A drawing of a face

Description automatically generated

A close up of a sign

Description automatically generated

CLI World Fact Book Documentation CO1401 Programming: (2019/2020)

Name: Nethmi Weeraman

IDNo: 3000149

Table of Contents:

* Table of figures
* Introduction
* World Fact-book Requirement
* Solution Approach
* Design
* Assumptions
* Implementation
* Testing the algorithms
* Testing
* Limitations
* Future Enhancements
* Conclusion

Table of Figures:

Figure 1. Flowchart for program

Figure 2. Functions

Figure 3.1. Struct for storing countries

Figure 3.2. Struct for adding commas in numbers

Figure 3.3. Struct for time and complexity calculation

Figure 4. Country vector

Figure 5. Array example

Figure 6. Some pointers from the program

Figure 7.1. Bubble sort test

Figure 7.2. Insertion sort test

Figure 7.3. Quick sort test

Figure 7.3.1. Unsorted table

Figure 7.3.2 ASC total population sort

Figure 7.3.3. ASC male population sort

Figure 7.3.4. ASC female population sort

Figure 7.3.5. ASC area sort

Figure 7.3.6. ASC calling code sort

Figure 7.3.7. DSC total population sort

Figure 7.3.8 DSC male population sort

Figure 7.3.9 DSC female population sort

Figure 7.3.10 DSC area sort

Figure 7.3.11 DSC calling code sort

Figure 7.4. Linear search test

Figure 7.5. Binary search test

Figure 7.6. Exponential search test

Figure 8. Error handling for main menu

Figure 9. Error handling for main menu

Figure 10. Error handling for add menu

Figure 11. Error handling for add menu

Figure 12. Error handling for add menu

Figure 13. Error handling for add menu

Figure 14. Error handling for sort menu

Figure 15. Error handling for sort menu

Figure 16. Error handling for search menu

Figure 17. Error handling for name search

Figure 18. Error handling for name search

Figure 19. Error handling for binary search

Figure 20. Error handling for exponential search

Figure 21. Unit test function for sorting

Figure 22. Applying unit test function

Figure 23. Unit testing for linear search

Figure 24. Unit testing for exponential search

Figure 25. Unit testing for binary search

Introduction:

The term ‘algorithm’ was founded around the 9th centuary, named after an arabian scholar at the time (Voiland, 2017). Since then and even before, unwittingly, it has been used as a backbone for all organised processes of resolving issues.

In the most fundamental definitions of the word, we can describe it as the steps used in solving a problem. In the realm of computers, it is a way computers process data, relying on user instruction to give out the required output. However, this in itself is not where the bar is set. The computer in solving a problem must follow the principal of accuracy and proper resource usage (Cormen, 2013).

The code written will attempt to fulfill these milestones to make a world factbook using searching and sorting algorithms to navigate information, along with file input and output.

World Fact Book Requirement:

The requirement specifications for this would primarly encompass, appending into a CSV file, searching and sorting through stored data, while dynamically allocating memory.

Ranking must be given accordingly to countries. The searching and sorting algorithm’s must have time taken in miliseconds and the complexity recorded as well.

The code should conform to readability and maintainability guidelines. The program should be tested with a suitable testing strategy and be documented.

Solution Approach:

The program is to be developed using C++;

The algorithms to be used are;

For searching;

* Linear search
* Binary search
* Exponential search

For sorting;

* Quicksort
* Insertion sort
* Bubble sort

The goal of the project is to create a simulated fact book, for this the following steps are to be taken,

* Data on countries is to be loaded into the program from a CSV file.
* User is to be given option to *sort, search* or *add*.
  + If chosen to *add*,

user is supposed to be prompted to enter details then asked if they’d like to add more or go to main menu.

* Error handling must be implemented so that no string can be entered where a number is required in details, if user does this they will have to be redirected to menu.
* Moreover, the program should automatically give correct casing in case of faulty input, for instance, the user inputting japAn, stores the country name as Japan automatically. This works for country name and capital. The added functionality works to bring more uniformity in data.
  + If chosen to *search*,

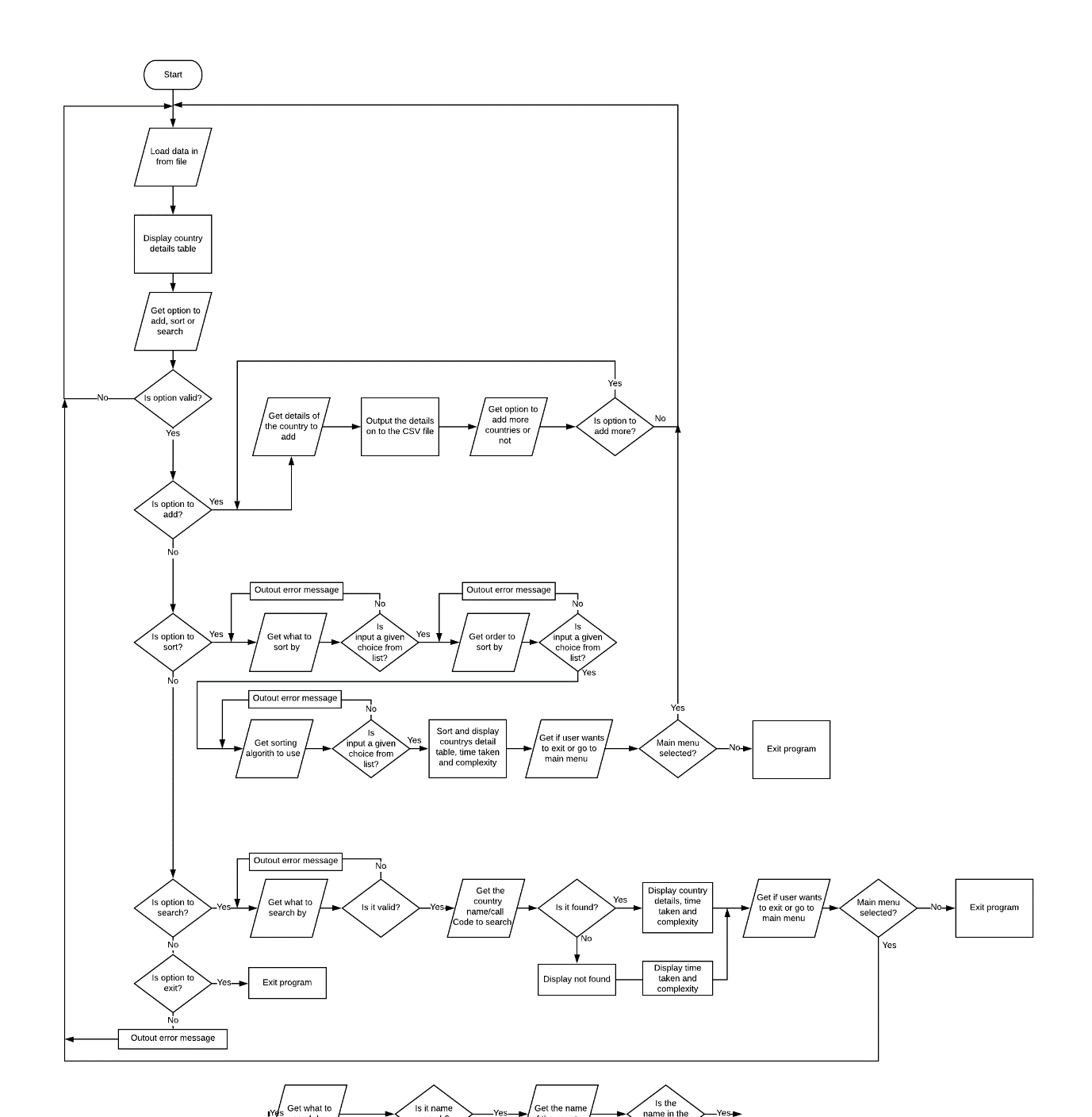
user is to be prompt to choose from name search using linear search algorithm, call code search using binary search algorithm or call code search using exponential search algorithm. After this, if there is a match or partial match, the details are displayed or if not, a message saying input not found is seen.

* When searching using country name, the program accepts various cases, for instance, it the user enters andorra or ANDORRA or anDorRa, it returns the country Andorra’s details.
* Moreover, there is a partial match system implemented, such that if the 4 consecutive letters in the suffix of the input is correct, the program will find a match. For instance, searching xdgZdorra, says a partial match is found and gives out the details of Andorra.
  + If chosen to *sort*,

user is to be questioned on what to sort by (eg: total population, male population, female population, area or calling code), which order(eg: ascending or descending), and what sort algorithm to use (eg: Quicksort, insertion sort or bubble sort). If all is entered correctly, country table is sorted and displayed. Then user can choose to exit or go to main menu.

* If at any stage of user gives an unlisted choice, the program must ask for correct input until it is entered. If user enters wrong data type, they are to be redirected to menu.
* A struct constructor and deconstructor is to be used to measure *time taken and complexity for algorithms* before and after they have been called respectively, and then find the difference and display it.

Design:

Below is a flowchart detailing the entire process described above, it is a graphical representation of the functionality of the entire program process.

- - - - - - - - - - - - - - - - - - - - - - - - -Searching process- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - - - - - -Sorting process- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - -- - - - - - - - - -

- - - - - - - - - - - - - - - - -Adding process- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - -

Figure 1. Flowchart for program

For the sake of simplicity, the function for input validation of correct data type is omitted from the diagram (Figure. 1). It works differently from the regular error handling, in that it redirects to main menu straight away unlike the error handling we see above that causes iteration until correct answer is given.

Assumptions:

* Program will run on other IDEs
* Users do not have any form of epilepsy

Implementation:

The code consists of,

* Functions
* Arrays
* Structs
* Vectors
* Pointers

and additionally,

* Constructor/Destructor

(A function object is instantiated at start of sorting and searching algorithms)

* Function template

(There is use of ‘typename’ template to allow for various datatypes to be entered into a single function)

Functions,

Functions allow for the program to be structured in a way where code can be divided into parts that serve to fulfil a purpose. Below the functions for the world fact book program are listed, and separated according to it’s use.

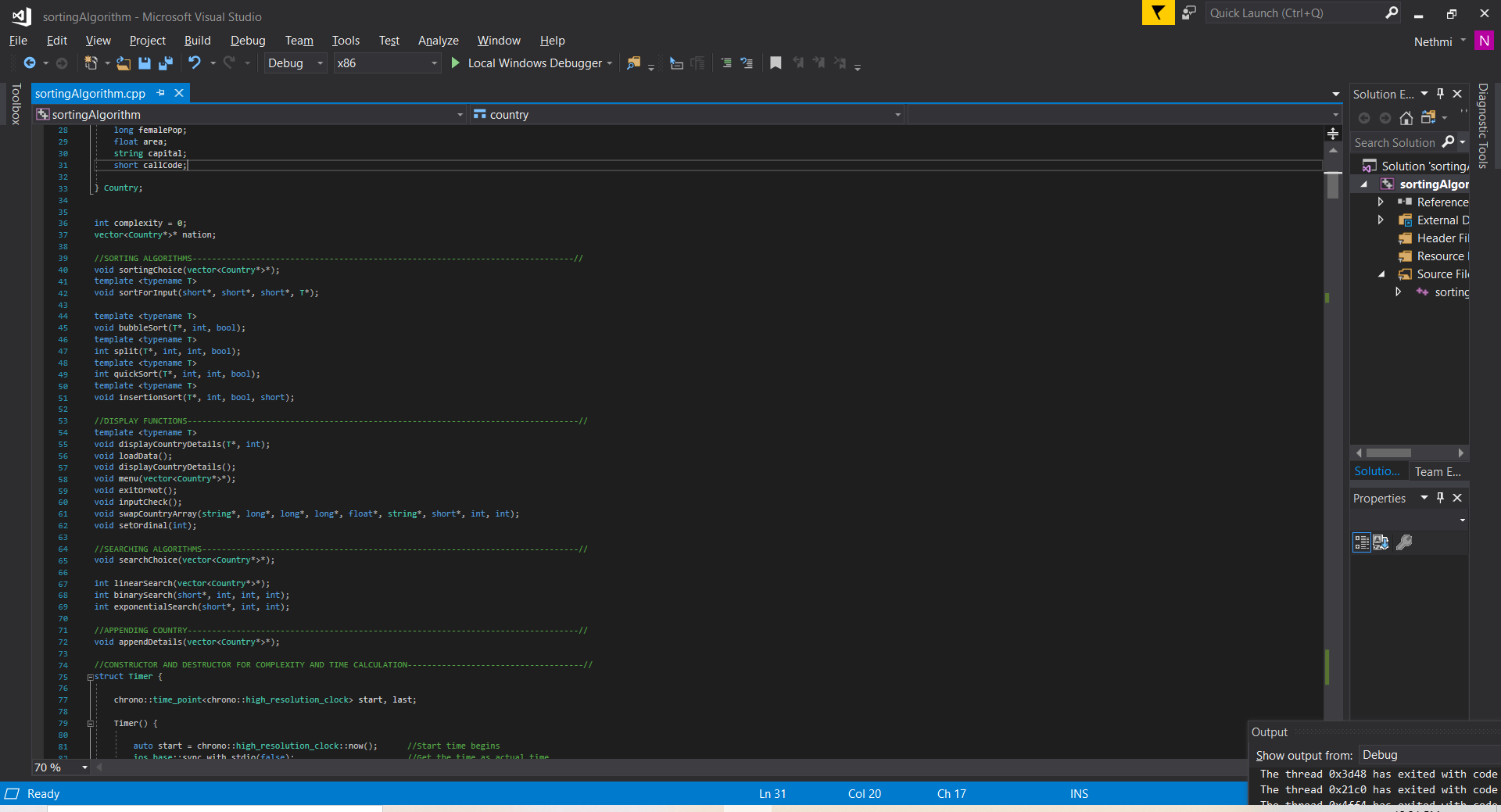


Figure 2. Functions

Structures,

Structures (structs) allow for data of different types to be held together. In the program code, there are three such structures.

1. Country struct ;

Stores country details.

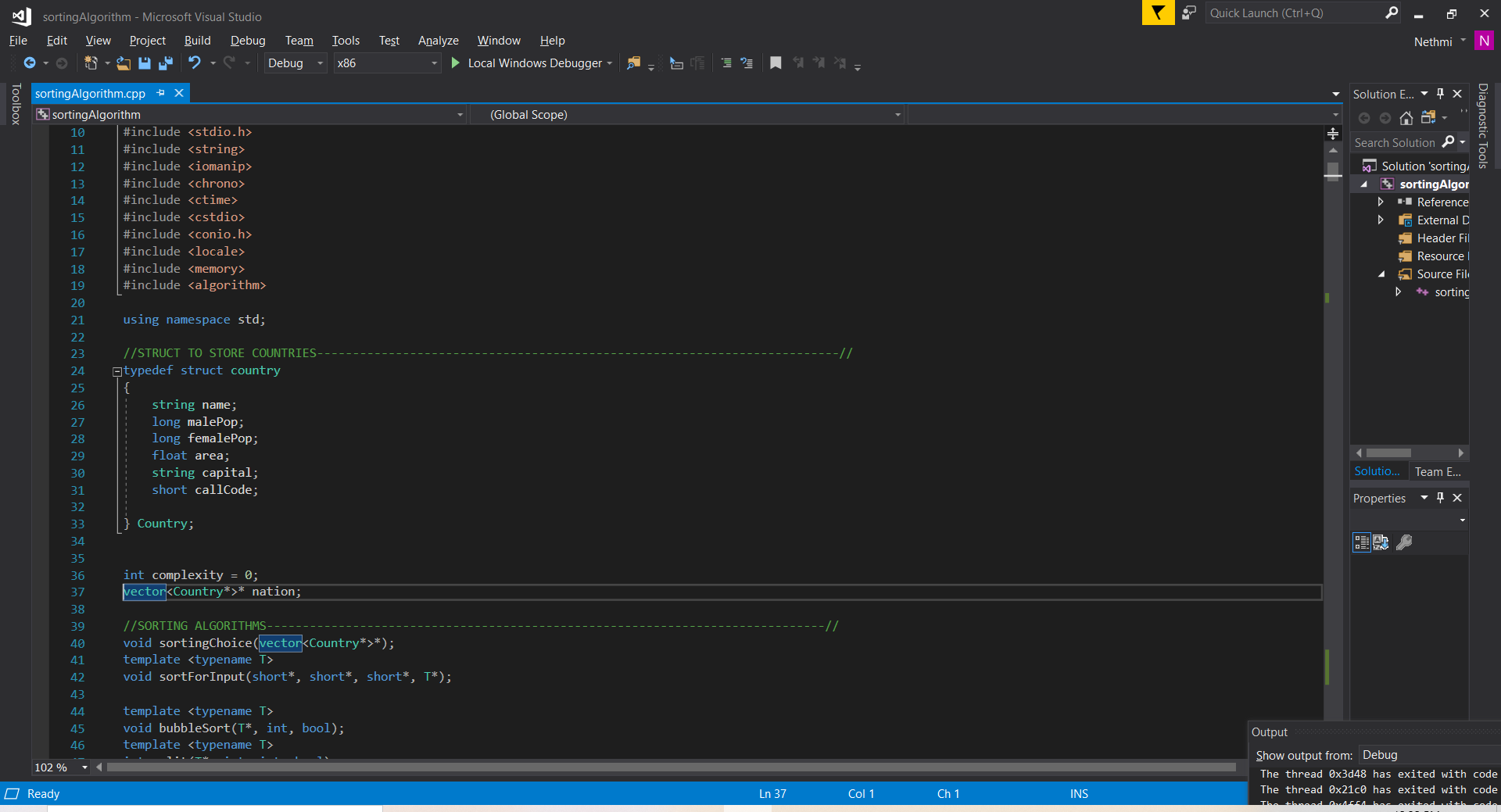


Figure 3.1. Struct for storing countries

1. Thousands separator struct;

Takes number as string type to place char ‘,’ and overrides numbers.

