3) Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. 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.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to merge two subarrays arr[l..m] and arr[m+1..r]

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

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

// Create temporary arrays

int L[n1], R[n2];

// Copy data to temporary arrays L[] and R[]

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

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

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

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

// Merge the temporary arrays back into arr[l..r]

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

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

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

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

// Copy the remaining elements of L[], if any

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// Copy the remaining elements of R[], if any

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Main function of Merge Sort

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

if (l < r) {

// Same as (l+r)/2, but avoids overflow for large l and r

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

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted halves

merge(arr, l, m, r);

}

}

// Function to generate random array of size n

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

srand(time(NULL)); // Seed for random number generator

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

arr[i] = rand();

}

}

int main() {

// Vary n from 5000 to 10000

for (int n = 5000; n <= 10000; n += 1000) {

// Dynamically allocate memory for array

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

// Generate random array

generateRandomArray(arr, n);

// Measure time taken for sorting

clock\_t start = clock();

mergeSort(arr, 0, n - 1);

clock\_t end = clock();

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

// Output n and time taken

printf("n = %d, Time taken: %lf seconds\n", n, time\_taken);

// Free dynamically allocated memory

free(arr);

}

return 0;

}

OUTPUT:

n = 5000, Time taken: 0.000801 seconds

n = 6000, Time taken: 0.000962 seconds

n = 7000, Time taken: 0.001083 seconds

n = 8000, Time taken: 0.001317 seconds

n = 9000, Time taken: 0.001457 seconds

n = 10000, Time taken: 0.001679 seconds