

## **Problem statement**

The first problem (Exercise 1) essentially asks us to design a program that can evaluate whether two vectors are equal or not despite duplicate elements and the order that the elements are inputted.

It is mentioned that the vectors' elements would be defined by the user. Therefore, there isn't a designated size nor a size limit for the vectors.

## **Analysis**

The input for this program is a set of integers that are separated by a single space between them. Each of these integers would later be considered an element of the vector it was inputted for.

The output of this program is both a value for Boolean variable called result and also sentence, which states whether the two vectors are equal or not. The mentioned two are made as result after the two vectors are evaluated.

There are several variables that need to be implemented in this program to carry out tasks such as storing the vectors, storing their sizes, storing the number of elements in one vector repeated in the other (For the first algorithm) as well as a string variable to hold the input before it is split into different elements and stored in an integer variable. Additionally, there also exist a Boolean variable, as mentioned earlier, to store the result of evaluation as a true or false value.

## **Design**

### **1. First Algorithm (less efficient)**

As the first step, the two vectors are received from the user. In this step, the user is asked to input a set of numbers which are separated by an empty space between them. These set of numbers are stored in a string variable called line. Then using a string stream, the mentioned line would be split into different elements, separated by the single space, set into an int variable and then written to a vector.

In the next step, the two input vectors are sent into a function called `same_vec`. There, by using a for loop, we first look for each of the elements of the first vector inside the second one. Each time we find an element from the first vector in the second, we add 1 to a variable called `sameValueCounter1`. Then we do the same thing for the second variable but this time, each time we find the same value from the second vector in the first we add 1 to a second int variable called `sameValueCounter2`.

As the last step, we will see whether the size of the first and second vector are equal to `sameValueCounter1` and `sameValueCounter2` respectively. This basically means that every element in the first vector has been found in the second vector vice versa. Therefore, if this condition is met, the two vectors are identical.

In summary, this algorithm is designed in a way to check, element by element, the elements of each vector with the other to decide whether they are equal or not.

## **2. Second Algorithm**

This algorithm is believed to be much more efficient compared to first Algorithm. The initial step of this algorithm is the same as the last one. In this step we ask for two vectors as mentioned before.

In this algorithm, in addition to the `same_vec` function, there are two more functions named `sortVector` and `duplicateRemover`. First, we send each vector to the `sortVector` and there the elements of the vector would be sorted from smallest to largest. Then each vector is sent to the `duplicateRemover`. Here we loop through the vector and remove any element that has been repeated.

After going through these two functions, the two vectors have to be exactly identical (same size and elements) to be considered identical. Therefore, a simple if condition is added as the last step to evaluate the mentioned.

In summary, this algorithm would sort and remove any repeated elements before comparing the two vectors. After the mentioned is completed, equal vectors are expected to be exactly identical and we can test that by a simple if condition containing “==”.

## Implementation

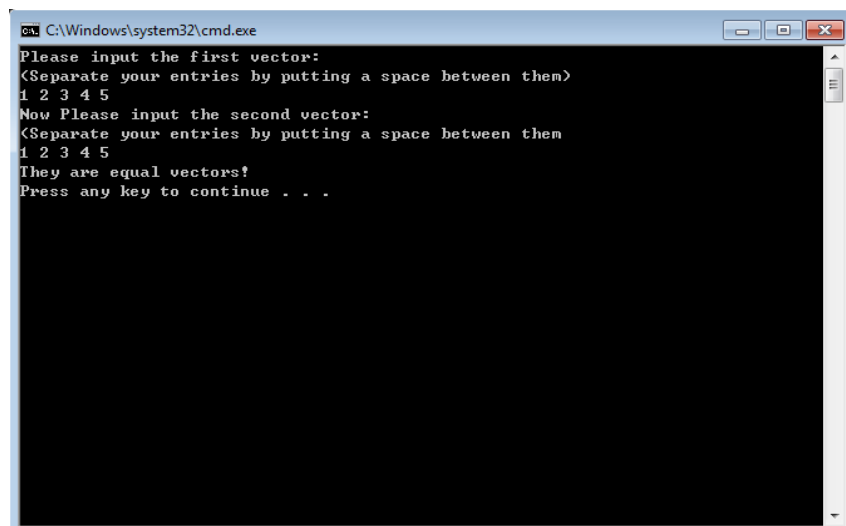
Name of the file regarding the first algorithm ==> Exercise1-FirstAlgorithm

Name of the file regarding the second algorithm ==> Exercise1-SecondAlgorithm

## Testing

### 1. First Algorithm

For testing the first algorithm, two exactly identical vectors were put in as input to secure that the basic part of the program is properly functioning (Figure 1).