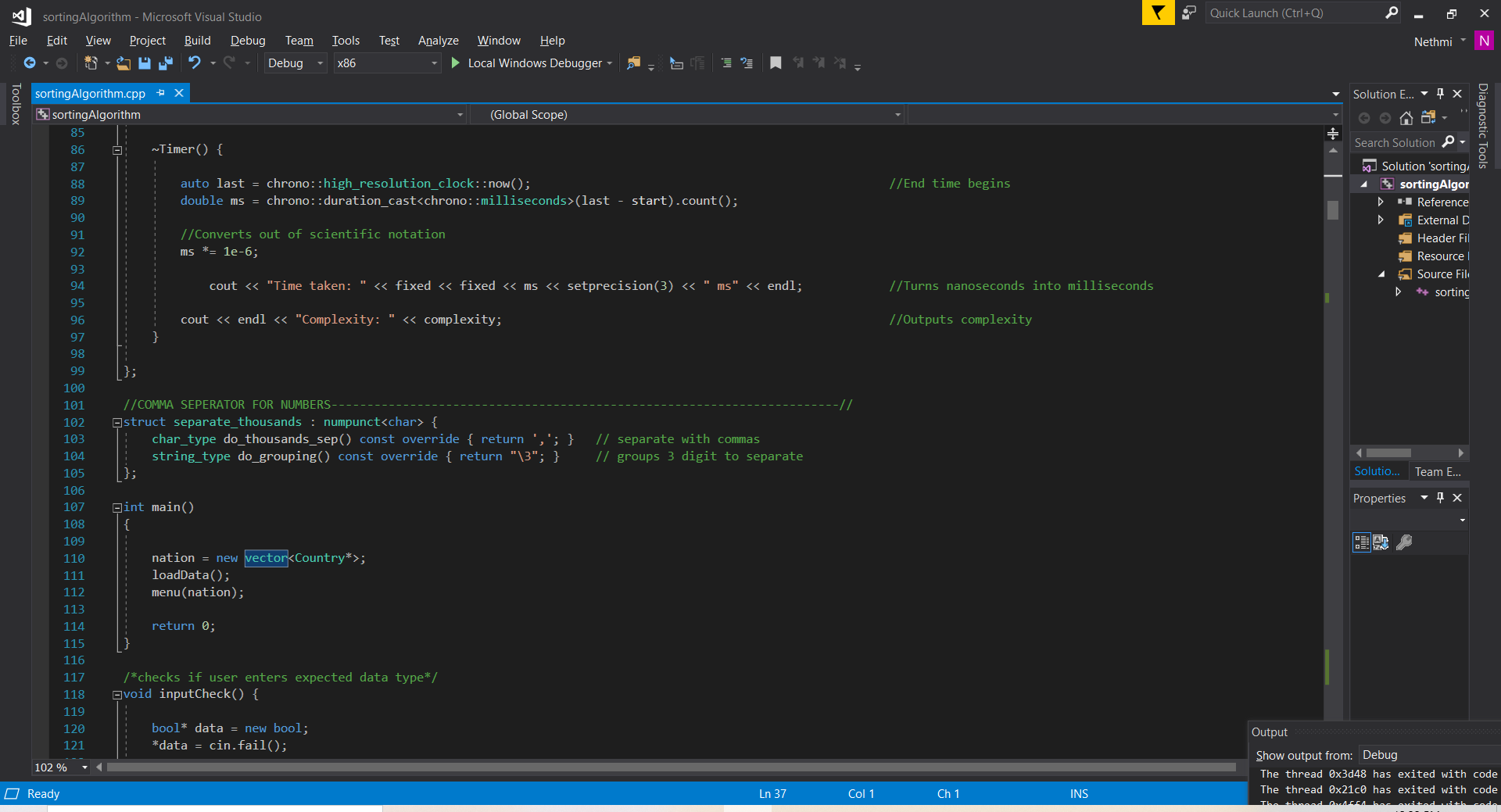


Figure 3.2. Struct for adding commas in numbers

1. Time and complexity struct;

Call constructor and destructor to the start and end of the function object is created in.

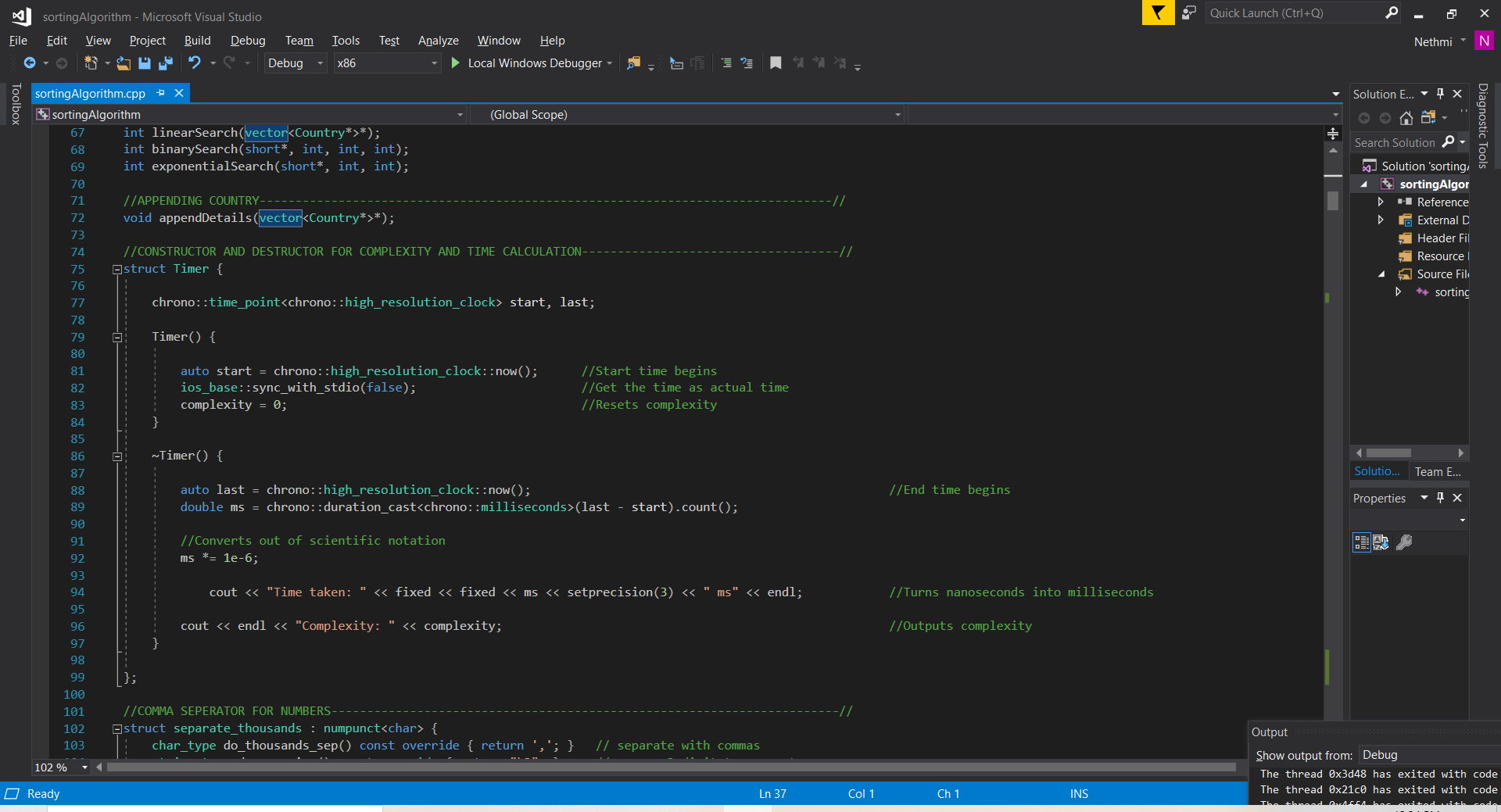


Figure 3.3. Struct for time and complexity calculation

Vector,

Vectors allow for dynamic memory allocation. In the program, a vector of type Country is made.



Figure 4. Country vector

Arrays,

Arrays allow for data of the same type to be stored together. There are temporary arrays within the program under sortingchoice( ) and displaycountry details( ), that are later deleted.

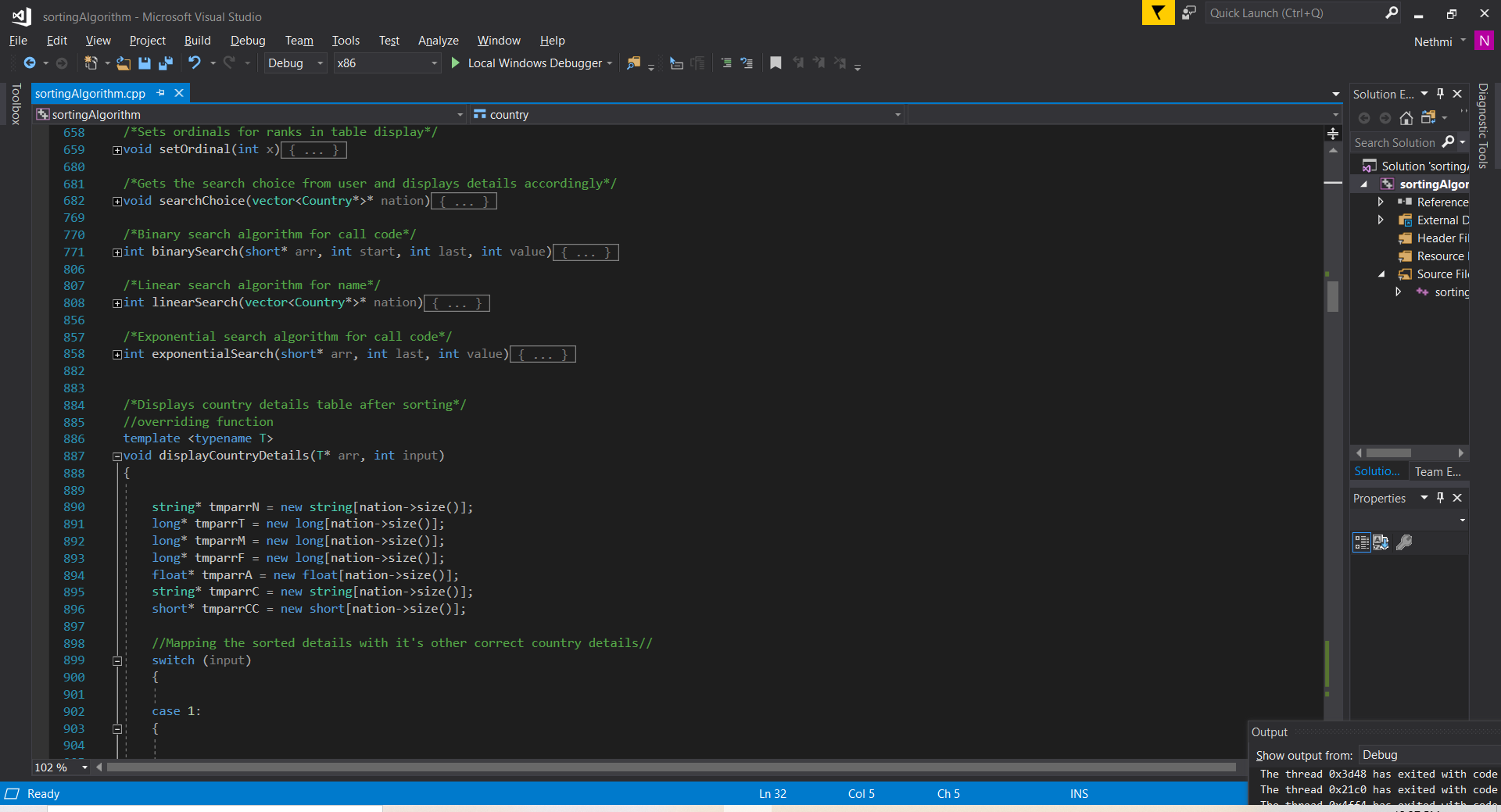
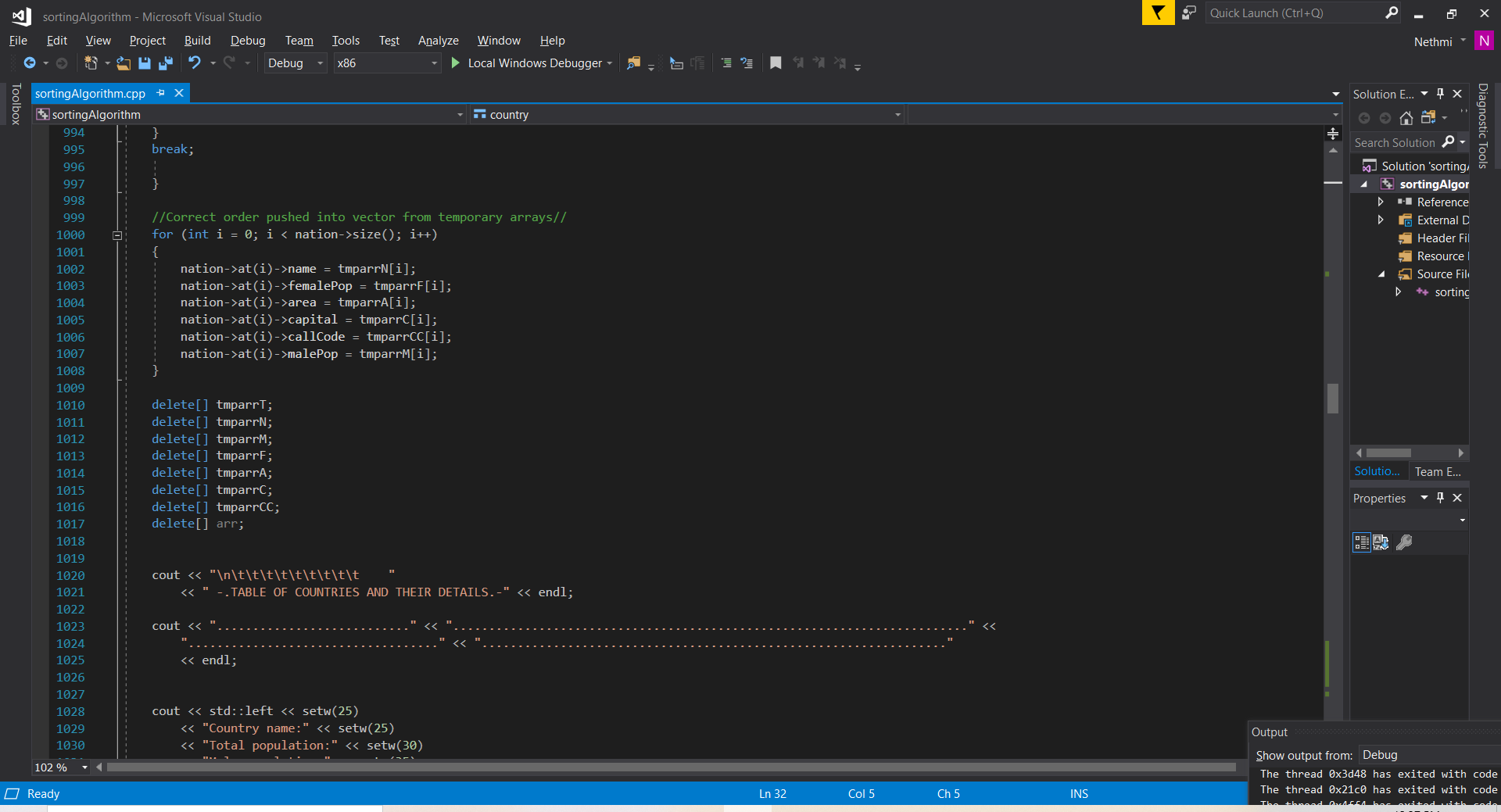
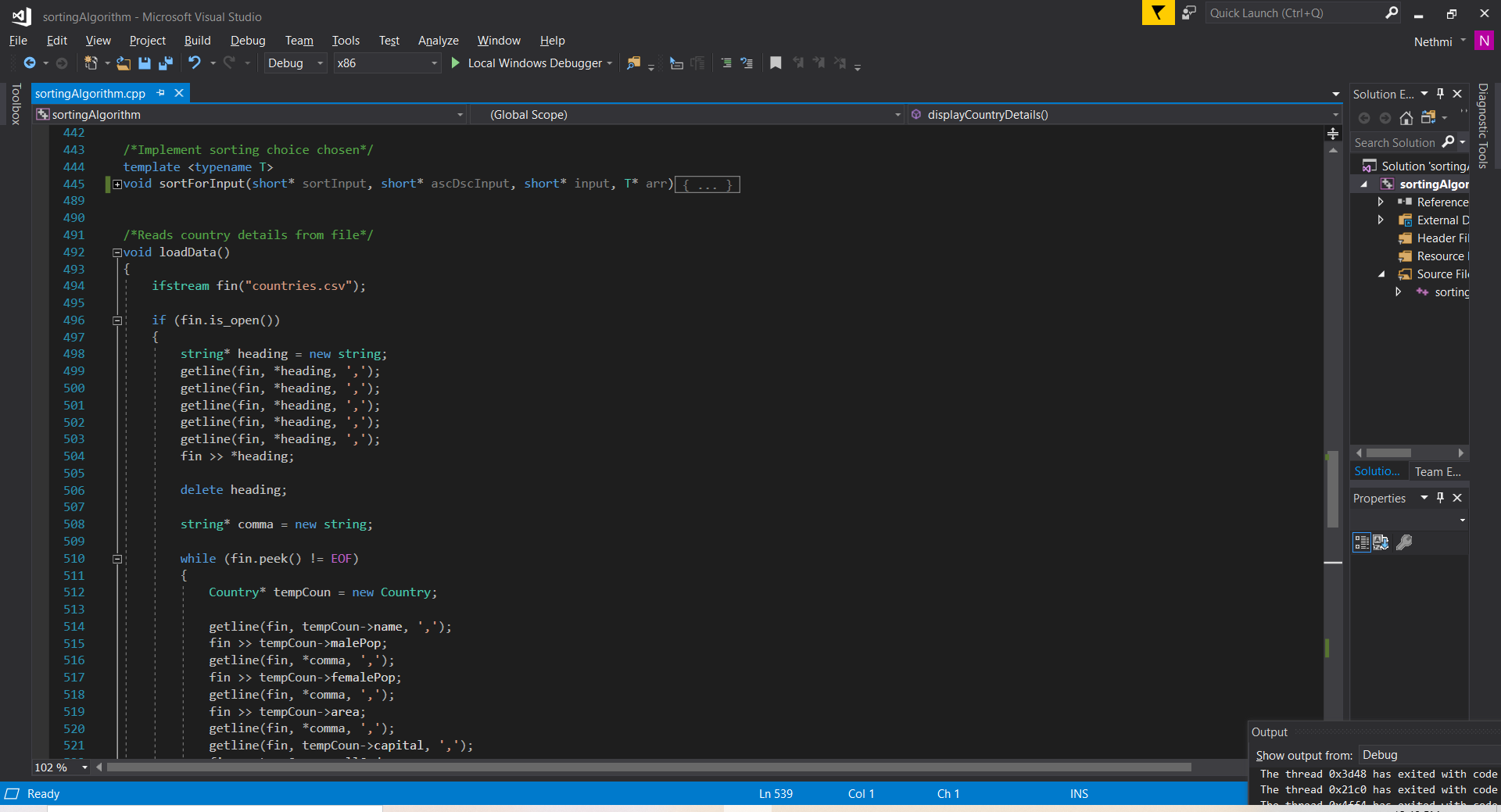
 

Figure 5. Array example

Pointers,

These pointers direct to the address of a variable they store in memory. The program makes use of pointers and in some parts use smart pointers as seen in top mostin the below figure, these delete automatically after use.

