2) Design and implement C/C++ Program to sort a given set of n integer elements using Quick 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 swap two elements

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Partition function for Quick Sort

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

int pivot = arr[high];

int i = (low - 1);

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

if (arr[j] < pivot) {

i++;

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

}

}

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

return (i + 1);

}

// Quick Sort function

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

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// 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();

quickSort(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: 0.000460 seconds

n = 6000, Time taken: 0.000554 seconds

n = 7000, Time taken: 0.000634 seconds

n = 8000, Time taken: 0.000732 seconds

n = 9000, Time taken: 0.000825 seconds

n = 10000, Time taken: 0.000987 seconds