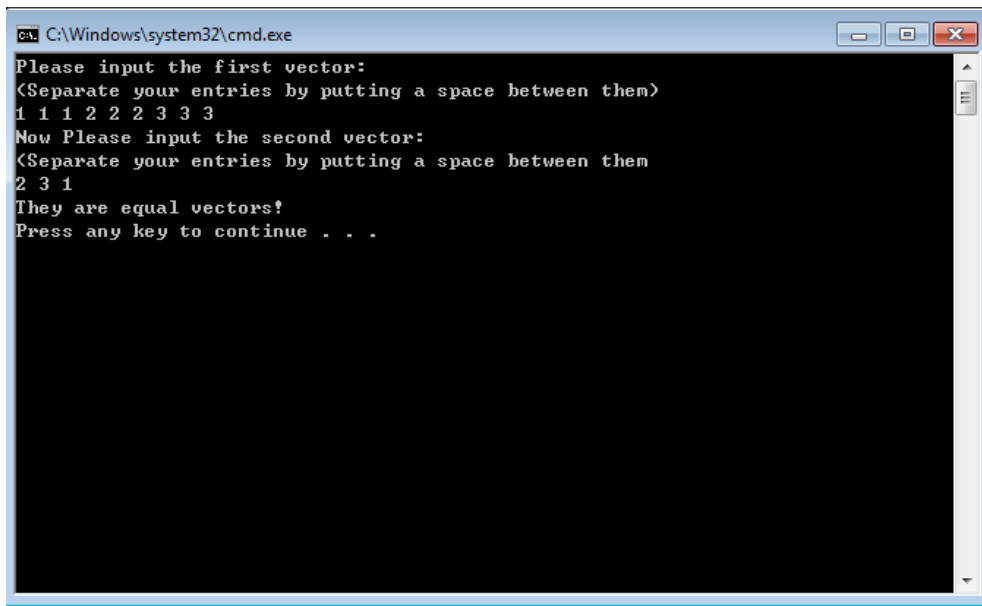


```
C:\Windows\system32\cmd.exe
Please input the first vector:
<Separate your entries by putting a space between them>
1 2 3 4 5
Now Please input the second vector:
<Separate your entries by putting a space between them>
1 2 3 4 5
They are equal vectors!
Press any key to continue . . .
```

Figure 1: the initial test

As the second and third step, two different sets of vectors, with different orders and duplicities were inputted to test the program as a whole (Figure 2).

**Note:** It should be noted that due to prevention of redundancy, many of the test screenshots were not included inside this report as it is believed they all show the same outcome. There were numerous possibilities and combinations tested for both of the algorithms to ensure that both algorithms function properly.

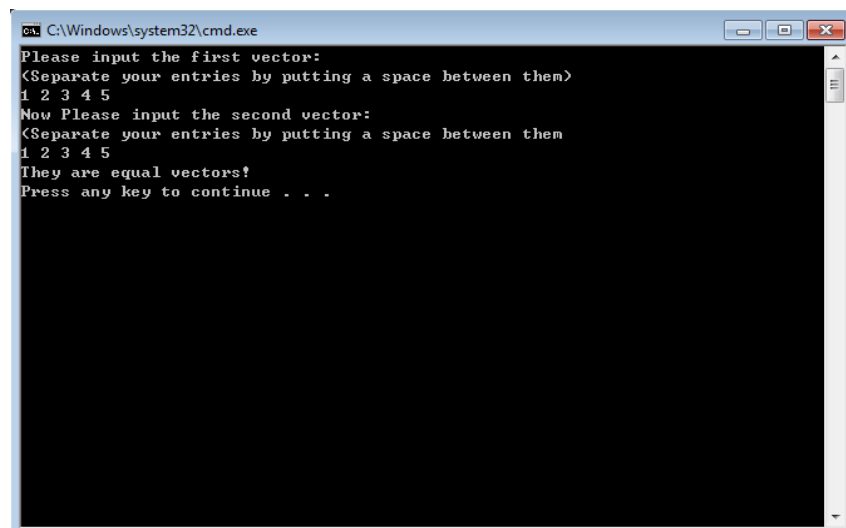


```
C:\Windows\system32\cmd.exe
Please input the first vector:
<Separate your entries by putting a space between them>
1 1 1 2 2 2 3 3 3
Now Please input the second vector:
<Separate your entries by putting a space between them>
2 3 1
They are equal vectors!
Press any key to continue . . .
```