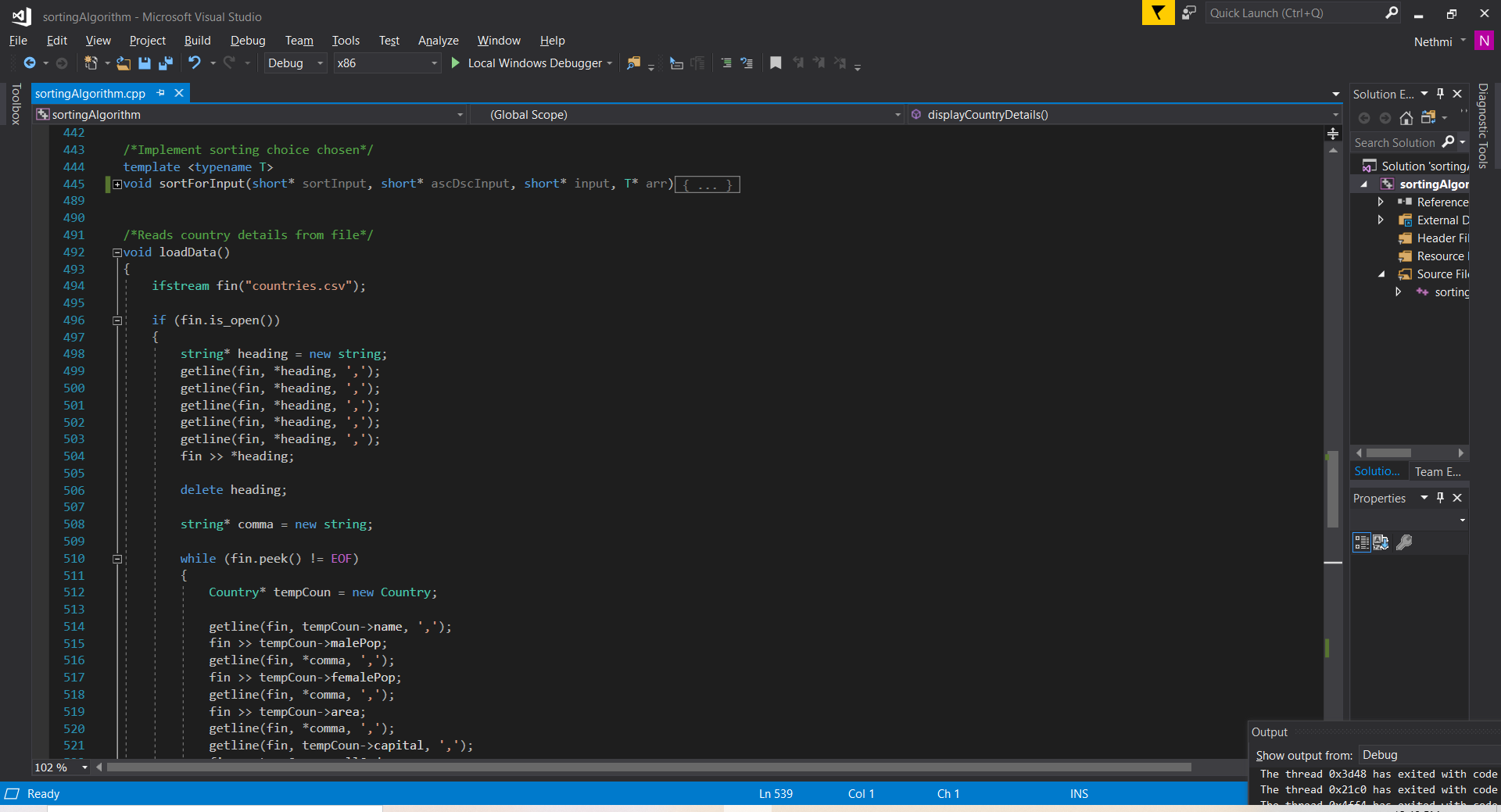


Figure 6. Some pointers from the program

Testing the algorithms;

**Sorting: *Time and complexity comparisons***

Bubble sort

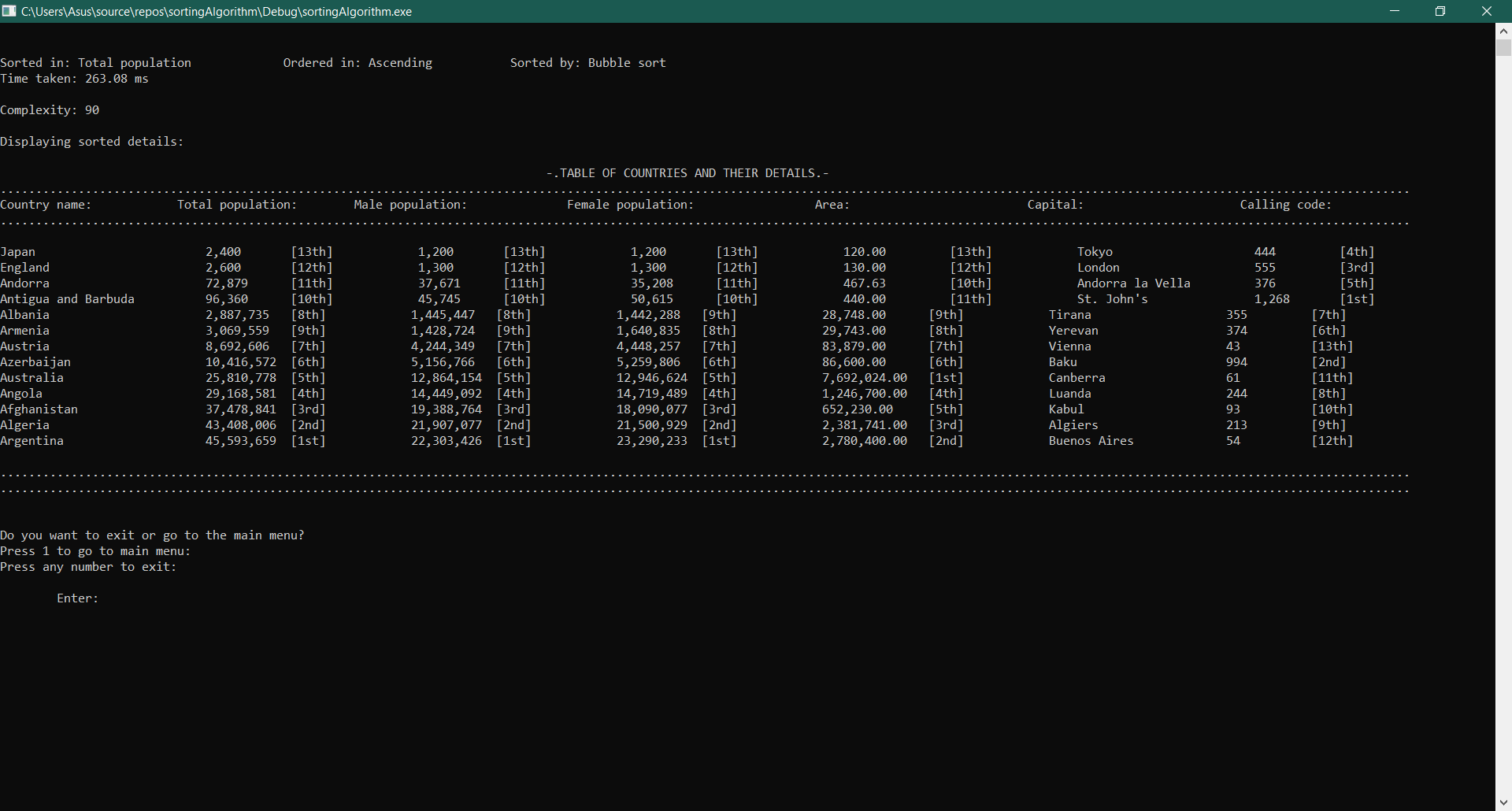


Figure 7.1. Bubble sort test

Insertion sort

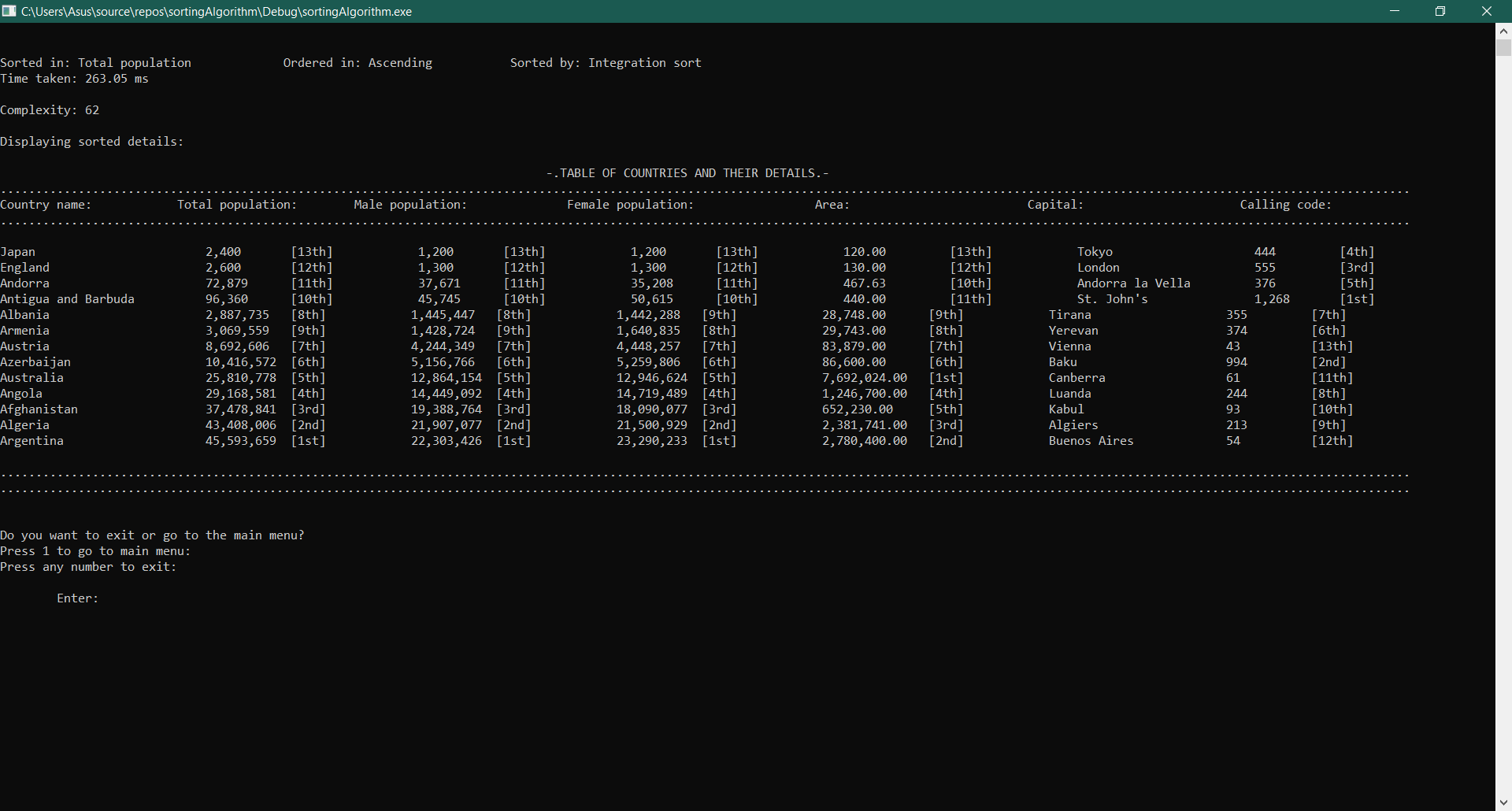


Figure 7.2. Insertion sort test

Quick sort

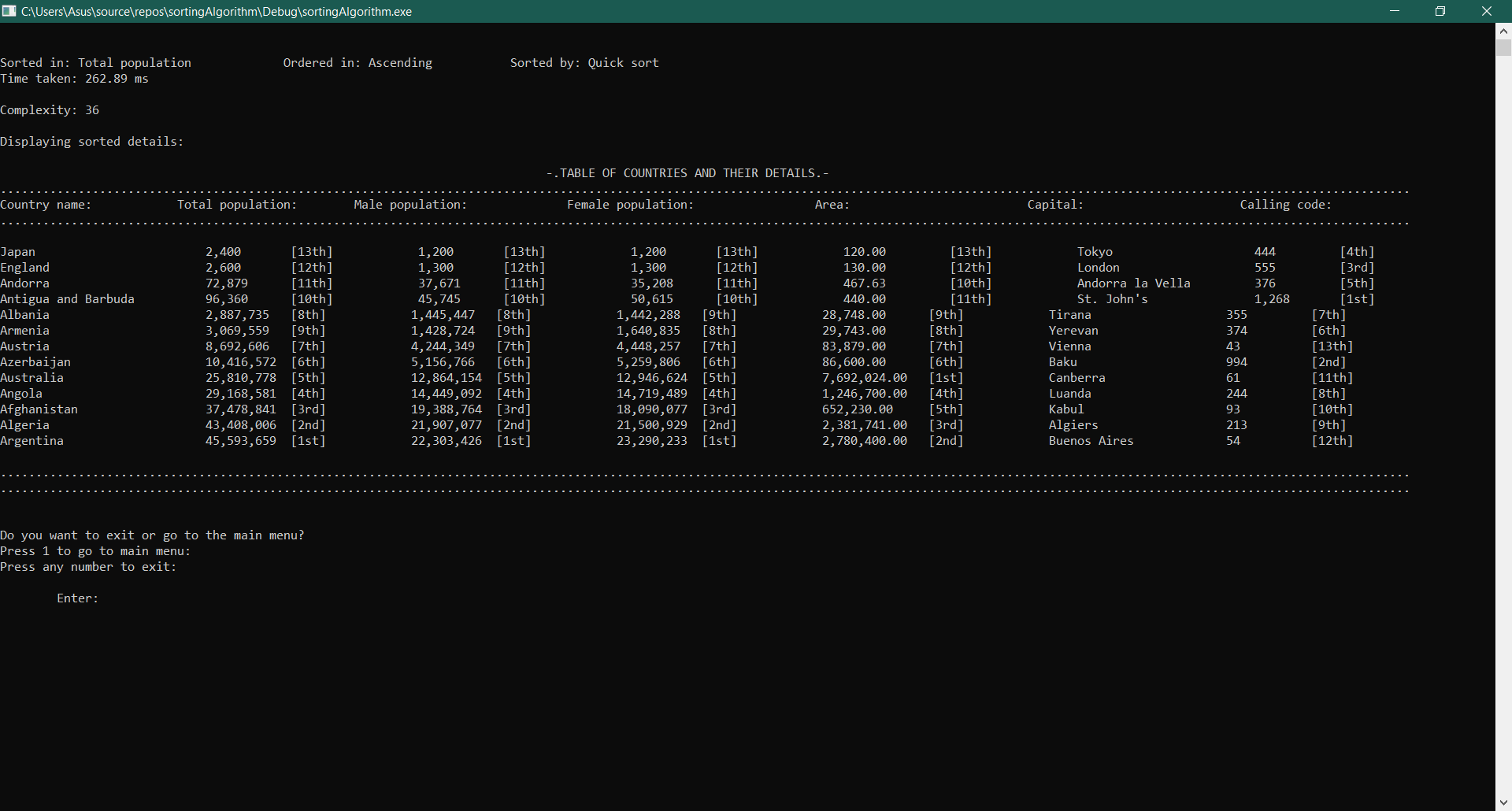


Figure 7.3. Quick sort test

Comparison between the sorting algorithms;

|  |  |  |  |
| --- | --- | --- | --- |
| Sort type | Bubble sort | Insertion sort | Quick sort |
| Time taken | 263.08 | 263.05 | 262.89 |
| Complexity | 90 | 62 | 36 |

The table above reflects the nature of the algorithms, with quick sort being the most efficient and bubble sort being the least.

Analysis: Sorting

*Complexity:*

Bubble sort

Best case: O(n) Average case: O(n2) Worst case: O(n2)

n in this case: 12, therefore

Best case: O(12) Average case: O(144) Worst case: O(144)

90 complexity lies in the range between average case and best case.

Insertion sort

Best case: O(n) Average case: O(n2) Worst case: O(n2)

n in this case: 12, therefore

Best case: O(12) Average case: O(144) Worst case: O(144)

62 complexity lies in the range between average case and best case.

Quick sort

Best case: O(n log n) Average case: O(n log n) Worst case: O(n2)

n in this case: 12, therefore

Best case: O(13) Average case: O(13) Worst case: O(144)

36 complexity lies in the range between average case and worst case.

*Time:*

The ranking of quickest are Quicksort, insertion sort and bubble sort respectively.

Although the disparity between the differences are not very dramatic with this number of inputs, as operations are scaled, onto a greater set of values to sort through, this gap becomes considerably more significant.

