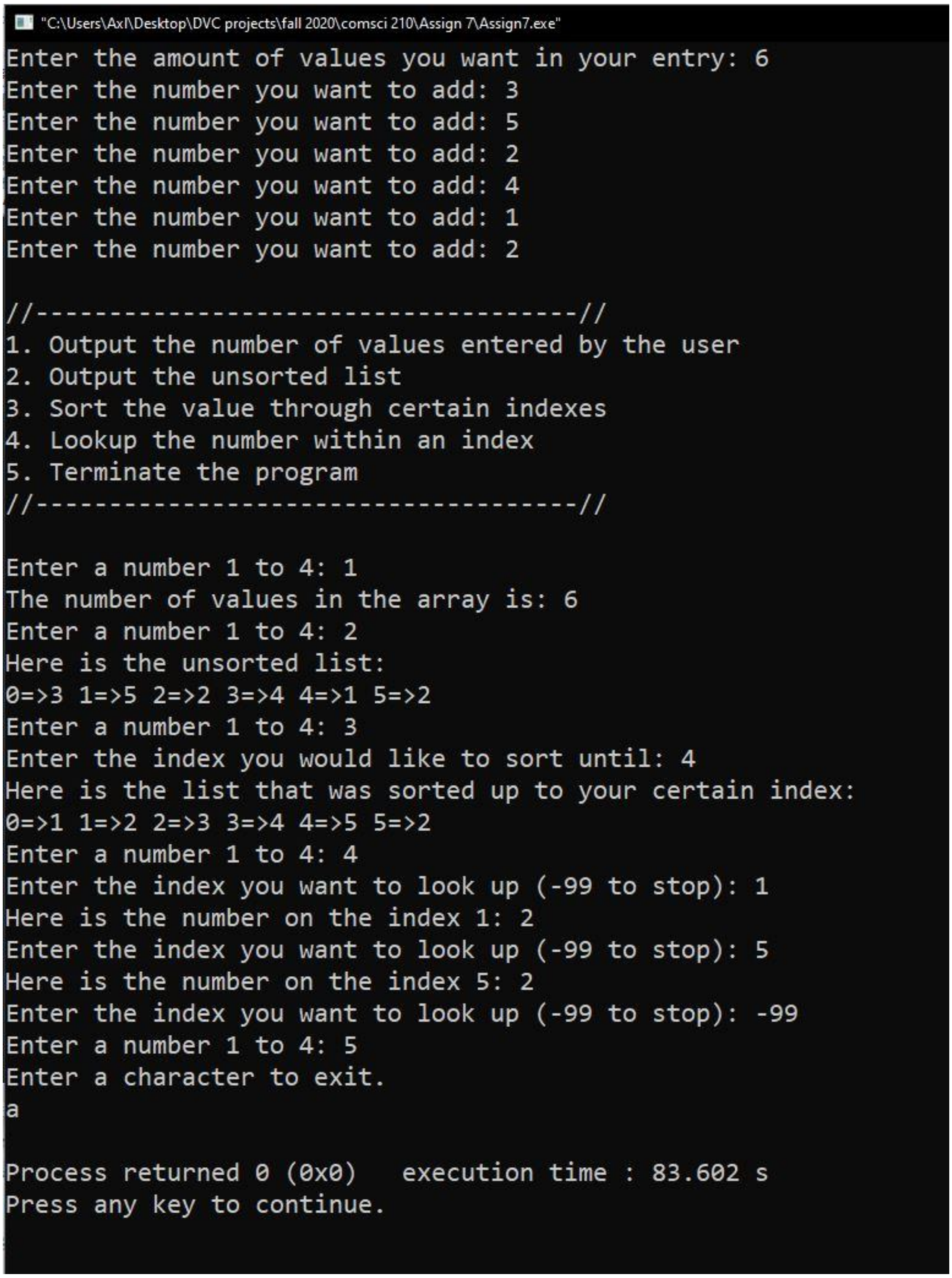
"C:\Users\Axl\Desktop\DVC projects\fall 2020\comsci 210\Assign 7\Assign7.exe"

```
Enter the amount of values you want in your entry: 6
Enter the number you want to add: 3
Enter the number you want to add: 5
Enter the number you want to add: 2
Enter the number you want to add: 4
Enter the number you want to add: 1
Enter the number you want to add: 2

//-------------------------------------//
1. Output the number of values entered by the user
2. Output the unsorted list
3. Sort the value through certain indexes
4. Lookup the number within an index
5. Terminate the program
//-------------------------------------//

Enter a number 1 to 4: 1
The number of values in the array is: 6
Enter a number 1 to 4: 2
Here is the unsorted list:
0=>3 1=>5 2=>2 3=>4 4=>1 5=>2
Enter a number 1 to 4: 3
Enter the index you would like to sort until: 4
Here is the list that was sorted up to your certain index:
0=>1 1=>2 2=>3 3=>4 4=>5 5=>2
Enter a number 1 to 4: 4
Enter the index you want to look up (-99 to stop): 1
Here is the number on the index 1: 2
Enter the index you want to look up (-99 to stop): 5
Here is the number on the index 5: 2
Enter the index you want to look up (-99 to stop): -99
Enter a number 1 to 4: 5
Enter a character to exit.