Figure 2: the integrated test

## 2. Second Algorithm

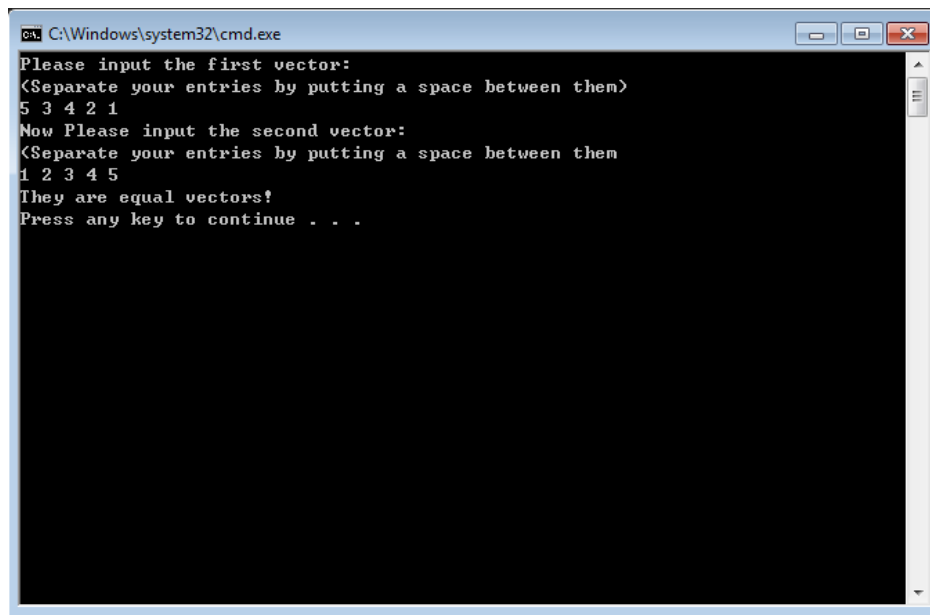
For testing the second algorithm, as done before, two exactly identical vectors were put in as input to secure that the basic part of the program is properly functioning (Figure 1).



```
C:\Windows\system32\cmd.exe
Please input the first vector:
<Separate your entries by putting a space between them>
1 2 3 4 5
Now Please input the second vector:
<Separate your entries by putting a space between them>
1 2 3 4 5
They are equal vectors!
Press any key to continue . . .
```

Figure 1: the initial test

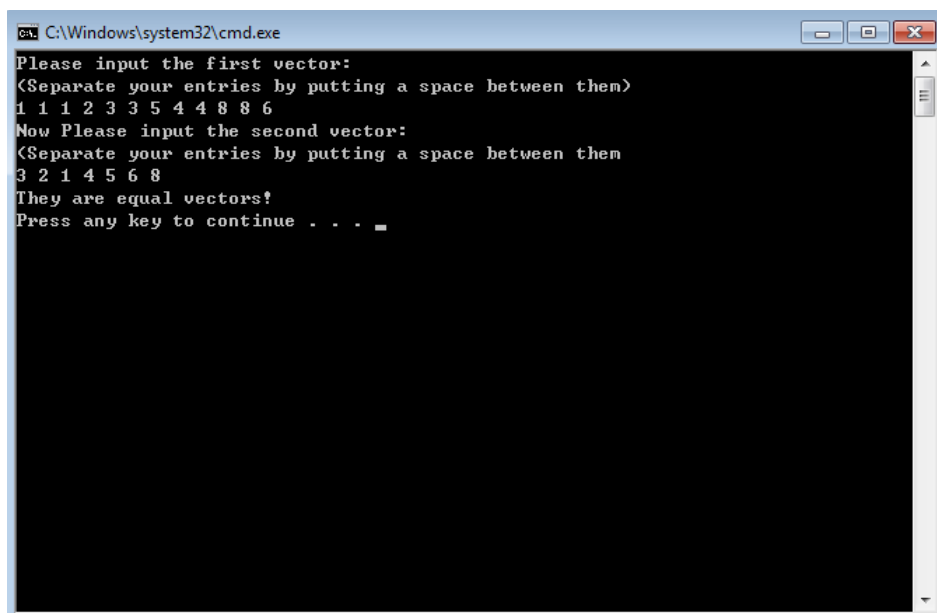
As the second test, the second vectors' elements were chosen in the exact opposite order compared to the first vector to assure that the sortVector function is properly functioning (Figure 2).



```
C:\Windows\system32\cmd.exe
Please input the first vector:
(Separate your entries by putting a space between them)
5 3 4 2 1
Now Please input the second vector:
(Separate your entries by putting a space between them)
1 2 3 4 5
They are equal vectors!
Press any key to continue . . .
```

Figure 2: the sortVector test

Consequently, the elements of the vectors were chosen to be the same, with different order and duplications. The aim of this was to check whether the duplicateRemover function was working without any problems or not (Figure 3).



```
C:\Windows\system32\cmd.exe
Please input the first vector:
(Separate your entries by putting a space between them)
1 1 1 2 3 3 5 4 4 8 8 6
Now Please input the second vector:
(Separate your entries by putting a space between them)
3 2 1 4 5 6 8
They are equal vectors!
Press any key to continue . . .
```

Figure 3: the duplicateRemover test