Ascending and descending sorting

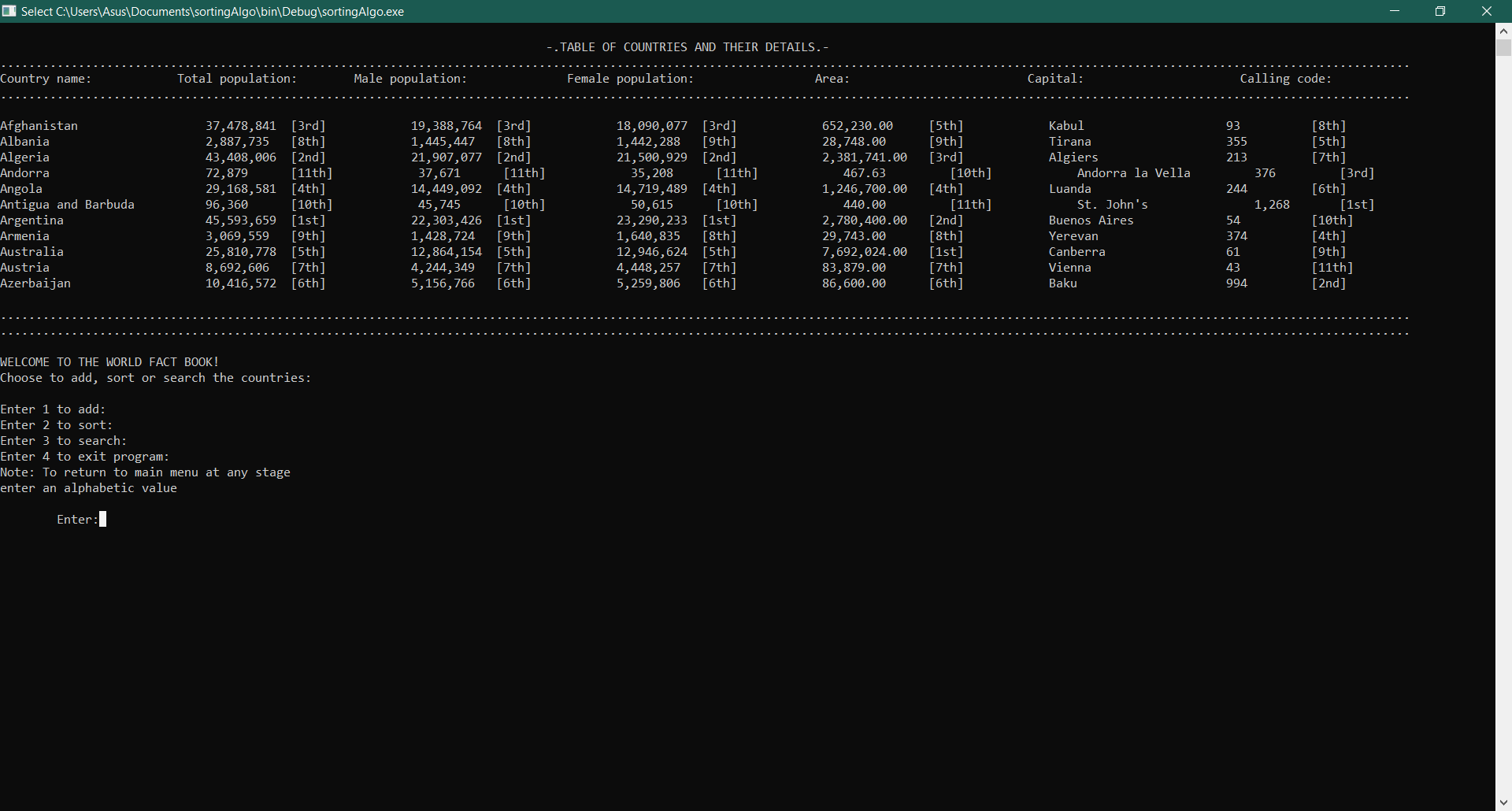
Unsorted table:

