Name: ANYA AGARWAL

Section: B (13)

Roll no: 2021122

I. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array.

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

int MergeSort(vector<int>& arr, int l, int r, int& comparisons);

int Merge(vector<int>& arr, int l, int m, int r, int& comparisons);

int main() {

ifstream inputFile("inputfile.txt");

if (!inputFile) {

cerr << "Error opening input file." << endl;

return 1;

}

ofstream outputFile("outputfile.txt");

if (!outputFile) {

cerr << "Error opening output file." << endl;

return 1;

}

int T;

inputFile >> T;

while (T--) {

int n, i, comparisons = 0;

inputFile >> n;

vector<int> arr(n);

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

inputFile >> arr[i];

}

int inversions = MergeSort(arr, 0, n - 1, comparisons);

// Output

outputFile << "Sorted array: ";

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

outputFile << arr[i] << " ";

}

outputFile << "\nTotal number of comparisons: " << comparisons << endl;

outputFile << "Total number of inversions: " << inversions << endl;

}

return 0;

}

int MergeSort(vector<int>& arr, int l, int r, int& comparisons) {

int inversions = 0;

if (l < r) {

int m = (l + r) / 2;

inversions += MergeSort(arr, l, m, comparisons);

inversions += MergeSort(arr, m + 1, r, comparisons);

inversions += Merge(arr, l, m, r, comparisons);

}

return inversions;

}

int Merge(vector<int>& arr, int l, int m, int r, int& comparisons) {

int i, j, k, inversions = 0;

int n1 = m - l + 1;

int n2 = r - m;

vector<int> L(n1);

vector<int> R(n2);

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

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

}

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

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

}

i = 0, j = 0, k = l;

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

comparisons++;

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

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

inversions += (n1 - i);

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

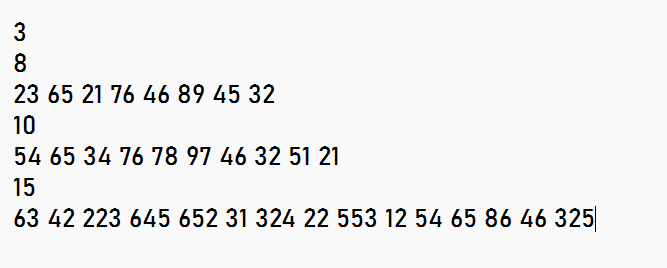
k++;

}

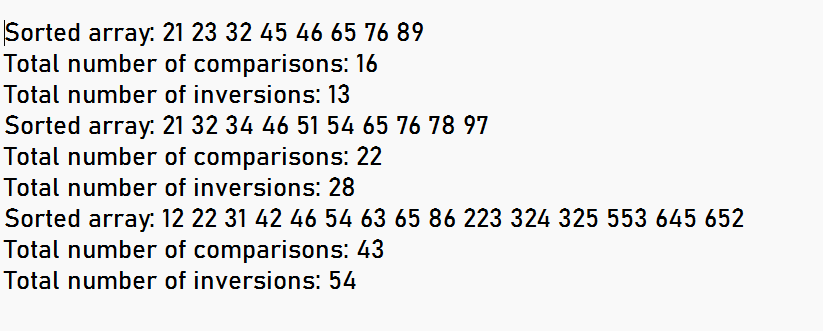
return inversions;

}

INPUT FILE



OUTPUT FILE



Name: ANYA AGARWAL

Section: B (13)

Roll no: 2021122

II. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <ctime>

using namespace std;

void QuickSort(int arr[], int l, int r, int\* comparisons, int\* swaps);

int Partition(int arr[], int l, int r, int\* comparisons, int\* swaps);

void Swap(int\* a, int\* b);

int main() {

ifstream inputFile("inputfile.txt");

if (!inputFile) {

cerr << "Error opening input file." << endl;

return 1;

}

ofstream outputFile("outputfile.txt");

if (!outputFile) {

cerr << "Error opening output file." << endl;

return 1;

}

int T;

inputFile >> T;

for (int t = 0; t < T; ++t) {

int n, comparisons = 0, swaps = 0;

inputFile >> n;

int\* arr = new int[n];

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

inputFile >> arr[i];

}

QuickSort(arr, 0, n - 1, &comparisons, &swaps);

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

outputFile << arr[i] << " ";

}

outputFile << endl;

outputFile << "Number of comparisons: " << comparisons << endl;

outputFile << "Number of swaps: " << swaps << endl;

delete[] arr;

}

inputFile.close();

outputFile.close();

return 0;

