# The Code

class compareArrays {

private:

int Array1Values[600];

int Array2Values[600];

int Size;

vector<int> values;

vector <int> valueMultiplicity;

vector<int> values2;

vector <int> valueMultiplicity2;

int compareArray() {

//compares the values in the arrays and see if they match, if they don't they aren't the same

if (values == values2 && valueMultiplicity == valueMultiplicity2) {

cout << "The Arrays are the same!!!!" << endl;

}

else {

cout << "The Arrays aren't the same..." << endl;

}

return 0;

};

void compareSize() {

int arr1Size = sizeof(Array1Values) / sizeof(Array1Values[0]);

int arr2Size = sizeof(Array2Values) / sizeof(Array2Values[0]);

//Compares the length of both of the vectors. if they aren't the same length they output saying so.

if (arr1Size != arr2Size) {

cout << "The Arrays have differing sizes. " << endl;

cout << endl;

}

else {

cout << "The Arrays have the same size." << endl;

cout << endl;

}

};

void multiplicity1() {

//search algorithm used to take a value in the vector and then push it into two different vectors

//initializes count, a variable to compare index numbers to be searches and initializes vectors.

int count = 1;

int searchNumber = 1;

int\* arrayPtr = Array1Values;

//sets first element to first value in Array1Values vector.

int element = arrayPtr[0];

//while loop runs until the last value of Array1Values

while (searchNumber < Size) {

//compares next number with the value, if same number it increments count and the index.

if (\*(arrayPtr + searchNumber - 1) == \*(arrayPtr + searchNumber)) {

count++;

searchNumber++;

}

else {

// if the value isn't a repeat, it pushes the number to values vector and the count to multiplicty vector

values.push\_back(element);

valueMultiplicity.push\_back(count);

//assigns the index to element then increments to the next number to be searched.

element = \*(arrayPtr + searchNumber);

searchNumber++;

//resets count to 1;

count = 1;

}

}

//pushes the last elements value and multiplicity to the different vectors.

values.push\_back(element);

valueMultiplicity.push\_back(count);

};

void outputMultiplicity() {

//outputs the Array with its values and multiplicity

cout << "Array 1 has the following values and multiplicity:" << endl;

cout << endl;

//for loop steps through each x index for both vectors.

for (int x = 0; x < values.size(); x++) {

cout << values[x] << " has a multiplicity of " << valueMultiplicity[x] << endl;

}

cout << endl;

};

void multiplicity2() {

//search algorithm used to take a value in the vector and then push it into two different vectors

//initializes count, a variable to compare index numbers to be searches and initializes vectors.

int count2 = 1;

int searchNumber2 = 1;

int\* array2Ptr = Array2Values;

//sets first element to first value in Array2Values vector.

int element2 = array2Ptr[0];

//while loop runs until the last value of Array2Values

while (searchNumber2 < Size) {

//compares next number with the value, if same number it increments count and the index.

if (\*(array2Ptr + searchNumber2 - 1) == \*(array2Ptr + searchNumber2)) {

count2++;

searchNumber2++;

}

else {

// if the value isn't a repeat, it pushes the number to values vector and the count to multiplicty vector

values2.push\_back(element2);

valueMultiplicity2.push\_back(count2);

//assigns the index to element then increments to the next number to be searched.

element2 = \*(array2Ptr + searchNumber2);

searchNumber2++;

//resets count to 1;

count2 = 1;

}

}

//pushes the last elements value and multiplicity to the different vectors.

values2.push\_back(element2);

valueMultiplicity2.push\_back(count2);

};

void outputMultiplicity2() {

//outputs the Array with its values and multiplicity

cout << "Array 2 has the following values and multiplicity:" << endl;

cout << endl;

//for loop steps through each x index for both vectors.

for (int x = 0; x < values2.size(); x++) {

cout << values2[x] << " has a multiplicity of " << valueMultiplicity2[x] << endl;

