Lab 10: Quick sort

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to perform QuickSort

void quickSort(int arr[], int low, int high) {

if (low < high) {

// Partition the array

int pi = partition(arr, low, high);

// Recursively sort the subarrays

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Partition function to place the pivot element in the correct position

int partition(int arr[], int low, int high) {

int pivot = arr[high]; // pivot element

int i = low - 1; // Index of smaller element

for (int j = low; j < high; j++) {

// If the current element is smaller than or equal to the pivot

if (arr[j] <= pivot) {

i++;

// Swap arr[i] and arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// Swap arr[i + 1] and arr[high] (pivot)

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1; // Return the partitioning index

}

// 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 read integers from a file

int readArrayFromFile(const char \*filename, int \*\*arr) {

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

if (file == NULL) {

printf("File could not be opened!\n");

return -1;

}

int size = 0;

while (fscanf(file, "%d", &(\*arr)[size]) == 1) {

size++;

\*arr = realloc(\*arr, (size + 1) \* sizeof(int)); // dynamically expand array

}

fclose(file);

return size;

}

// 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 QuickSort'\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];

// Dynamically allocate memory for array

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

if (arr == NULL) {

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

return 1;

}

// Generate an array of random integers or read from a file

generateRandomArray(arr, n);

// Record the start time

clock\_t start\_time = clock();

// Call the quickSort function

quickSort(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;

}