}

void QuickSort(int arr[], int l, int r, int\* comparisons, int\* swaps) {

if (l < r) {

int pivotIndex = Partition(arr, l, r, comparisons, swaps);

QuickSort(arr, l, pivotIndex - 1, comparisons, swaps);

QuickSort(arr, pivotIndex + 1, r, comparisons, swaps);

}

}

int Partition(int arr[], int l, int r, int\* comparisons, int\* swaps) {

srand(time(0));

int pivotIndex = l + rand() % (r - l + 1);

int pivot = arr[pivotIndex];

int i = l - 1;

Swap(&arr[pivotIndex], &arr[r]);

(\*swaps)++;

for (int j = l; j < r; j++) {

(\*comparisons)++;

if (arr[j] <= pivot) {

i++;

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

(\*swaps)++;

}

}

Swap(&arr[i + 1], &arr[r]);

(\*swaps)++;

return i + 1;

}

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

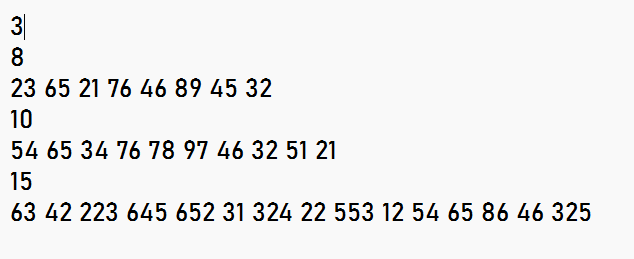
int temp = \*a;

\*a = \*b;

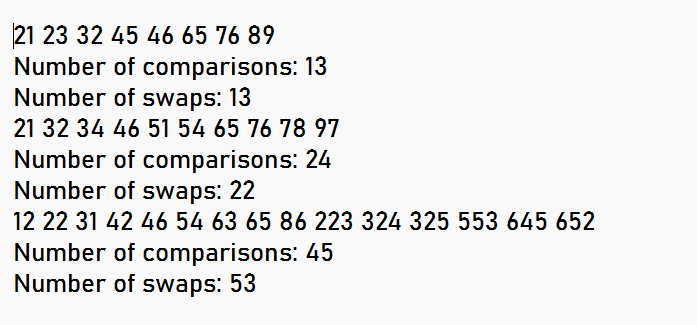
\*b = temp;

}

INPUT FILE



OUTPUT FILE



Name: ANYA AGARWAL

Section: B (13)

Roll no: 2021122

III. Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

for (int i = 0; i < n - 1; ++i) {

int min\_idx = i;

for (int j = i + 1; j < n; ++j) {

if (arr[j] < arr[min\_idx]) {

min\_idx = j;

}

}

if (min\_idx != i) {

swap(&arr[i], &arr[min\_idx]);

}

}

}

int main() {

ifstream inputFile("inputfile.txt");

if (!inputFile) {

cerr << "Error opening input file." << endl;

return 1;

}

ofstream outputFile("outputfile.txt");

if (!outputFile) {

cerr << "Error opening output file." << endl;

return 1;

}

int T;

inputFile >> T;

for (int t = 0; t < T; ++t) {

int n, k;

inputFile >> n;

int\* a = new int[n];

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

inputFile >> a[i];

}

inputFile >> k;

selectionSort(a, n);

int c = 1;

int i;

for (i = 1; i < n; ++i) {

if (a[i] != a[i - 1]) {

c++;

}

if (c == k) {

break;

}

}

if (i < n) {

outputFile << "For test case " << t + 1 << ", the " << k << "th smallest element is: " << a[i] << endl;

} else {

outputFile << "For test case " << t + 1 << ", not present" << endl;

}

delete[] a;

}

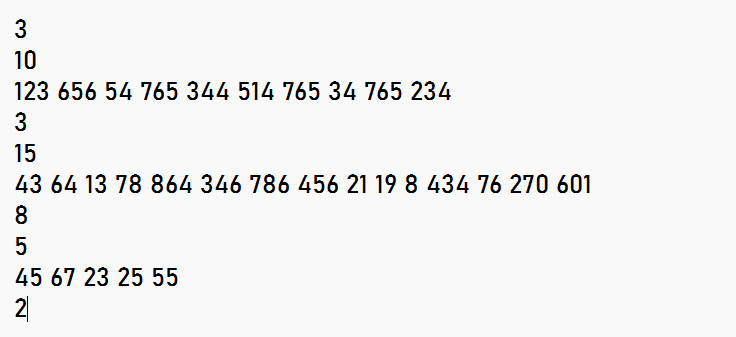
inputFile.close();

outputFile.close();

return 0;

}

INPUT FILE



OUTPUT FILE