Figure 7.3.1. Unsorted table

Ascending

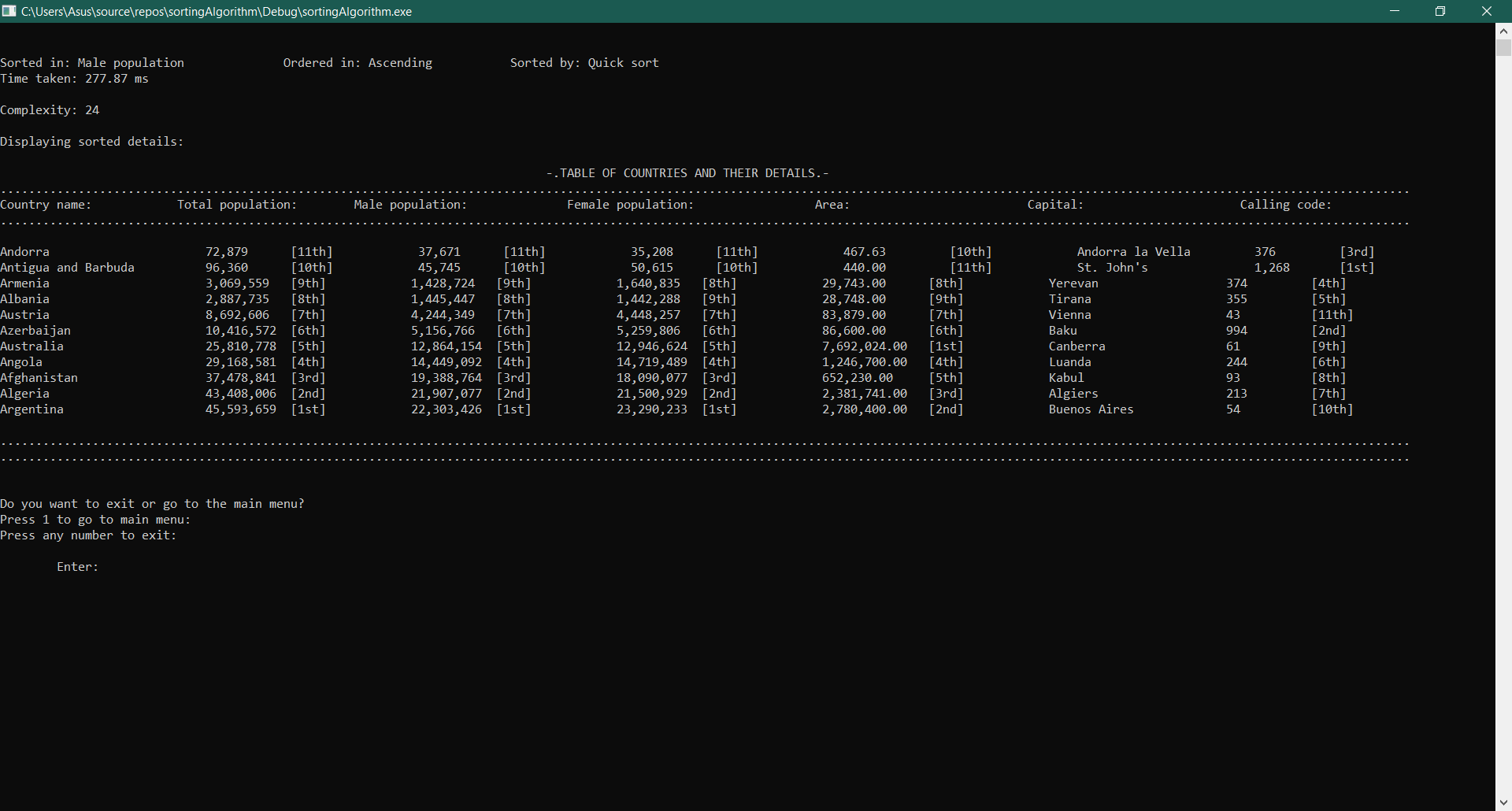
Table sorted by: Total population

Figure 7.3.2 ASC total population sort

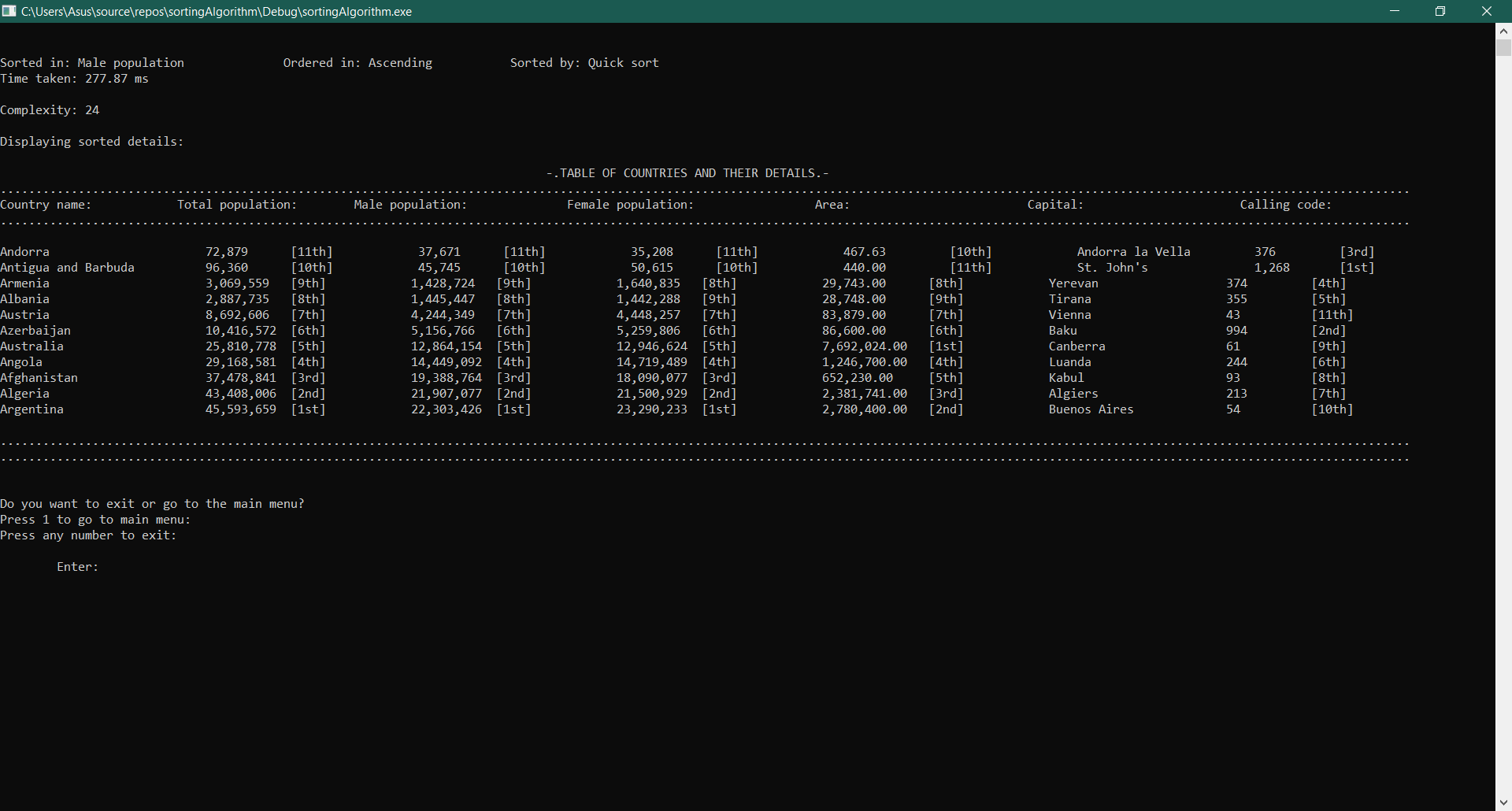
Table sorted by: Male population

Figure 7.3.3. ASC male population sort

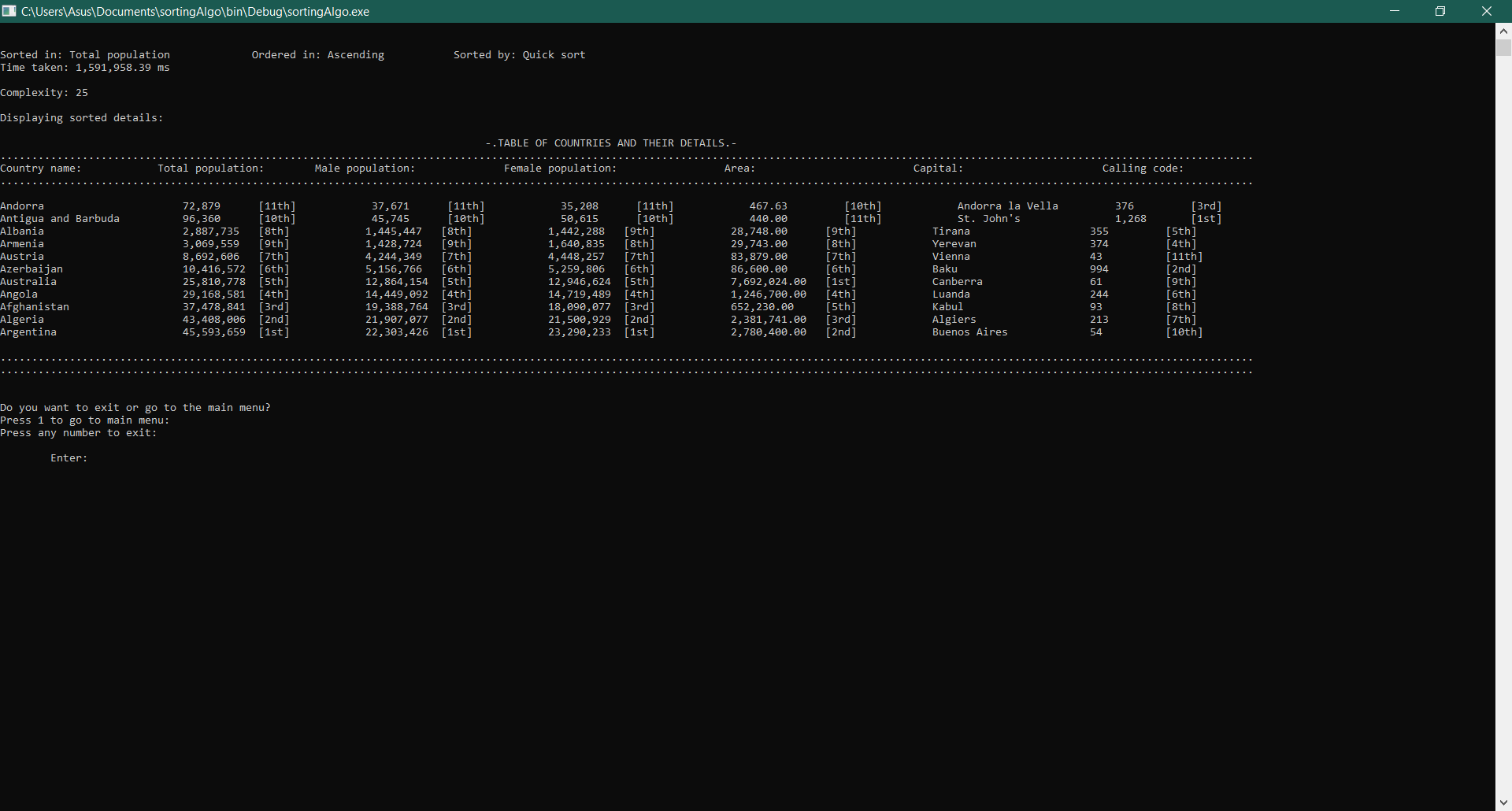
Table sorted by: Female population

Figure 7.3.4. ASC female population sort

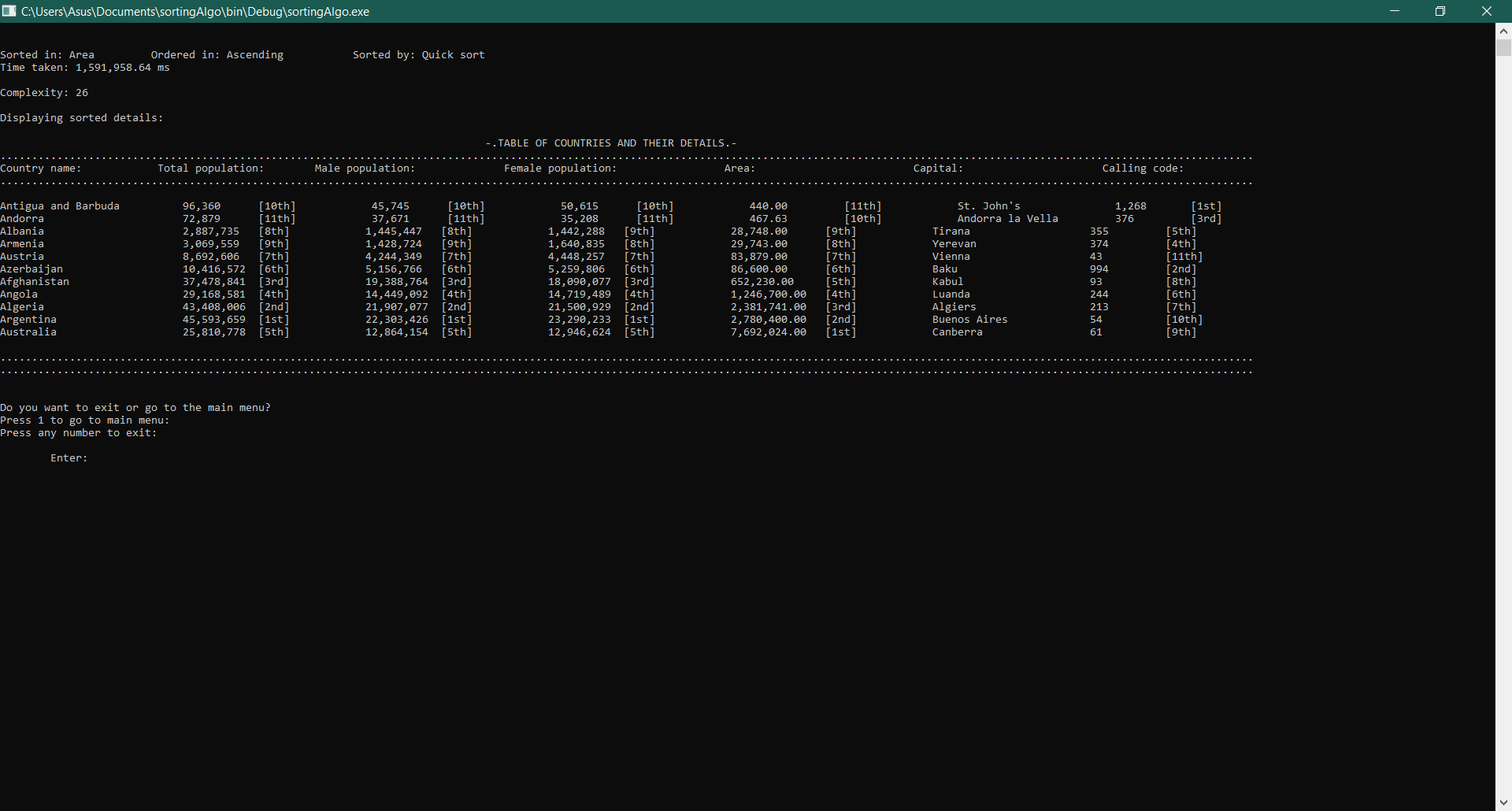
Table sorted by: Area

Figure 7.3.5. ASC area sort

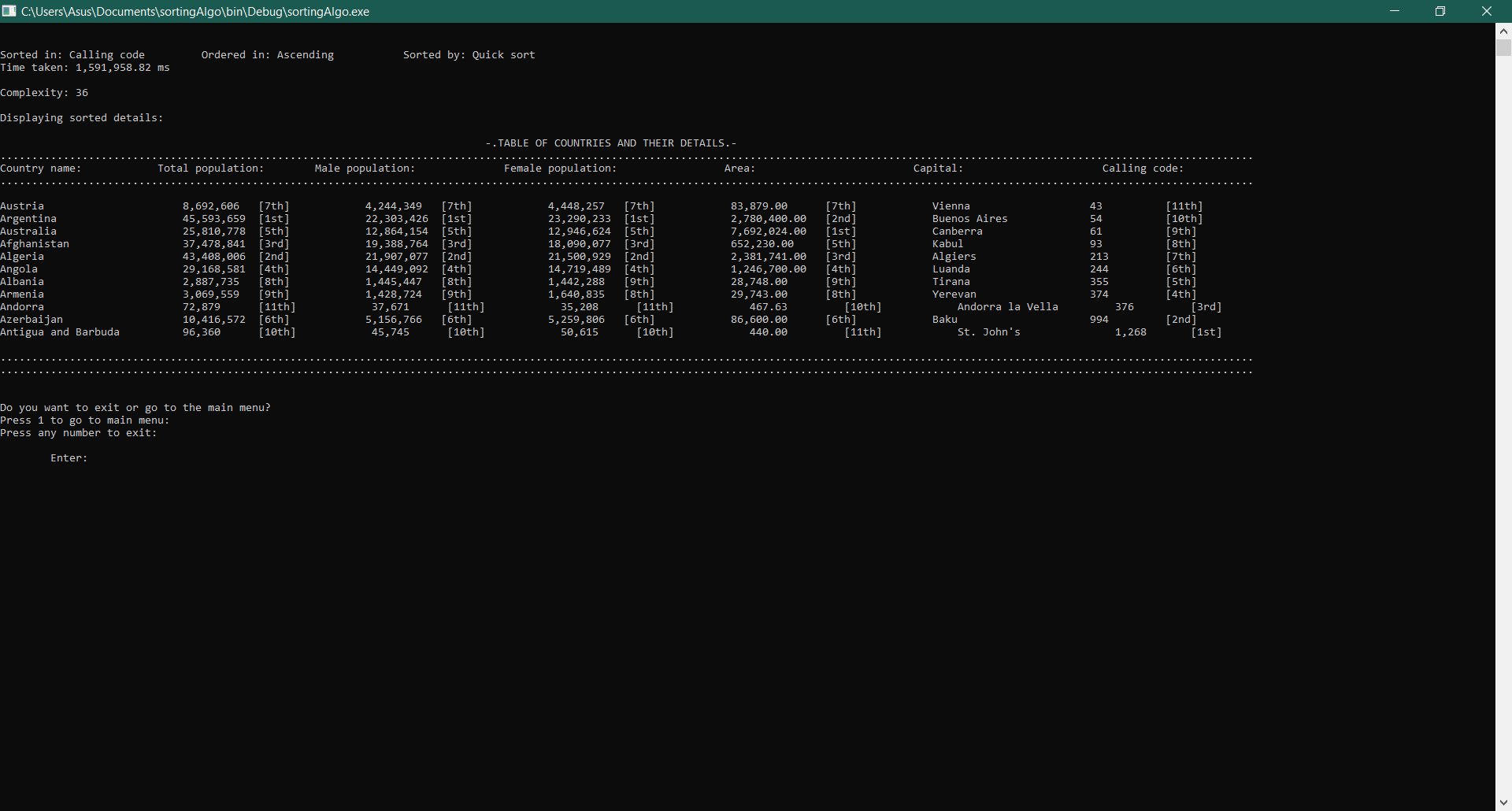
Table sorted by: Calling code

Figure 7.3.6. ASC calling code sort

Descending

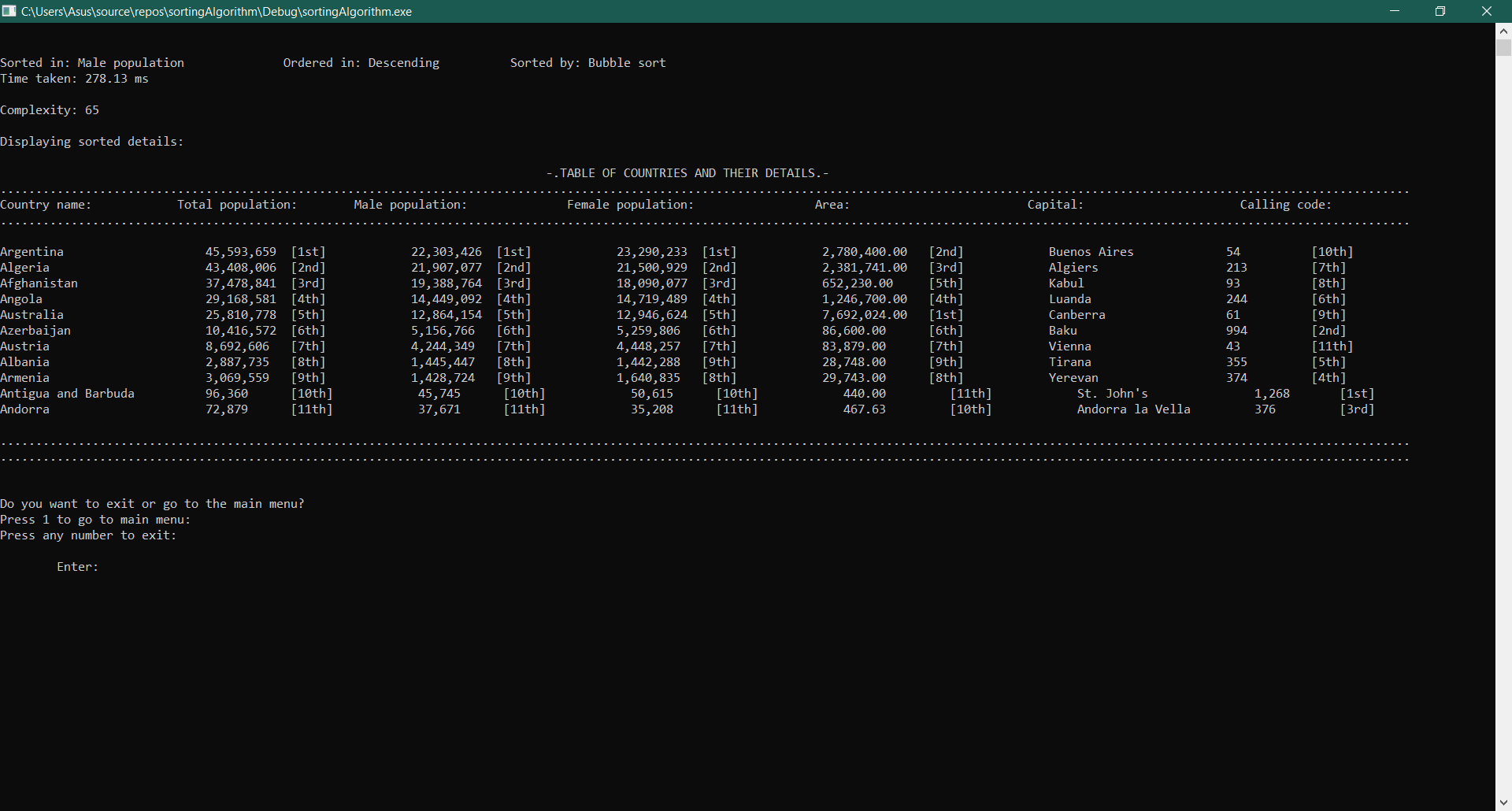
Table sorted by: Total population

Figure 7.3.7. DSC total population sort

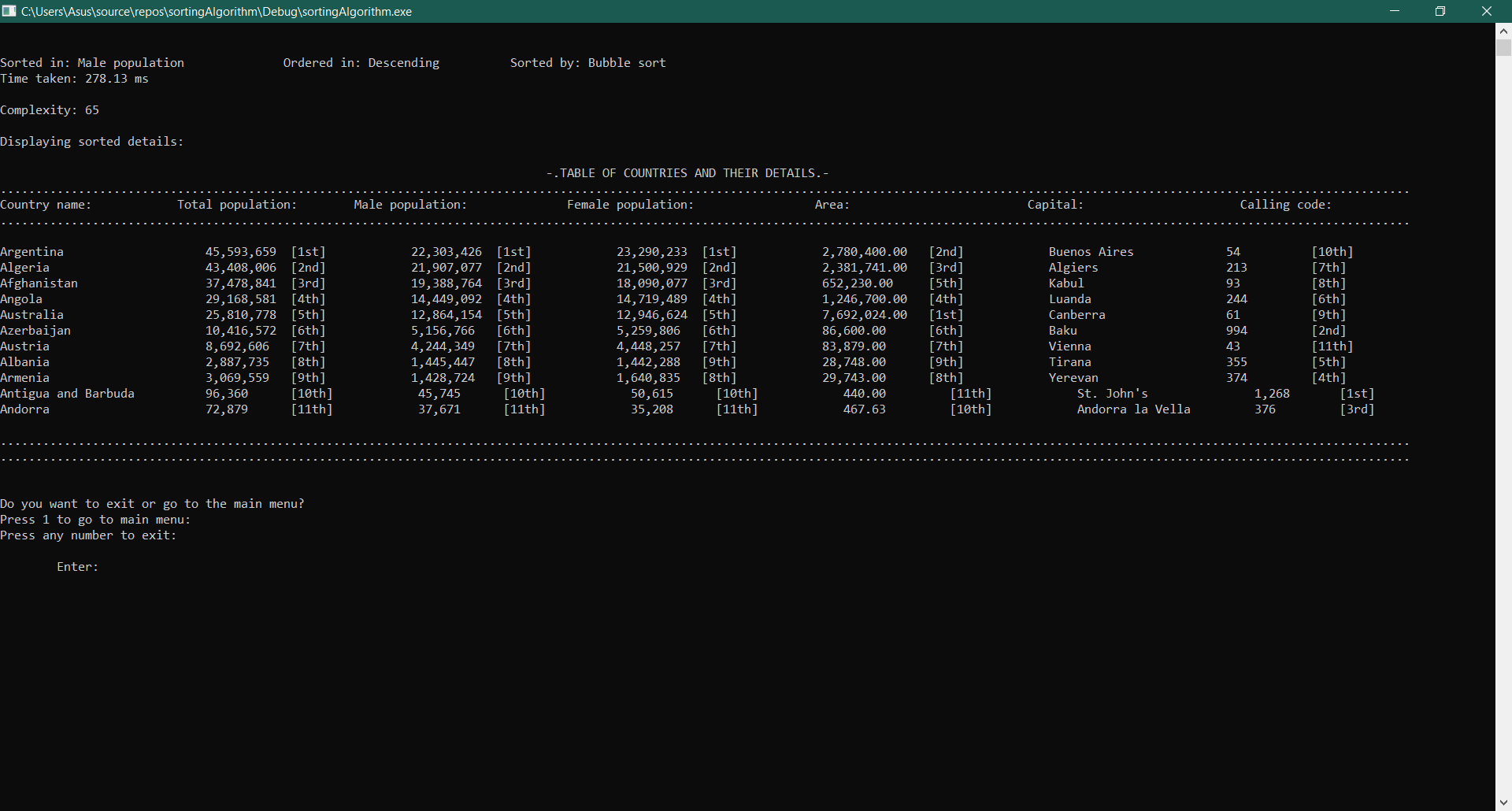
Table sorted by: Male population

Figure 7.3.8 DSC male population sort

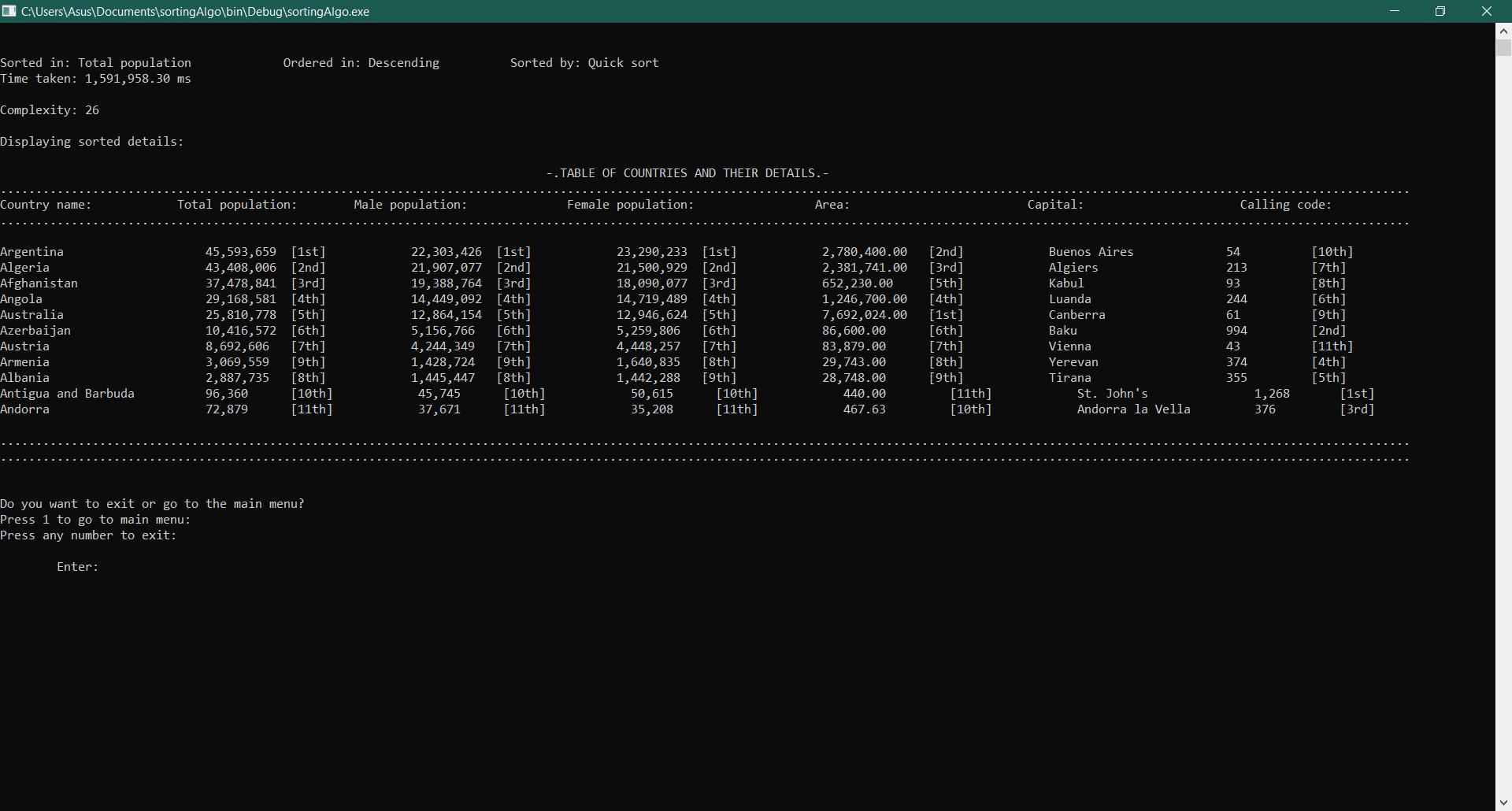
Table sorted by: Female population

Figure 7.3.9 DSC female population sort

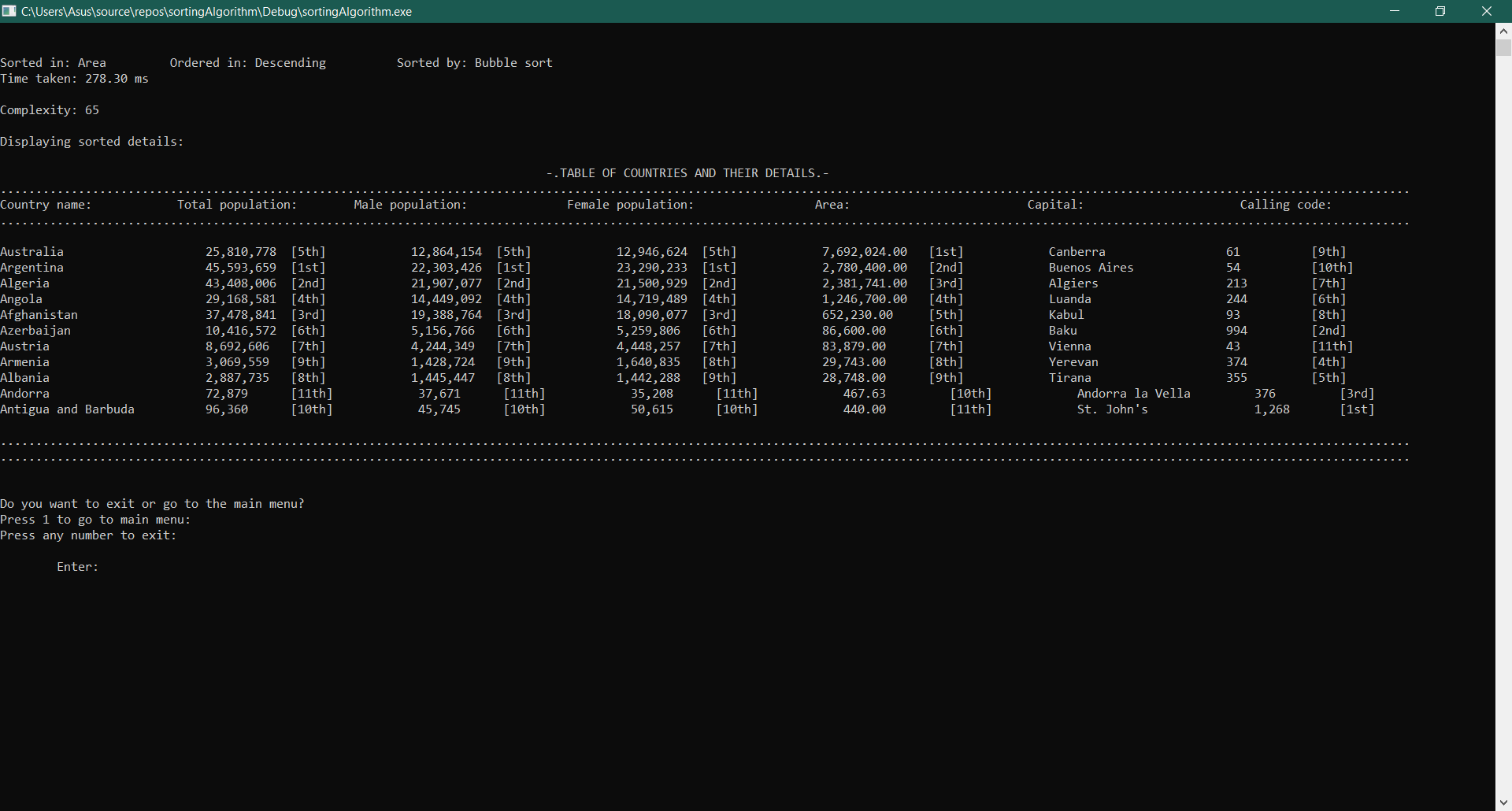
Table sorted by: Area

Figure 7.3.10 DSC area sort

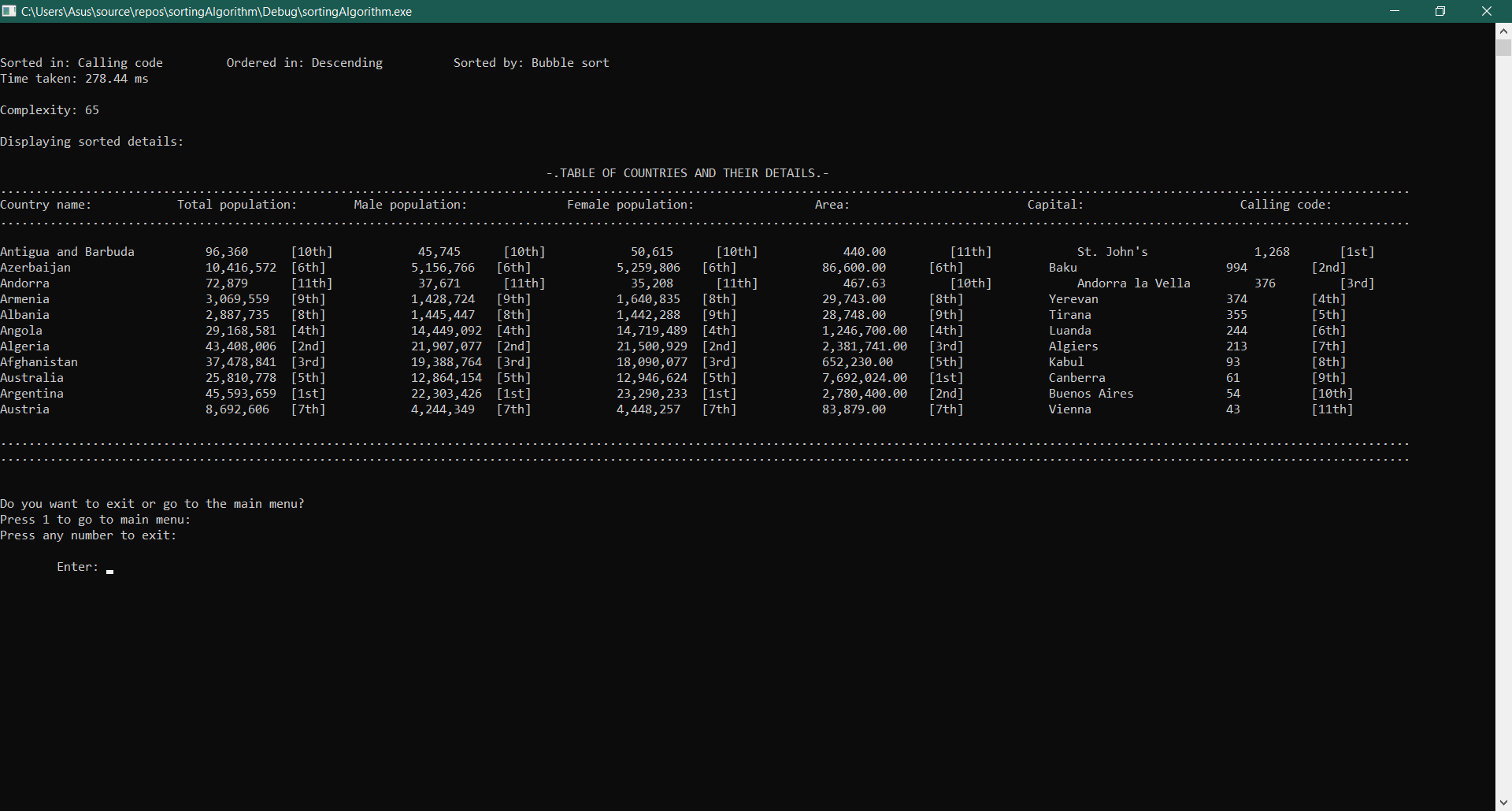
Table sorted by: Calling code

Figure 7.3.11 DSC calling code sort

**Searching: *Time and complexity comparisons***

Linear search

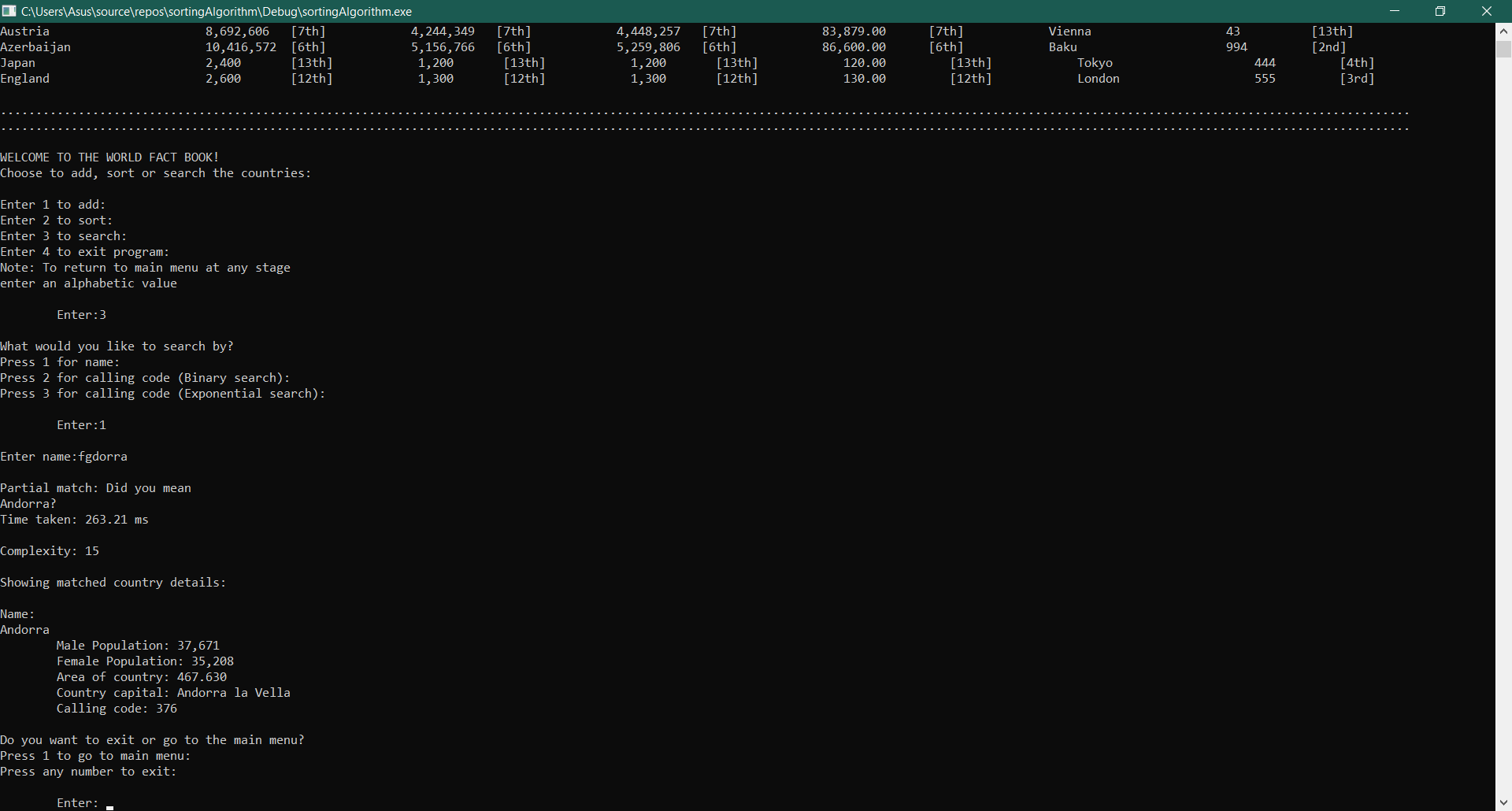


Figure 7.4. Linear search test

Binary search

Figure 7.5. Binary search test

Exponential search

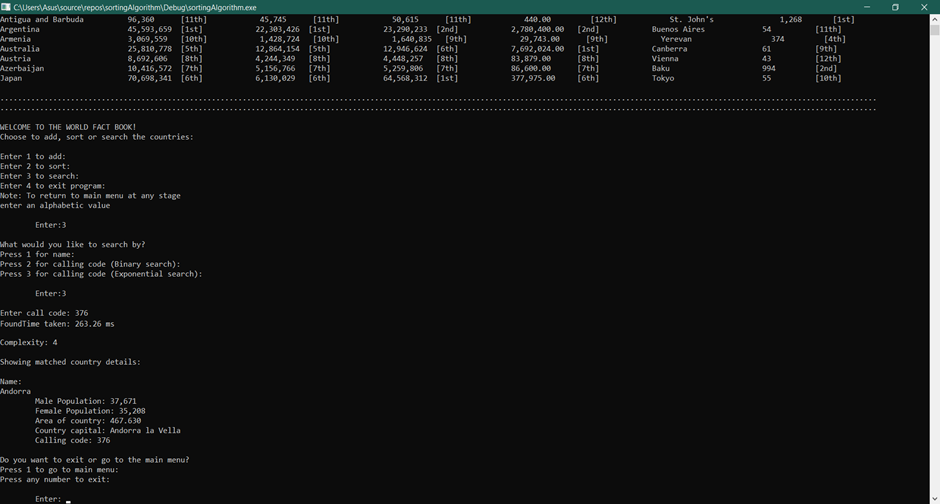


Figure 7.6. Exponential search test

Comparison between the searching algorithms;

Test case: Andorra

|  |  |  |  |
| --- | --- | --- | --- |
| Search type | Linear | Binary | Exponential |
| Time taken | 263.21 | 263.24 | 263.26 |
| Complexity | 15 | 3 | 4 |

Analysis: Sorting

*Complexity and time:*

It is difficult to compare complexities and time in this case since there is a string search and an integer one. However all across, we see linear search as least efficient. In this case, although generally exponential search is best, binary search seems to take its place. Why is this? This happens because the jumping by the double of the value and calculating it takes more resources than divide and conquer approach alone. This is due to the lower amount of input and the placement of this particular value to be found

