**9923103023 – F1**

**15B17CI371 – Data Structures Lab ODD 2024**

**Week 4-LAB B**

**Practice Lab**

1. Write a program to count the frequency of each element in a given array. Test Case: Input: array = {9, 12, 3, 31, 3, 19, 9, 3}; Output: Unique: {9, 12, 3, 31, 19} Frequency: {2, 1, 3, 1, 1}.

Ans :

#include <iostream>

using namespace std;

const int MAX\_SIZE = 100;

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

int unique[MAX\_SIZE];

int frequency[MAX\_SIZE];

int uniqueCount = 0;

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

bool found = false;

for (int j = 0; j < uniqueCount; ++j) {

if (arr[i] == unique[j]) {

frequency[j]++;

found = true;

break;

}

}

if (!found) {

unique[uniqueCount] = arr[i];

frequency[uniqueCount] = 1;

uniqueCount++;

}

}

cout << "Unique: {";

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

if (i > 0) cout << ", ";

cout << unique[i];

}

cout << "}" << endl;

cout << "Frequency: {";

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

if (i > 0) cout << ", ";

cout << frequency[i];

}

cout << "}" << endl;

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

if (n > MAX\_SIZE) {

cout << "Number of elements exceeds the maximum size of " << MAX\_SIZE << endl;

return 1;

}

int array[MAX\_SIZE];

cout << "Enter the elements: ";

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

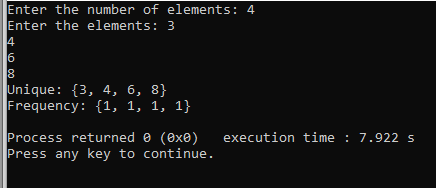
cin >> array[i];

}

countFrequency(array, n);

return 0;

}



2. Given a sorted array of n elements, write a function to search for a given target value using the Jump Search algorithm. The Jump Search algorithm works by dividing the array into blocks of a fixed size, jumping ahead by these block sizes, and then performing a linear search within the block where the target might be present. Test Case: Input: array = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}; size = 10; key = 11; Output: Element found at index 5:

Ans :

#include <iostream>

#include <cmath>

using namespace std;

int jumpSearch(int arr[], int size, int key) {

int step = sqrt(size);

int prev = 0;

while (arr[min(step, size) - 1] < key) {

prev = step;

step += sqrt(size);

if (prev >= size) return -1;

}

while (arr[prev] < key) {

prev++;

if (prev == min(step, size)) return -1;

}

if (arr[prev] == key) return prev;

return -1;

}

int main() {

int size;

cout << "Enter the number of elements: ";

cin >> size;

if (size <= 0) {

cout << "Array size must be positive." << endl;

return 1;

}

int\* array = new int[size];

cout << "Enter the elements (sorted): ";

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

cin >> array[i];

}

int key;

cout << "Enter the key to search: ";

cin >> key;

int index = jumpSearch(array, size, key);

if (index != -1) {

cout << "Element found at index " << index << endl;

} else {

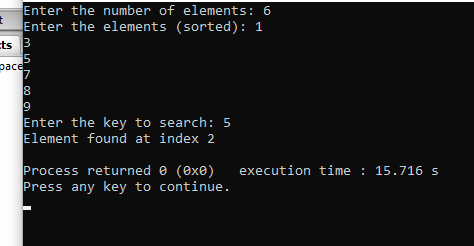
cout << "Element not found" << endl;

}

delete[] array;

return 0;

}



3. Given an array of integers, sort the array according to the frequency of elements. Elements with higher frequency come first. If two elements have the same frequency, they are sorted by their value. Input: arr[] = {4, 5, 6, 5, 4, 3} Output: Pair Found: (4, 4, 5, 5, 3, 6

Ans :

#include <iostream>

using namespace std;

const int MAX\_SIZE = 100;

void countFrequency(int arr[], int n, int unique[], int freq[], int& uniqueCount) {

uniqueCount = 0;

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

bool found = false;

for (int j = 0; j < uniqueCount; ++j) {

if (arr[i] == unique[j]) {

freq[j]++;

found = true;

break;

}

}

if (!found) {

unique[uniqueCount] = arr[i];

freq[uniqueCount] = 1;

uniqueCount++;

}

}

}

void sortByFrequency(int unique[], int freq[], int n) {

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

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

if (freq[i] < freq[j] || (freq[i] == freq[j] && unique[i] > unique[j])) {

swap(freq[i], freq[j]);

swap(unique[i], unique[j]);

}

}

}

}

void sortArrayByFrequency(int input[], int size) {

int freq[MAX\_SIZE];

int unique[MAX\_SIZE];

int uniqueCount;

countFrequency(input, size, unique, freq, uniqueCount);

sortByFrequency(unique, freq, uniqueCount);

cout << "Pair Found: ";

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

for (int j = 0; j < freq[i]; ++j) {

cout << unique[i] << " ";

}

}

cout << endl;

}

int main() {

int size;

cout << "Enter the number of elements: ";

cin >> size;

if (size <= 0 || size > MAX\_SIZE) {

cout << "Invalid size. Size must be positive and less than or equal to " << MAX\_SIZE << endl;

return 1;

}

int array[MAX\_SIZE];

cout << "Enter the elements: ";

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

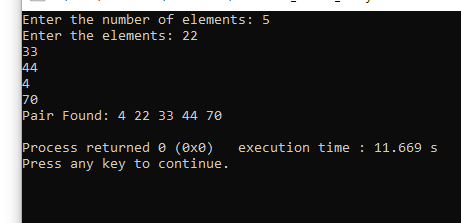
cin >> array[i];

}

sortArrayByFrequency(array, size);

return 0;

}



4. Given an array of integers, return an array depicting the differences between two successive elements sorted in descending order. Input: arr[] = {4, 1, 3, 5, 4, 3} Output: out[] ={3, 2, 2, 1,1} Input: arr[] = {3, 1, 4, 5, 4, 3} Output: out[] = {3,2,1,1,1} Difference array is {2, 3, 1, 1,1}, then it is sorted in descending order

Ans :

#include <iostream>

using namespace std;

int myAbs(int x) {

return (x < 0) ? -x : x;

}

void calculateDifferences(const int arr[], int size, int diff[], int& diffSize) {

if (size <= 1) {

diffSize = 0;

return;

}

diffSize = size - 1;

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

diff[i] = myAbs(arr[i + 1] - arr[i]);

}

}

void sortDescending(int arr[], int size) {

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

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

if (arr[i] < arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

int main() {

int size;

cout << "Enter the number of elements: ";

cin >> size;

if (size <= 1) {

cout << "Array size must be greater than 1." << endl;

return 1;

}

int\* array = new int[size];

cout << "Enter the elements: ";

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

cin >> array[i];

}

int\* diff = new int[size - 1];

int diffSize;

calculateDifferences(array, size, diff, diffSize);

sortDescending(diff, diffSize);

cout << "Output: {";

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

if (i > 0) cout << ", ";

cout << diff[i];

}

cout << "}" << endl;

delete[] array;

delete[] diff;

return 0;

}