a

Process returned 0 (0x0)   execution time : 83.602 s
Press any key to continue.
```

Assign 7.cpp:

```
#include<iostream>

#include<string>

#include<cstdlib>

#include "MySortableArray.h"

#include "MySortableArray.cpp"


using namespace std;



int main()

{

   //variable declarations

   MySortableArray<int> nums;

   int choice;

   int sortIndex;

   int lookup;

   int max_entry;

   int value;


   //user instruction for array construction

   cout << "Enter the amount of values you want in your entry: ";

   cin >> max_entry;



   //create the array

   for (int i = 0; i < max_entry; i++)

   {

      cout << "Enter the number you want to add: ";

      cin >> value;

      nums.addEntry(value);
```

```
}


//menu
cout << endl;
cout << "//-------------------------------------//" << endl;
cout << "1. Output the number of values entered by the user" << endl;
cout << "2. Output the unsorted list" << endl;
cout << "3. Sort the value through certain indexes" << endl;
cout << "4. Lookup the number within an index" << endl;
cout << "5. Terminate the program" << endl;
cout << "//-------------------------------------//" << endl;
cout << endl;



//menu loop
do{
  cout << "Enter a number 1 to 4: ";
  cin >> choice;
  switch(choice)
  {
  case 1:
    cout << "The number of values in the array is: " << nums.getSize() << endl;
    break;
  case 2:
    cout << "Here is the unsorted list: " << endl;
    for (int i = 0; i < nums.getSize(); i++)
    cout << i << "=>" << nums.getEntry(i) << " ";
    cout << endl;
    break;
  case 3:
    cout << "Enter the index you would like to sort until: ";
```

```
        cin >> sortIndex;
        nums.sort(0, sortIndex);
        cout << "Here is the list that was sorted up to your certain index: " << endl;
        for (int i = 0; i < nums.getSize(); i++)
        cout << i << "=>" << nums.getEntry(i) << " ";
        cout << endl;
        break;
      case 4:
        while(lookup != -99)
        {
          cout << "Enter the index you want to look up (-99 to stop): ";
          cin>> lookup;
          if(lookup != -99)
          {
            cout << "Here is the number on the index " << lookup << ": "
            << nums.getEntry(lookup) << endl;
          }


        }
        break;
      }
    } while(choice != 5);




    cout << "Enter a character to exit." << endl;
    char wait;
    cin >> wait;
    return 0;
}
```

MySortableArray.h:

```
#ifndef MYSORTABLEARRAY_H
#define MYSORTABLEARRAY_H


#include <iostream>
using namespace std;


//Templated class and function declarations
//Alot of these are copy pasted from Assignment 3 and 2 so not all of them are used in this assignment


template <class item>
class MySortableArray //string dynamicarray has been changed to class dynamic array to allow for general usage
{


public:
   MySortableArray();


   MySortableArray(const MySortableArray& dsa);


   int getSize();


   void addEntry(item user);


   bool deleteEntry(item input);


   item getEntry(int input);


   MySortableArray operator==(const MySortableArray& dsa);


   void swap(int* a, int* b);
```

```
    int partition (int low, int high);


    void sort(int, int);


    ~MySortableArray();



private:
    item *dynamicArray;
    int size;


};


#endif // DYNAMICARRAY_H
```

MySortableArray.cpp:

```
#include "MySortableArray.h"


#include <iostream>
using namespace std;


//These are all functions
//They are also templated
//Most of these are copy pasted form assignment 3 so there are some unused functions
  template <class item>
  MySortableArray<item>::MySortableArray()
  {
    //constructor that sets the pointer to null and sizer to 0
    dynamicArray = nullptr;
    size = 0;