Testing:

Testing Environment;

The following tests were carried out in a computer with windows 10 operating system and the recommended compiler IDE ‘Visual studio’ with 2017 version.

System testing:

System testing is done by checking the finished product of code against all possible inputs and analysing the output, if it is the one desired or not. This type of testing concerns itself with how the user experiences the application. It is a black box testing method (Guru99, 2020).

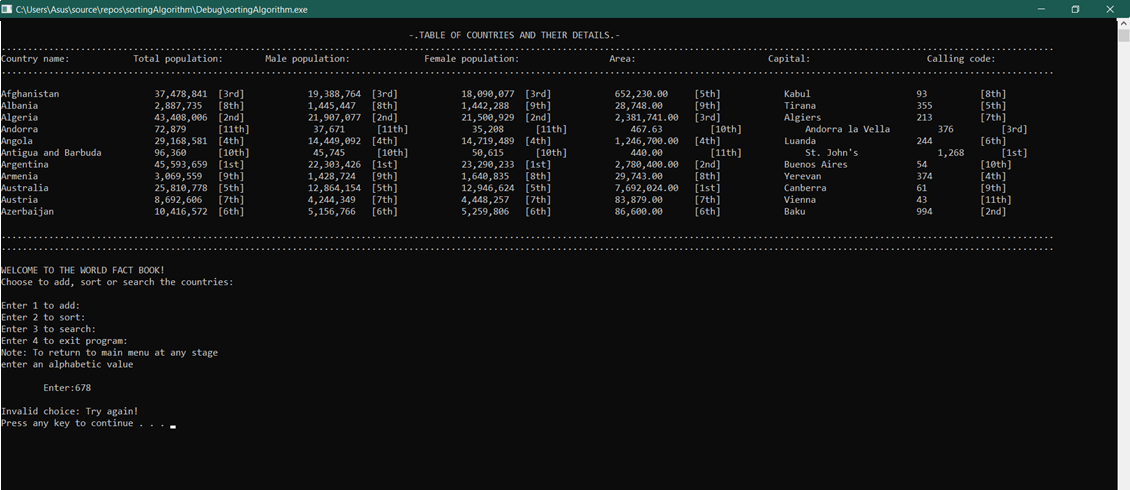
*Test cases*

**(**Note: Correct implementation of inputs is seen in the video demo**)**

**Test Group:** Inputting wrong options.

**Pre-Conditions:** User should select an option not listed in given option

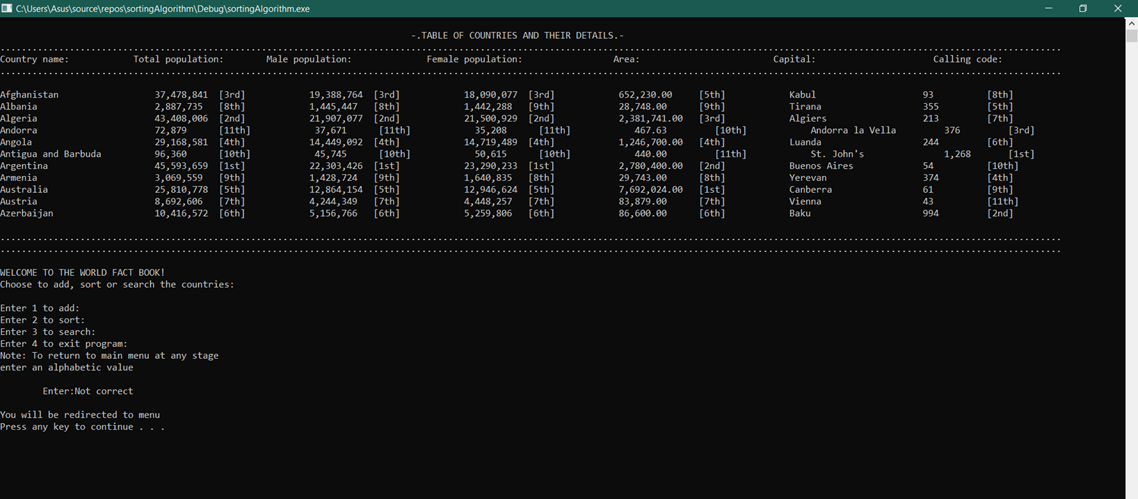
**Current test case:** Wrong options in main menu.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 1 | 1 | User enters wrong options in main menu | 678 | Redirect back to menu | Pass |

Figure 8. Error handling for main menu

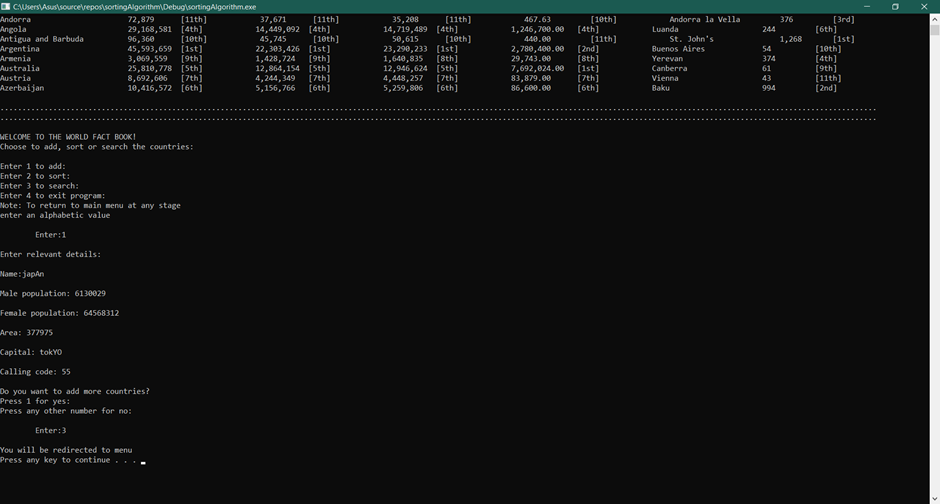
**Current test case:** Wrong options in main menu.

v

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 1 | 1.2 | User enters wrong options in main menu | “Not correct” | Redirect back to menu | Pass |

Figure 9. Error handling for main menu

**Current test case:** Wrong casing of country name/capital in add menu.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 2 | 2.1 | User enters wrong casing when entering country name and capital | 1, “japAn”, 6130029, 64568312,  “tokYO”, 55, 3 | Redirect back to menu and see new country added case corrected | Pass |

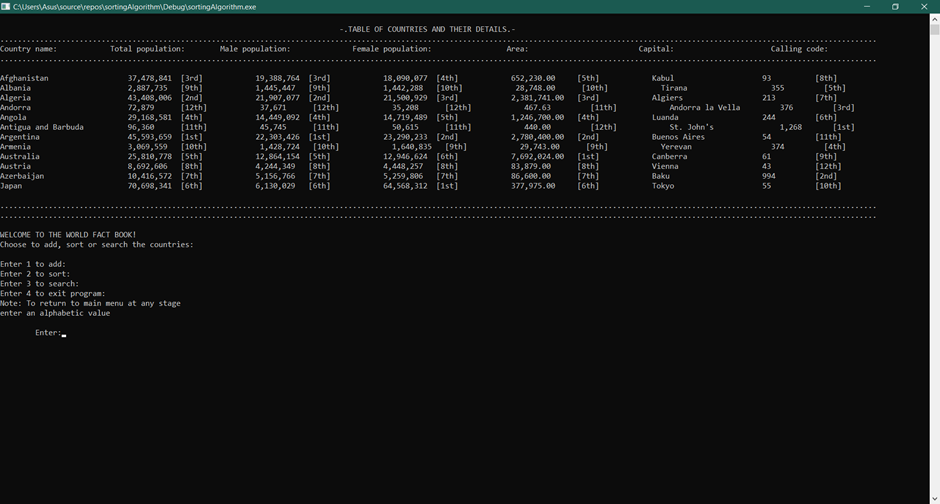
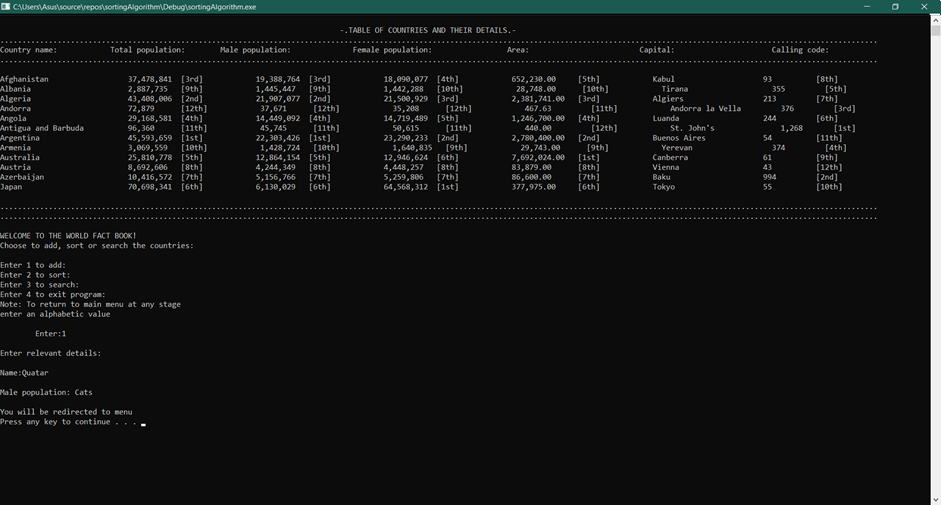
Figure 10. Error handling for add menu

