



SEARCHING FOR THE MINIMUM ELEMENT IN A VECTOR USING BUILT-IN FUNCTION `min_element()` AND LOCATING THE MINIMUM ELEMENT FROM THE VECTOR USING POPULAR SEARCHING TECHNIQUE LINEAR SEARCH METHODOLOGY, FURTHER ANALYZING THE TIME COMPLEXITY OF THESE ALGORITHMS – A CASE STUDY

G.V Karthihesh¹, A.Krishang²

¹6789, ²6784

Class- XII 2025-26, Sainik School Amaravathinagar

Post: Amaravathinagar, Udumalpet Taluka, Tirupur Dt, Tamilnadu State

ABSTRACT

Searching for minimum values in data sets is ubiquitous in computer science and software development. Being able to easily find the smallest value is especially important for sorting and optimization algorithms, for scheduling and for data analysis.

Here, two methods for finding minimum values in C++ vectors are compared. The first is an STL (Standard Template Library) function, `min_element()`, and the other is the Linear Search method. The methods are compared in terms of their implementation and the way they work. Their computational complexities are compared too.

Both methods are expected to have the same time complexity in practice, that is $O(n)$. However, they differ in the degree to which they are convenient for programming and maintainability compared to each other.

KEYWORDS: Vector, Linear Search, STL, `min_element()`, Time Complexity, Algorithm Analysis, C++.

1. INTRODUCTION

Searching is a fundamental process in computer science and has multiple uses in terms of working with and optimizing code. One of the types of searching that many applications require is the least element of a data structure. The searching process is essential to performing sorting algorithms, performing statistical calculations and creating decision-making systems.

A vector is a dynamic collection of data that allows for efficient management of data by allowing for dynamic data sizes (grows and shrinks). A vector provides the means of finding the least value via two means; through the use of a library search function or through a standard searching algorithm. In C++, there is a function in the STL called `min_element()` which provides the simplest method for retrieving the least value from a vector.

An assessment is made comparing the two different approaches for exposing the least element of a vector through qualitative means. This examination does not just focus on understanding how both approaches work in producing the least element of a vector, but also evaluates their efficiency and evaluates both time and space complexity of each approach. This assessment clearly illustrates that both approaches produce the same result and will operate in $O(n)$ linear time. Therefore, either approach will provide an efficient solution for determining the least element in an unsorted vector.

2. RELATED WORK

Efficient searching methods are essential when managing and designing algorithms using data. The linear search, or linear

scan, is considered a basic way of searching data in sequence. Generally, this method is used with unsorted data. Most modern computer programming languages supply built-in linear search functions. For example, the C++ Standard Template Library (STL) provides the `min_element()` function to find the minimum value within a collection known as a vector. Reports indicate that both linear search and minimum value search methods have equal time complexity when used on unsorted vectors, $O(n)$; therefore, all items must be visited during the search process. The current paper will examine the implementation of both the linear search and minimum value search methods and provide an estimation of the time required to perform each method.

3. THEORETICAL BACKGROUND

The search operation is one of the essential operations in the field of computer science and consists of the operation of locating the required information within the collection of data. An example of this type of operation is the process of finding the least significant number in a collection of numbers or the minimum (min) number in the dataset. This process is often required for solving optimization problems or sorting and statistical calculations and in any application that analyzes or processes data.

The vector is a dynamic array that contains multiple data elements stored in contiguous locations (physically located near each other). Because of this characteristic, retrieval and traversal of the data items in the vector is done quickly and efficiently. To find the least number in an unsorted vector, each data item must be evaluated and checked against the last found



least number. In other words, the vector must be completely traversed (i.e., all data elements) so that all of the values are checked against the least number.

In C++, the C++ Standard Template Library (STL) provides a function called `min_element()` that takes as input a range of values (the beginning and ending indexes of the data items) and returns an iterator (pointer) to the least number of all the numbers in that range. In this implementation, the function performs a sequential comparison of all the data items in the vector to arrive at the least number using comparisons. In contrast, the Linear Search method manually traverses through the data elements in the vector, performing a sequential comparison of each number in the vector with the last found least number, and updating the least number if a smaller number is found.

Since both techniques require an examination of all the components of the vector for each of the methods exactly one time, they will have a linear relationship (proportional) with respect to their execution time for n (where n equals number of elements in Input data set). This means that both methods have the same time complexity which is $O(n)$ as well as both having the same amount of space needed (Space Complexity) $O(1)$ making either technique efficient and useful for finding the minimum numbers within an unordered number array.

4. METHODOLOGY

This proposed research will compare two methods of finding the minimum number from a vector of numbers, either by using the standard C++/STL library function `min_element()` or by performing a linear search algorithm. Unlike algorithms that sort all the numbers in a set of data, finding the minimum value is meant to efficiently find the minimum number from a set of numbers along with its location in the vector.

