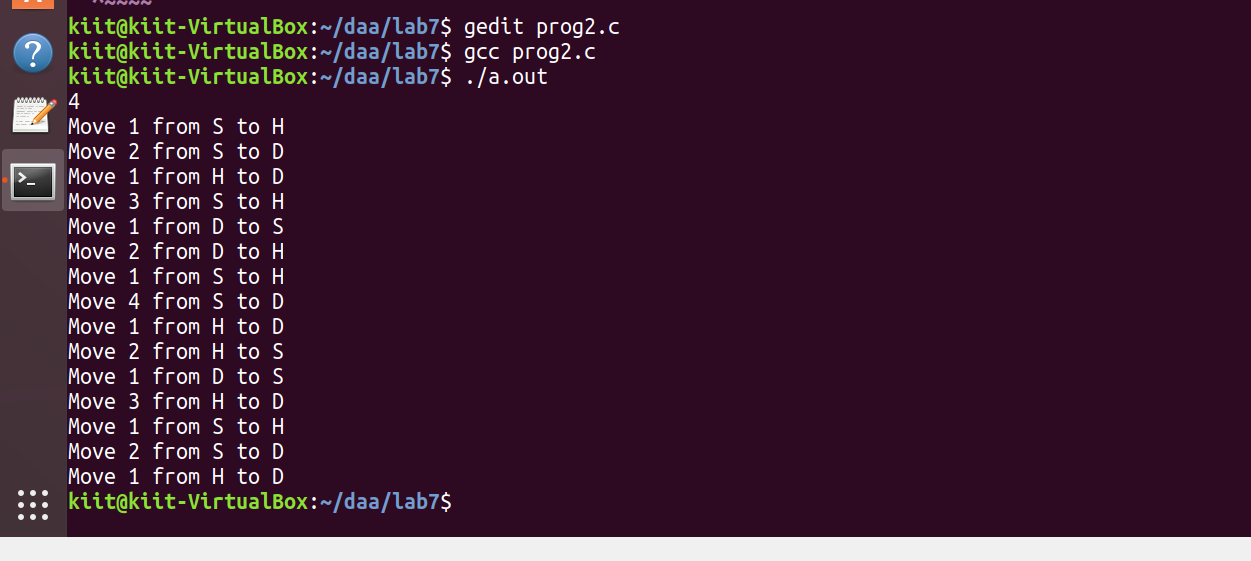
**scanf("%d",&n);**

**towerOfHanoi(n, 'S', 'H', 'D');**

**return 0;**

**}**

**Output**

****