Figure 11. Error handling for add menu

**Current test case:** Wrong casing of country name/capital in add menu.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 2 | 2.2 | User enters wrong options in add menu | 1, “Quatar”, “Cats” | Redirect back to menu and see new country not added | Pass |

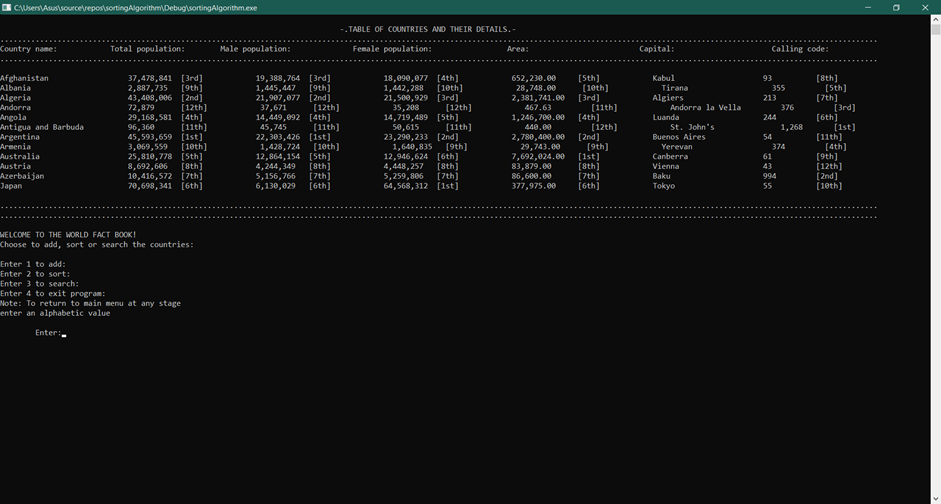
Figure 12. Error handling for add menu

Figure 13. Error handling for add menu

**Current test case:** Wrong choice in sort menu.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 3 | 3.1 | User enters wrong options in sort menu | 2, -3, 6, 5 | Redirect back to question till correct answer is given | Pass |

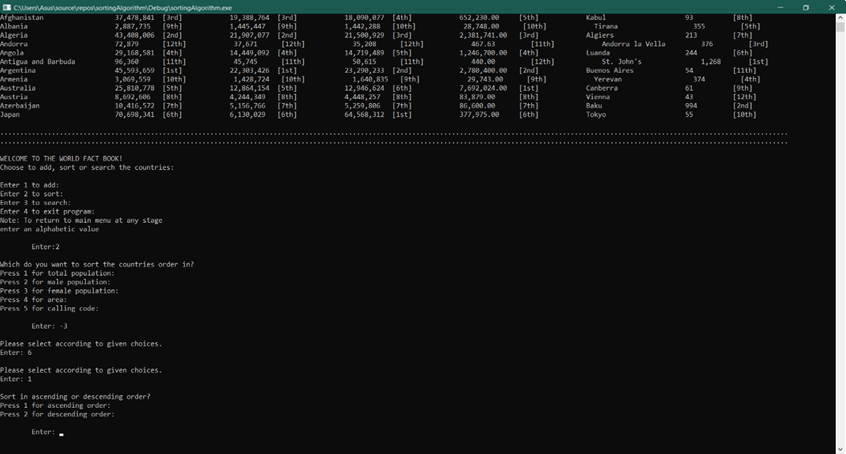


Figure 14. Error handling for sort menu

**Current test case:** Wrong choice in sort menu.

**e**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 3 | 3.2 | User enters wrong option in sort menu | “banana” | Redirect back to menu | Pass |

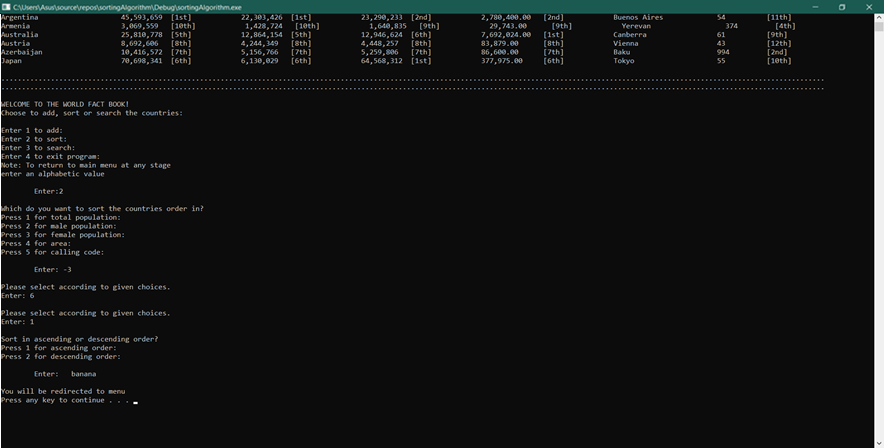
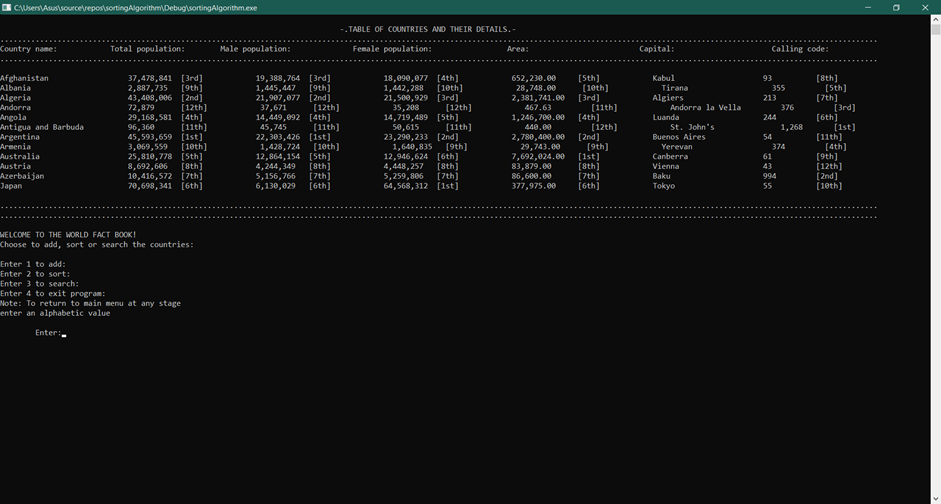
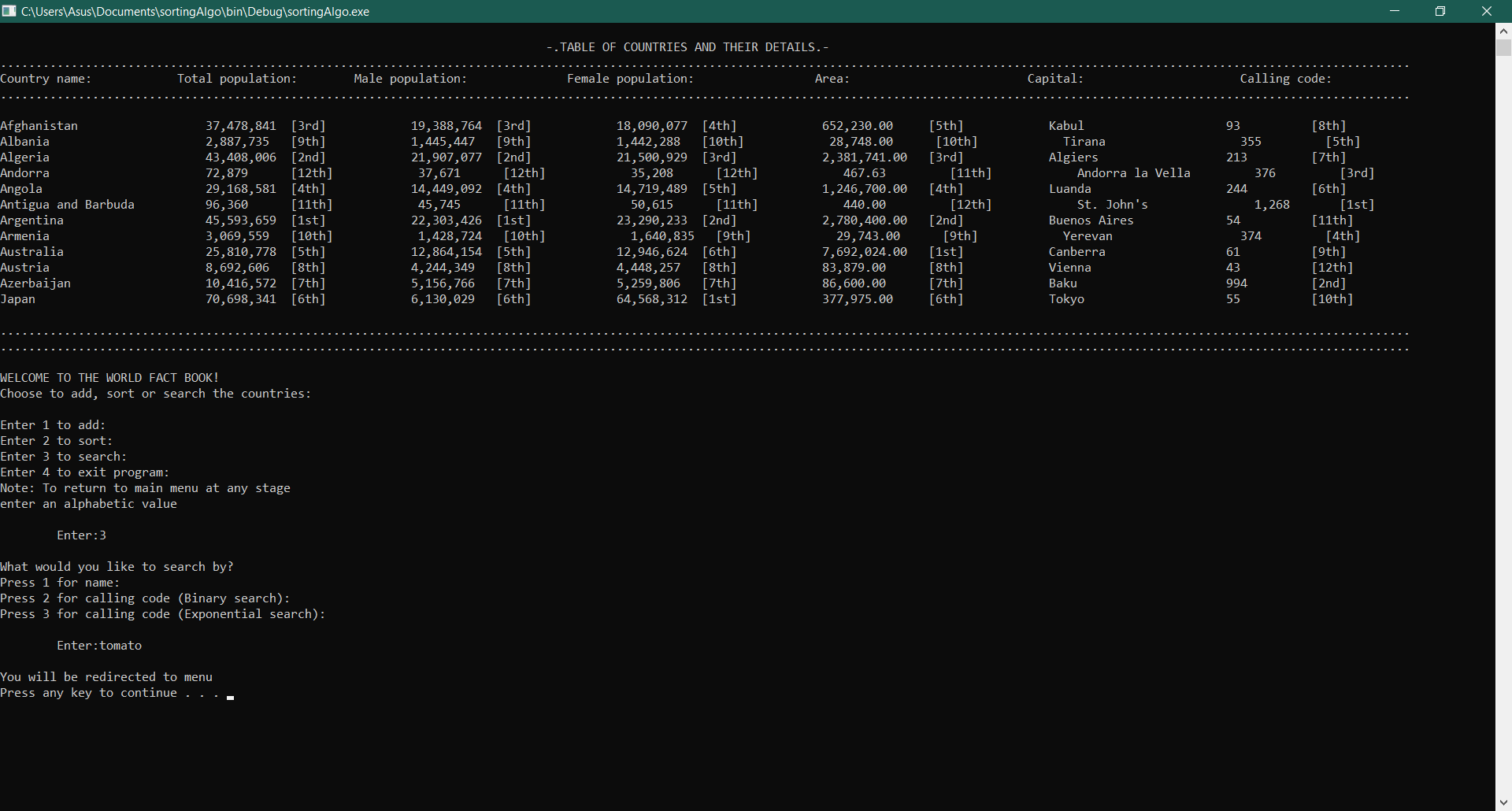


Figure 15. Error handling for sort menu

**Current test case:** Wrong choice in search menu.

******e**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 4 | 4. 1 | User enters wrong option in search menu | “tomato” | Redirect back to menu | Pass |

Figure 16. Error handling for search menu

**Current test case:** Wrong choice in search menu (Name search).

**e**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 4 | 4. 2 | User enters wrong option in name search | “hytgfr” | Ask to redirect to menu or exit | Pass |

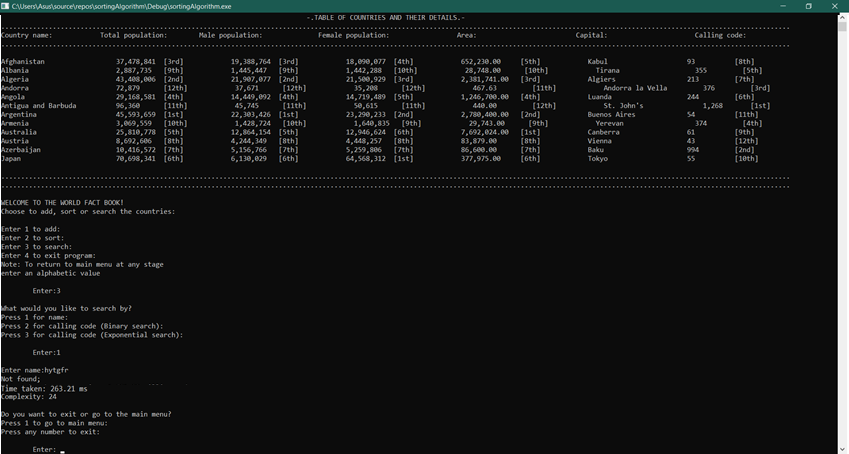
****

Figure 17. Error handling for name search

**Current test case:** Wrong choice in search menu (Name search).

**e**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 4 | 4. 3 | User enters wrong option in name search | “incoherencyngola” | Get partial match then ask to redirect to menu or exit | Pass |

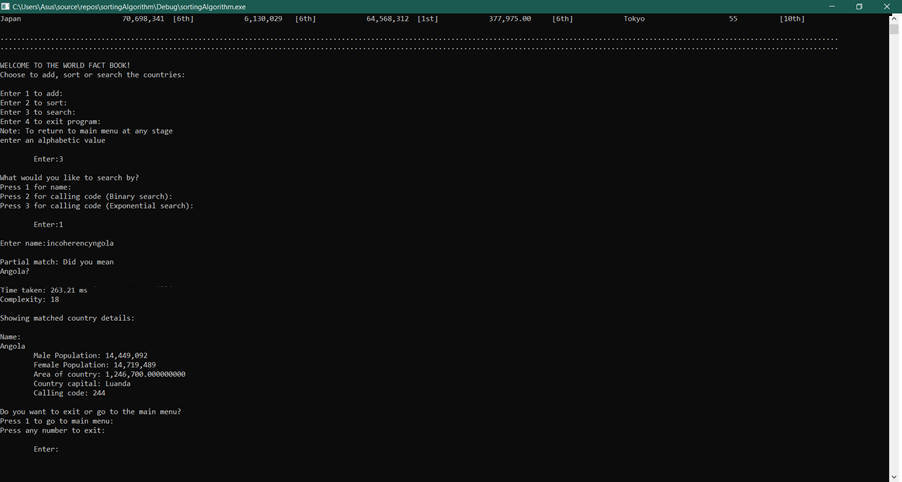


Figure 18. Error handling for name search

**Current test case:** Wrong choice in search menu (Binary call code search)

**e**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 4 | 4. 4 | User enters wrong option in binary call code search | 444 | Ask to redirect to menu or exit | Pass |

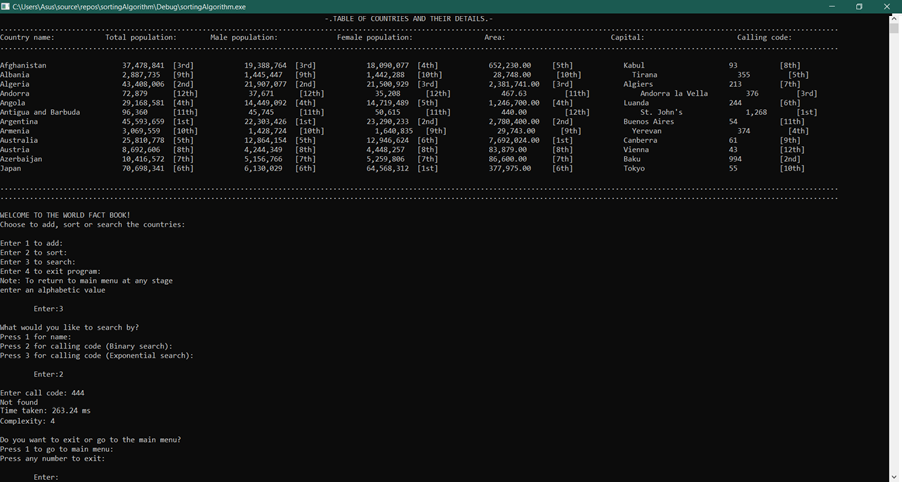
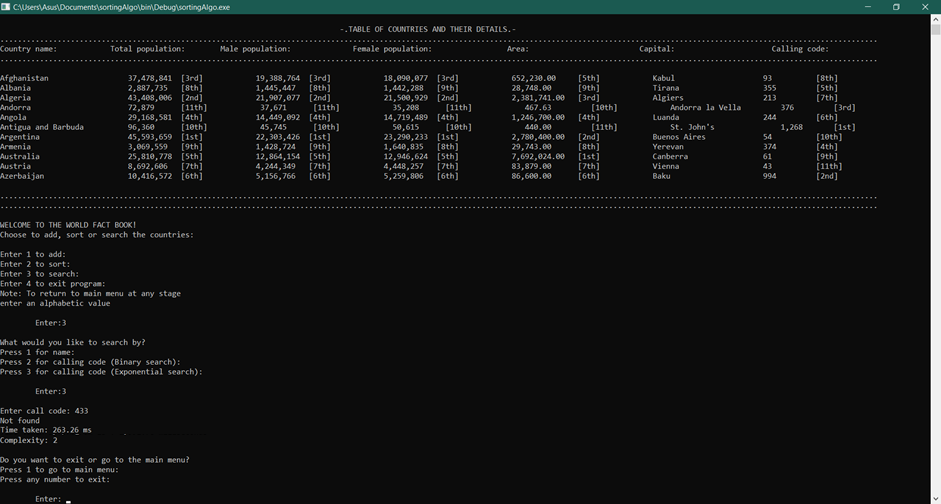


Figure 19. Error handling for binary search

**Current test case:** Wrong choice in search menu (Exponential call code search)

**e**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Group | Case | Description | Input Values | Expected Results | Status |
| 4 | 4. 5 | User enters wrong option in exponential call code search | -34 | Ask to redirect to menu or exit | Pass |

Figure 20. Error handling for exponential search

Unit testing,

Sorting:

Unit testing is a white box testing method, that is used to check if functions use sound logic and get correct output in development. It focuses on the internal operations (Guru99, 2020). This program uses testing to see if some needed conditions are met. In this case, it checks whether the array containing particular country details is completely sorted. This is later removed.