To begin the procedure, the user will provide a vector of numbers. In the first case, the standard C++/STL library function `min_element()` will be used to find the location of the Minimum element, which is also a pointer to the value of that Minimum; in addition to returning that Pointer, that pointer will be used in two ways: first, to retrieve the value of the Minimum element; and second, to find the index of the Minimum Element within the vector. Types of use for STL functions to assist in the location of the Minimum element will reduce manual programming work, as well as being able to take advantage of an implementation optimized for STL. The second method to locating the Minimum element will be performed using a Linear Search algorithm. The Linear Search algorithm will begin with the first element of the vector as the current, Minimum number. This algorithm works by comparing each element of the vector against the current minimum element; if a smaller Number is encountered, that Number will replace the current minimum number, and to point of keeping track of that Number will also be updated.

After all the numbers of the vector have been checked, the minimum number and its position have been identified.

For illustration, consider the vector:

Index	Element
0	23
1	15
2	42
3	8
4	16
5	4
6	29
7	10

Throughout traversal, whenever any element is found in the current traversal to be less than the previous recorded lowest element, the current recorded lowest element is updated. After identifying all of the elements by traversing through the list, you would find a minimum value of 4 for the element located at index 5 in the list. Then the result from both methods will be compared and evaluated based upon the level of simplicity as implemented; how the individual methods perform from an execution perspective, and their respective order of execution complexity.

5. APPROACH 1 FINDING THE MINIMUM ELEMENT USING STL `min_element()`

The first method makes use of a pre-existing STL function called `min_element()` which is equipped with the ability to find the minimum number (or “smallest”) element from a range of possible numbers (i.e., an array). Thus using this function, the programmer will save time and effort since they do not need to compare each number themselves, but instead simply call one library function to get back the minimum number in an efficient and accurate manner.

ALGORITHM

To find the minimum integer in a vector (list of integers), you will need to do the following:

1. Create a vector with the integers.
2. Use the `min_element()` function to find the minimum of these integers using the range of the `begin()` and `end()` functions of the vector as arguments.
3. Save the result of the `min_element()` function in an iterator.
4. Dereference the iterator to return the minimum integer in the vector.
5. Display the minimum integer and its index position.

IMPLEMENTATION

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```
using namespace std;
int main()
{
    vector<int> data = {23, 15, 42, 8, 16, 4, 29, 10};

    auto minPos = min_element(data.begin(), data.end());

    cout << "Minimum Element = " << *minPos << endl;
    cout << "Index Position = "
        << distance(data.begin(), minPos);
```



```
return 0;  
}
```

OUTPUT

Minimum Element = 4
Index Position = 5

6. APPROACH 2

LOCATING THE MINIMUM ELEMENT USING LINEAR SEARCH

The second technique uses a traditional way to search for the minimum number from a vector with the Linear Search method. Unlike the STL function `min_element()`, where the comparisons from each of the elements to find the minimum number is done implicitly by STL; the method below that we will implement finds the minimum by treating the first element in the vector as the minimum and then updating that initial minimum by checking each of the other elements against it to determine if a smaller value has been found during our traversal through the vector.

ALGORITHM

Step 1: Create a vector of data elements.
Step 2: Use the first element in the vector as the current minimum.
Step 3: Record where the current minimum was found (this will be used to locate its index position).
Step 4: For each subsequent element from the 2nd through the last element, check that element against the value of the current minimum.
Step 5: If the value being checked is smaller than the current minimum stored value, (i.e. minimum), then store both its value and index position for future reference.
Step 6: Repeat these steps until all elements are compared.
Step 7: Display the minimum number that was found along with where it was found in the vector (index position).

IMPLEMENTATION

```
#include <iostream>  
#include <vector>  
  
using namespace std;  
  
int main()  
{  
    vector<int> data = {23, 15, 42, 8, 16, 4, 29, 10};  
  
    int minimum = data[0];  
    int position = 0;  
  
    for(int i = 1; i < data.size(); i++)  
    {  
        if(data[i] < minimum)  
        {  
            minimum = data[i];  
            position = i;  
        }  
    }  
}
```

```
cout << "Minimum Element = " << minimum << endl;  
cout << "Index Position = " << position << endl;
```