  }


  template <class item>
  MySortableArray<item>::MySortableArray(const MySortableArray& dsa)
  {
    //Creates a new array with size +1
    size = dsa.size;
    dynamicArray = new item[size]; //creates a new dynamic array with an item property
    for(int i = 0; i<size; i++)
    {
      //reassign values
      dynamicArray[i] = dsa.dynamicArray[i];
    }
  }


  template <class item>
```

```
int MySortableArray<item>::getSize()
{
  //returns the size of the array
  return size;
}

template <class item>
void MySortableArray<item>::addEntry(item user)
{
  //recreates a new dynamic array with an increased size
  item* newArray = new item[size+1]; //creates a new 'item' instead of the previous string
  size = size + 1;
  int i;
  for (i=0; i<size-1; i++)
  {
    //loop to assign the values to the new array
    newArray[i] = dynamicArray[i];
  }
  newArray[size-1] = user;
  delete[] dynamicArray;
  dynamicArray = newArray;
}

template <class item>
bool MySortableArray<item>::deleteEntry(item input)
{
  int j; //counter for the loop
  for(j = 0; j<size; j++)
  {
    if(dynamicArray[j] == input)
    {
      break;
      //breaks the loop when the input is found
    }
  }

  if(j==size)
  {
    //otherwise return false from the function
    return false;
  }

  //create a new dynamic array with a size that is 1 less
  item* newArray = new item[size-1];

  int l = 0;

  for(int k=0; k<size; k++)
  {
    if(dynamicArray[k]!=input)
    {
      //assigns every value that is not equal to the user input
      newArray[l++] = dynamicArray[k];
    }
  }
  //delete the dynamic array
  delete[] dynamicArray;
  //reduce the size
  size--;
  //reassign the new array to the dynamic array
  dynamicArray = newArray;
  //return true after recreating the arrays
```

```
        return true;


    }


    template <class item>
    item MySortableArray<item>::getEntry(int input)
    {
        //get entry and return the value when the input is considered valid
        if(input<size && input>=0)
        {
            return dynamicArray[input];
        }
        else
        {
            return NULL; //if invalid input, return nothing
        }
    }


    template <class item>
    MySortableArray<item> MySortableArray<item>::operator==(const MySortableArray& dsa)
    {
        //operator overloading for the == assignment operator
        size = dsa.size;
        dynamicArray = new item[size]; //now an "item" instead of string
        for(int i=0; i<size; i++)
        {
            dynamicArray[i] = dsa.dynamicArray[i];
        }
        return *this;
        //return the submitted input
```

```
}

//general swap function to swap 2 values
template <class item>
void MySortableArray<item>::swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

//function to divide the array into two by its pivot for the desired indexes
template <class item>
int MySortableArray<item>::partition (int low, int high)
{
    int pivot = dynamicArray[high];    // pivot
    int i = (low - 1);  // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (dynamicArray[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap(&dynamicArray[i], &dynamicArray[j]);
        }
    }
    swap(&dynamicArray[i + 1], &dynamicArray[high]);
    return (i + 1);
}
```

```
//the sorting function which uses recursion to sort through a QuickSort
template <class item>
void MySortableArray<item>::sort(int startIdx, int endIdx)
{
   if (startIdx < endIdx)
   {
      int pi = partition(startIdx, endIdx);

      sort(startIdx, pi - 1);
      sort(pi + 1, endIdx);
   }
}

template <class item>
MySortableArray<item>::~MySortableArray()
{
   //deconstructor that deletes the array
   delete[] dynamicArray;
}
```