}

cout << endl;

};

void vectPush(int array1[], int Size) {

//initializes a pointer variable

int\* intPtr = array1;

//sets max elements to be max value then subtracts as element is placed in proper position

for (int maxElement = Size - 1; maxElement > 0; maxElement--) {

//checks to see if value being sorted is greater than the next if it is it calls swap function

for (int index = 0; index < maxElement; index++) {

if (\*(intPtr + index) > \*(intPtr + index + 1))

swap(\*(intPtr + index), \*(intPtr + index + 1));

}

}

for (int z = 0; z < Size; z++) {

\*(Array1Values+z) = \*(intPtr+z);

};

multiplicity1();

};

void vectPush2(int array2[], int Size) {

//creates pointer variable for second array

int\* intPtr2 = array2;

//sets max elements to be max value then subtracts as element is placed in proper position

for (int maxElement2 = Size - 1; maxElement2 > 0; maxElement2--) {

//checks to see if value being sorted is greater than the next if it is it calls swap function

for (int index = 0; index < maxElement2; index++) {

if (\*(intPtr2 + index) > \*(intPtr2 + index + 1))

swap(\*(intPtr2 + index), \*(intPtr2 + index + 1));

}

}

//pushes values into a vector to be compared

for (int z = 0; z < Size; z++) {

Array2Values[z] = intPtr2[z];

};

multiplicity2();

};

void swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

public:

void setvalues() {

//public array for the class to access private member functions

//sets size of the arrays and initialized two arrays with inputs from different files

const int size = 600;

Size = size;

static int Array1[size] = {};

static int Array2[size] = {};

int\* file1Ptr = Array1;

int\* file2Ptr = Array2;

//initializes file for first array to get data from

ifstream inputFile;

inputFile.open("ComFile1.txt");

if (!inputFile) {

cout << "Error opening data file\n";

}

else

{

for (int x = 0; (x < size) && (inputFile >> \*(file1Ptr+x)); x++);

};

//closes file

inputFile.close();

//opens second file to get data from

ifstream inputFile2;

inputFile2.open("ComFile2.txt");

if (!inputFile2)

cout << "Error opening data file\n";

else

{

for (int c = 0; (c < size) && (inputFile2 >> \*(file2Ptr+c)); c++);

};

//closes second file

inputFile2.close();

//calls all the different private functions to perform their functions

//pushes values into different array

vectPush(Array1, size);

vectPush2(Array2, size);

//takes both arrays put them into vectors then displays their values and their multiplicity

//compars vector values and sizes

compareArray();

compareSize();

outputMultiplicity();

outputMultiplicity2();

}

};

int main()

{

//makes instance of compareArrays class and names it compare.

compareArrays compare;

//calls setvalues public method from compare class.

compare.setvalues();

cout << endl;

cout << endl;

//terminates program

return 0;

}

# Complexity:

* CompareArray complexity is O(n).
* The complexity for comparesize is O(n).
* Multiplicty1 and multiplicity2 have worst case complexity at also O(n).
* OutputMultiplicity1 and OutputMultiplicity2 also have a worst case complexity of O(n).
* Vectpush and vectpush2 have a worst case complexity of O(n^2).
* Setvalues has an interesting worst case complexity, it goes through two different files both being O(n) so it would put its worst case complexity of O(n^2). I believe even if it was two different files you must go through each one time.
* The swap function also has a complexity of N because it is just swapping the numbers.

Most of these programs have a worst case complexity of O(n) because they must run through the array once to compare or to push the values into an array or vector. The others that have O(n^2), must either do a swap of the values or must run through each array more than once.

I believe the overall complexity of this program is still about O(n) because many of the functions run through the array once to do their functions.

The only part of the program that I think could have been designed faster would be the areas where there are swapping (the vectpush and vectpush2). I believe there could have been a way to possibly made it an O(n) rather than o(n^2).