```
return 0;  
}
```

OUTPUT

Minimum Element = 4
Index Position = 5

7. EXPERIMENTAL ANALYSIS

The results show the same output and similar computational behavior in both methods. The STL `min_element()` function internally performs a sequential scan through the vector, which is essentially the same as a Linear Search algorithm. As a result, in an array of length n , both methods require $n - 1$ comparisons.

There is no major difference in asymptotic performance for either method, while STL has better code simplicity and readability compared to Linear Search, as Linear Search gives a better understanding of the algorithmic process and allows further customization of the search logic when needed. Thus, the choice of the two methods depends on programming needs; not on the performance of either method.

8. TIME COMPLEXITY

The performance of an algorithm can be quantified using its Time Complexity which is defined based on the time taken to run an algorithm as a function of the input size. Analyzing how many operations will be executed as the data set becomes larger provides insight into both the efficiency and scalability of an algorithm.

In this paper, we examine the time complexity of two different approaches for finding the minimum element in an unsorted vector: the STL `min_element()` function and the linear search method. Since neither method has prior knowledge about where the minimum element is located, they both require that all elements of the vector be checked in order to correctly identify the minimum.

For an unsorted vector with n elements it is only possible to verify that an element is the minimum by checking each of the other elements. Thus, the number of total comparisons performed by both algorithms would be:

$$[C(n) = n - 1]$$

Where:

- $C(n)$ is the number of comparisons made,
- n is the number of elements in the vector.

Therefore, as the size of the input increases so does the number of comparisons, therefore causing the running time to grow linearly with the size of the vector.

$$[T(n) = O(n)]$$

This means that if the number of elements in the vector doubles, the execution time will about also double.

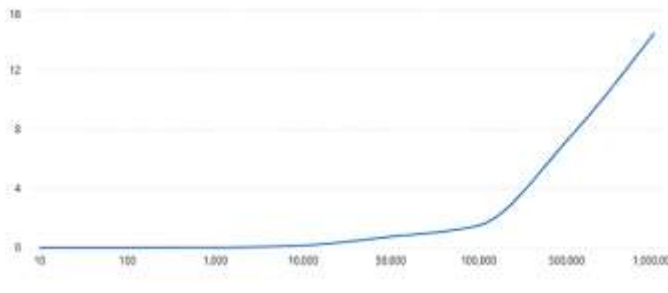


9. RUNTIME COMPLEXITY OF STL `min_element()` AND LINEAR SEARCH

Number of Elements (n)	Comparisons (n-1)	Time Complexity
10	9	0.001
100	99	0.005
1,000	999	0.020
10,000	9,999	0.150
50,000	49,999	0.750
100,000	99,999	1.500
500,000	499,999	7.200
1,000,000	999,999	14.500

10. GRAPHICAL REPRESENTATION Run-Time Growth for Minimum Element Search

Estimated execution time as input size increases. Applicable to both STL `min_element()` and Linear Search.



11. RESULT

Through experimentation, both the STL `min_element()` function and the Linear Search method have been validated to be correct at determining the minimum number in the vector and the position of that number. Both methods produced the same results, thereby showing that all methods determine the smallest number in the right place.

Both methods require a linear time complexity $O(n)$ since it explores the whole vector to determine the lowest number; thus, both methods have a constant space complexity of $O(1)$. The STL `min_element()` method produces a more concise and maintainable implementation, while the Linear Search gives the user increased insight into how the search is done.

12. COMPARATIVE DISCUSSION

PARAMETER	STL <code>min_element()</code>	LINEAR SEARCH
Method Type	Built-in STL Function	User-Defined Algorithm
Implementation Complexity	Simple	Moderate
Lines of Code	Fewer	More
Code Readability	High	Moderate
Ease of Maintenance	High	Moderate
Programmer Effort	Minimal	Higher

Number of Comparisons	$n - 1$	$n - 1$
Run-Time Complexity	$O(n)$	$O(n)$
Space Complexity	$O(1)$	$O(1)$
Flexibility for Customization	Limited	High
Reliability	High (Library Tested)	Depends on Implementation
Industrial Usage	Widely Preferred	Mainly Educational and Custom Applications
Minimum Element Found	Yes	Yes
Index Position Retrieval	Yes	Yes

13. CONCLUSION

The purpose of this research was to compare two algorithms used to find the minimum value within a vector: C++'s STL function `min_element()` and the linear search algorithm. In both cases the algorithm ran from the start of the vector to the end, checking every element. This means both algorithms have an overall runtime complexity of $O(n)$ and a space complexity of $O(1)$ (i.e., they do not store any additional information in memory)..

Even though both algorithms run at the same speed, using STL's `min_element()` function is generally considered to be easier to read, understand, and maintain than using a linear search and therefore is more useful for developing actual software. Using a linear search provides excellent insights into how the searching process takes place and is helpful for learning about searching. Consequently, both methods will successfully locate the minimum value in a vector, but the method you select depends on what you want to use it for, and how you want to write it.

14. ACKNOWLEDGEMENT

This paper would not have materialized without the support, guidance, and encouragement from many individuals and institutions.

I will always remain thankful to Almighty God for giving me strength, courage, and wisdom to carry out and finish this research work.

Words could never express my thanks adequately to my parents for the encouragement, support, and motivation shown throughout this study.

A heartfelt thanks to Captain (IN) Rahul Thiyyath, Principal, Sainik School Amaravathinagar, for an excellent academic culture and for instilling among students an urge towards excellence in the field of studies and research.

All my sincere thanks go to Lt Col Kaushok Nandini Arun, Vice Principal, Sainik School Amaravathinagar, for the honest guidance, encouragement, and support during this research work.

My special thanks and profound gratitude extend to Mr. Praveen Kumar M. Jigajinni, M.Tech (IT), M.Phil (Comp Sc), MCA, M.Sc (IT), ADCA, PGDCA, DCE, Master In-charge,



guide, mentor, and motivator, whose expert guidance, continuous encouragement, valuable suggestions, and critical review led to the successful completion of this research paper. It was through his experience, dedication, and support that I could overcome the challenges faced during the course of this work.

15. REFERENCES

1. <https://www.freecodecamp.org/news/time-complexity-of-algorithms/>
2. <https://www.educative.io/edpresso/time-complexity-vs-space-complexity>
3. <https://en.wikipedia.org/wiki/Space-complexity>