LAB 11 QUICK SORT

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to merge two subarrays into a single sorted array

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

// Create temporary arrays

int L[n1], R[n2];

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

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

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

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

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

// Merge the temporary arrays back into arr[left..right]

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

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++;

}

}

// Function to perform Merge Sort

void mergeSort(int arr[], int left, int right) {

if (left < right) {

// Find the middle point

int mid = left + (right - left) / 2;

// Recursively sort the first and second halves

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

// Merge the sorted halves

merge(arr, left, mid, right);

}

}

// Function to generate a random array of integers

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

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

arr[i] = rand() % 10000; // random integers between 0 and 9999

}

}

// Function to plot the graph using gnuplot

void plotGraph(int n\_values[], double time\_taken[], int num\_tests) {

// Open a pipe to gnuplot

FILE \*gnuplot = popen("gnuplot -persistent", "w");

if (gnuplot == NULL) {

fprintf(stderr, "Error opening gnuplot\n");

return;

}

// Send commands to gnuplot

fprintf(gnuplot, "set title 'Time Complexity of Merge Sort'\n");

fprintf(gnuplot, "set xlabel 'Number of Elements (n)'\n");

fprintf(gnuplot, "set ylabel 'Time Taken (seconds)'\n");

fprintf(gnuplot, "plot '-' with linespoints title 'Time vs n'\n");

// Send the data points to gnuplot

for (int i = 0; i < num\_tests; i++) {

fprintf(gnuplot, "%d %lf\n", n\_values[i], time\_taken[i]);

}

// End the plot

fprintf(gnuplot, "e\n");

fclose(gnuplot);

}

int main() {

// Seed the random number generator

srand(time(NULL));

// Set the values for n (can be modified or read from a file)

int n\_values[] = {5000, 10000, 15000, 20000, 25000}; // example values of n

int num\_tests = sizeof(n\_values) / sizeof(n\_values[0]);

double time\_taken[num\_tests]; // Array to store time taken for each n

// Loop over different values of n

for (int test = 0; test < num\_tests; test++) {

int n = n\_values[test];

// Generate an array of random integers of size n

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

if (arr == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

generateRandomArray(arr, n);

// Record the start time

clock\_t start\_time = clock();

// Call the mergeSort function

mergeSort(arr, 0, n - 1);

// Record the end time

clock\_t end\_time = clock();

// Calculate the time taken (in seconds)

time\_taken[test] = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC;

// Output the results

printf("Time taken to sort an array of size %d: %f seconds\n", n, time\_taken[test]);

// Free the dynamically allocated memory

free(arr);

}

// Plot the graph using gnuplot

plotGraph(n\_values, time\_taken, num\_tests);

return 0;

}