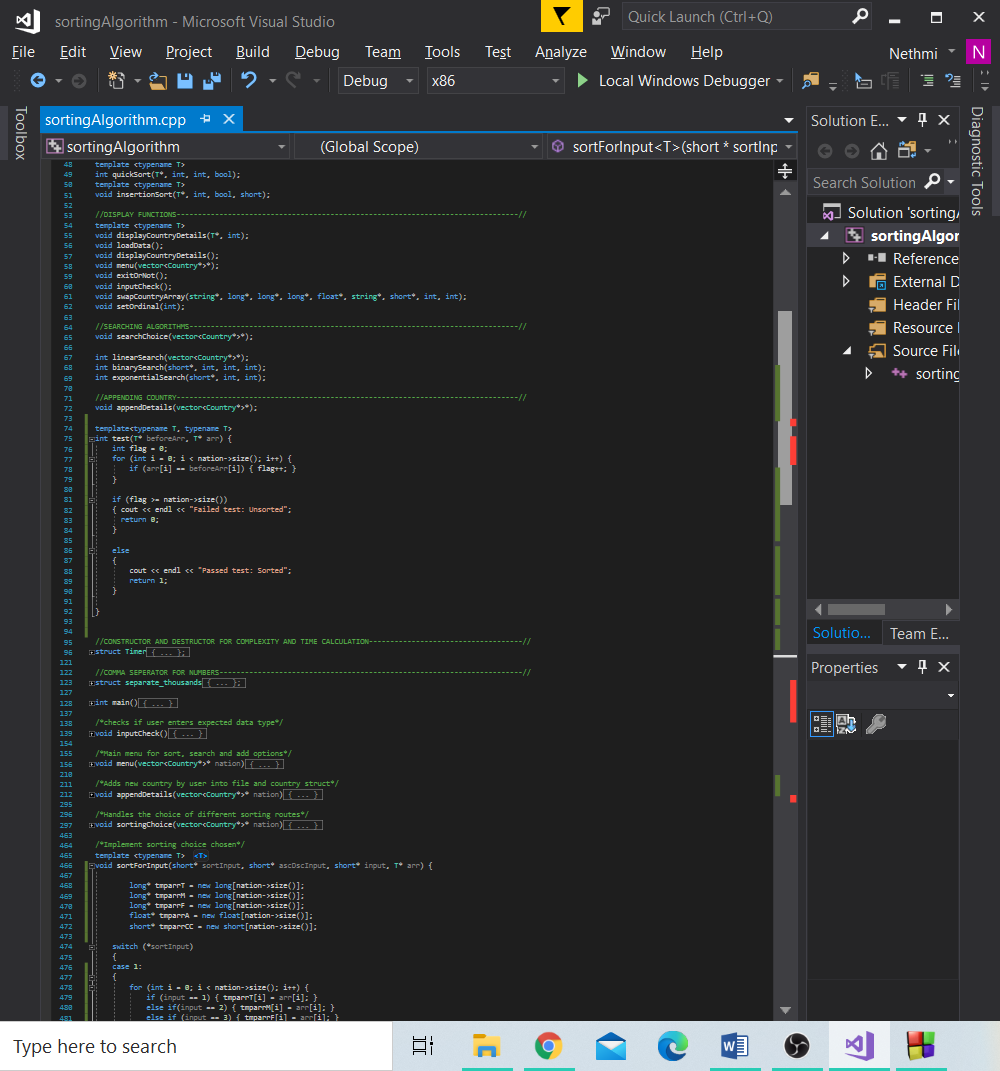


Figure 21. Unit test function for sorting

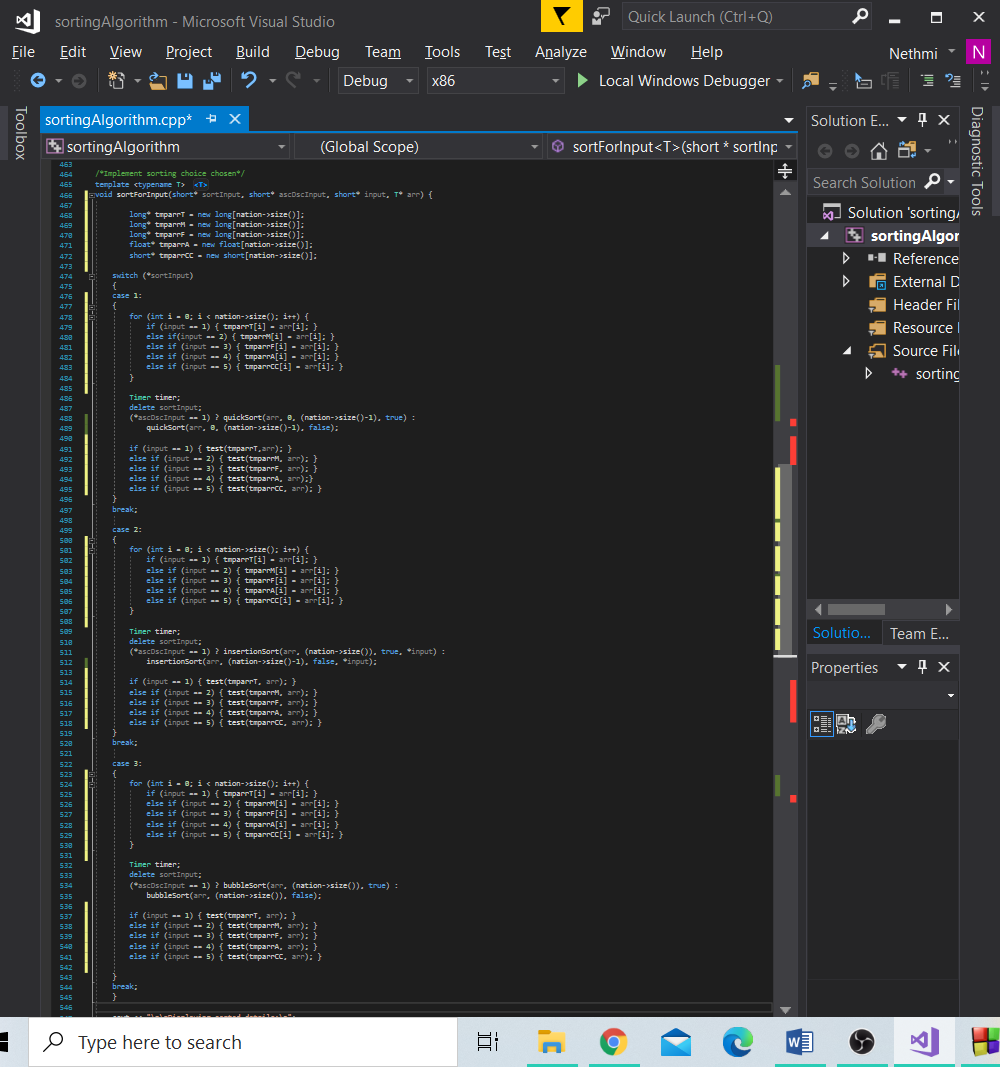


Figure 22. Applying unit test function

Searching:

As detailed above we know the definition of unit testing, below is testing code for linear search, binary search and so respectively.

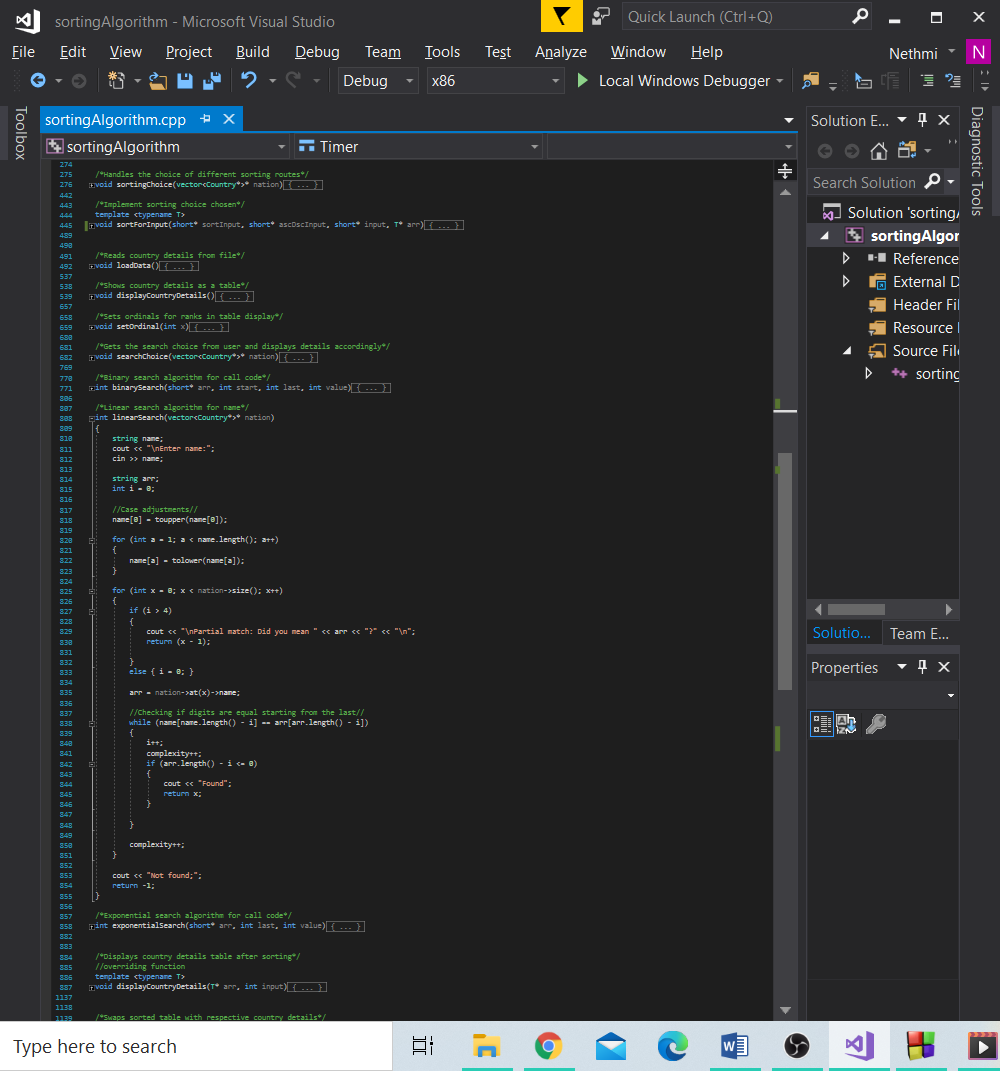


Figure 23. Unit testing for linear search

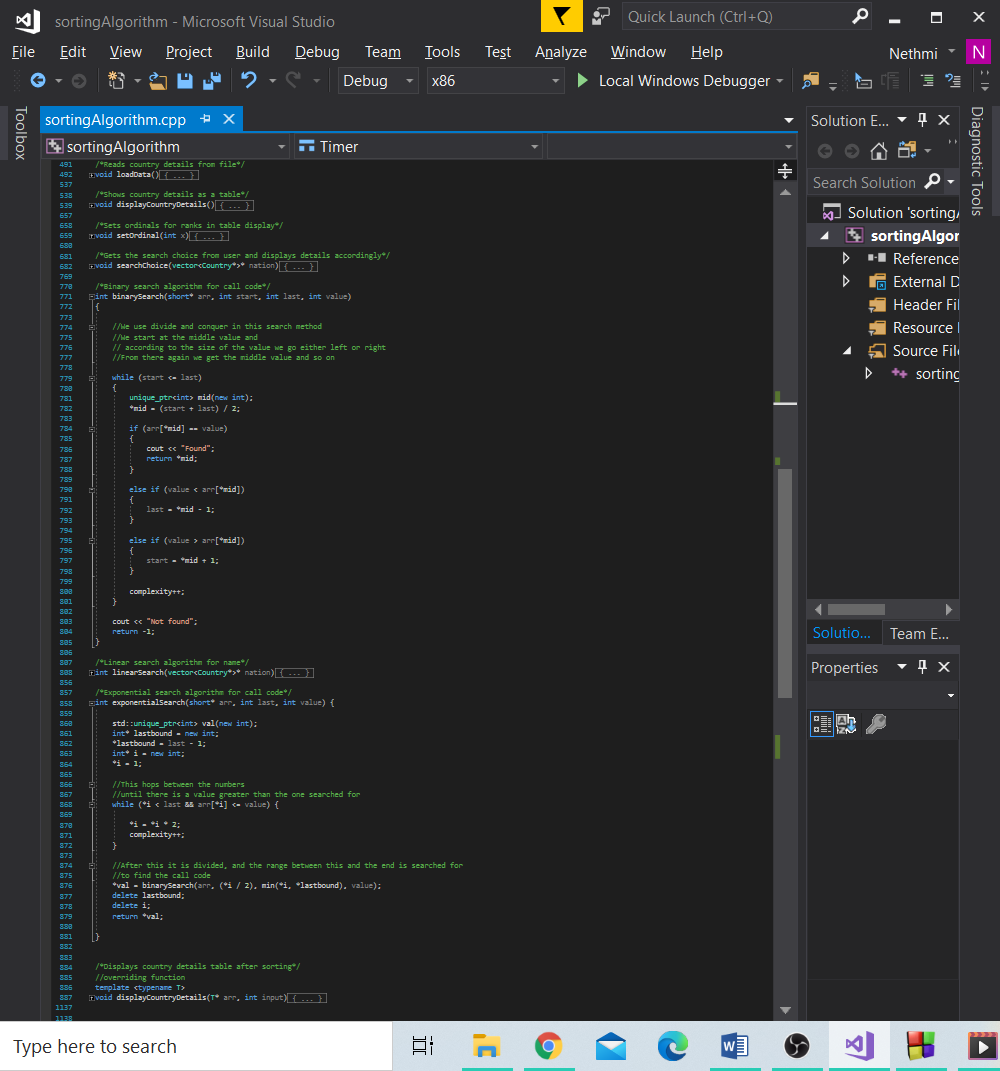


Figure 24. Unit testing for exponential search

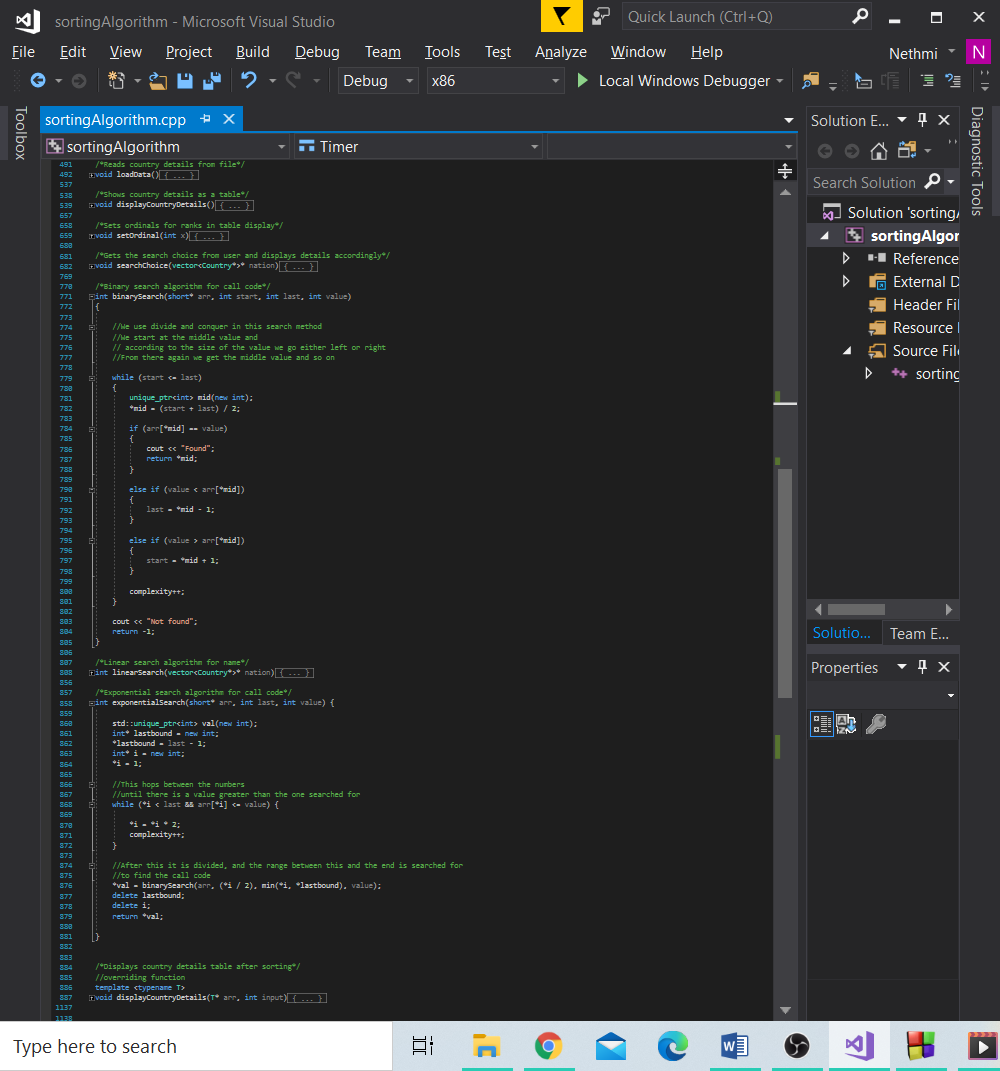


Figure 25. Unit testing for binary search

Limitations:

* Time varies between different systems (since real time is taken in Chrono library and CPU time is not). Moreover, there is a possibility because of that, a lag in performance of a compiler could wrongly variate time. Potential caching problems may arise (Jogojapan, 2012).
* For instance, in exponential search, if value is the foremost in the sorted array, (the smallest) complexity could appear 0 since no comparisons are done (Can be seen in the case of Austria, call code 43).

This is because it would not go through the loop for jumping values but directly to binary search where it is immediately found as its range is limited to itself. Binary search has start (1/2) to the max of (1). Loops are broken before complexity can be incremented. We don’t count the comparison of the value with the value itself.

* Unit testing is used to check validity, but not comparisons.

Future Enhancements:

* Use CPU time measurement with ctime library or QueryPerformanceCounter.
* Have complexity incremented at the start of the loops so the comparison of the value with the value itself is counted.
* Better unit testing system.
* Create an option to allow for screen colour and text colour to be chosen so as to not trigger any epilepsy from black/white contrast.

Conclusion:

The program is one for a world fact book, it uses functions, structures, arrays, vectors, pointers, constructor and templates to achieve this end.

References:

Cormen, T. H. (2013) Algorithms unlocked. Cambridge, Massachusetts: The MIT Press.

Voiland, A. (2017) How Algorithm Got Its Name [Online]. Available at: <https://earthobservatory.nasa.gov/images/91544/how-algorithm-got-its-name> (Last accessed: June 8 2020).

Guru99. (2020) What is System Testing? Types & Definition with Example [Online]. Available at: <https://www.guru99.com/system-testing.html> (Accessed: June 10, 2020).

Jogojapan. (2012) Measuring CPU Time, Boost and C++11 [Online]. Available at: <http://jogojapan.github.io/blog/2012/11/25/measuring-cpu-time/> (Accessed: June 11, 2020).