ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

SEARCHING FOR THE MAXIMUM ELEMENT IN A VECTOR USING BUILT-IN FUNCTION *max()* AND LOCATING THE MAXIMUM ELEMENT FROM THE VECTOR USING POPULAR SEARCHING TECHNIQUE LINEAR SEARCH METHODOLOGY FURTHER ANALYSING THE TIME COMPLEXITY OF THESE ALGORITHMS- A CASE STUDY

Kavish A M
Roll No: 6714, Sainik School Amaravathinagar, Taluka Udumalpet, Tiruppur Dist Tamilnadu

ABSTRACT
In Computer Science the most popular algorithms are searching and sorting algorithms. Finding a maximum element from the given vector is the most common application in any domain. To find a maximum element from the vector various strategies can be applied and various languages supports different built-in functions for the same operation.

To find a maximum element in a language generally max() function is implemented, which can be invoked by importing modules in a particular script/ code, which will make easier for an programmer to implement. In contrast using linear search to find a maximum element in a vector needs deep understanding of an algorithm and implementation methodologies.

This manuscript presents a comparative case study of the built-in function max() approach and the linear search method to find an element from the given vector. It analyzes their working principles, performance, computational complexity, and practical usability.

KEYWORDS: *Vector, Maximum Element, Linear Search, Built-in Function, Python, Algorithm Analysis, Computational Efficiency, Time Complexity, Performance Evaluation*

1. INTRODUCTION

Vectors and arrays are among the most fundamental and widely used data structures in computer science, as they allow efficient storage and management of collections of elements. These structures are essential for various computational tasks because of their simplicity, flexibility, and ease of access. One of the most common and important operations performed on vectors and arrays is the identification of the maximum element in a dataset.

The process of finding the maximum value is not only a basic computational task but also a critical operation in many real-world applications due to which numerous of algorithms were built to get optimised results.

It serves as a foundational step in more complex algorithms and decision-making processes. Some of the key application areas include

- **Data Analytics** – identifying the highest value or peak data point in datasets
- **Machine Learning Pre-processing** – normalization and scaling of features
- **Ranking and Decision Systems** – selecting the best or highest-scoring option
- **Optimization Problems** – determining optimal solutions based on maximum values

Owing to its wide applicability, it is important to implement this operation efficiently and accurately, especially when dealing with large-scale datasets.

Two primary approaches can be used to solve this problem. The first approach involves the use of built-in functions provided by modern programming languages, such as **max()** in Python and **max_element()** in C++. These functions are designed to be efficient, concise, and optimized for performance, allowing developers to perform the operation with minimal code.

The second approach involves implementing a manual algorithm, typically using a linear search. In this method, each element in the vector is examined sequentially, and the maximum value is determined through repeated comparisons. Although this approach requires more code and effort, it provides a deeper understanding of algorithmic logic and computational processes.

The built-in function approach offers advantages such as simplicity, reduced development time, and improved execution efficiency. In contrast, the linear search method provides greater flexibility, transparency, and educational value, making it useful for learning and customization.

This research paper focuses on a detailed comparative analysis of these two approaches. It evaluates their

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

performance, usability, and flexibility and examines their computational behaviour. Special attention is given to practical implementation aspects, particularly in Python, to understand how built-in optimizations influence overall performance.

2. IMPORTANCE OF FINDING MAXIMUM ELEMENT

Efficient identification of the largest element within a dataset is of paramount importance across various domains of computer science and practical applications. Although this task may appear straightforward, it constitutes a fundamental component in numerous computational processes, decision-making systems, and data-driven applications. The significance of this task is particularly pronounced when handling extensive datasets, where considerations of both performance and speed are critical.

Key Applications Include

(a) **Data Analysis**
In the field of data analytics, identifying the maximum value within a dataset is crucial for recognizing peak data points, trends, and outliers. For instance, it can be employed to ascertain the highest recorded temperature, the maximum sales achieved, or the peak usage within a system. Such information is vital for making informed decisions and comprehending patterns in the data.

(b) **Educational**
In academic settings, the computation of maximum values is employed to ascertain the highest grades, rankings, or performance levels of students. This process aids institutions in efficiently evaluating top performers and analysing academic achievements.

(c) **Financial:**
In the domains of finance and business, the identification of maximum values is essential for the analysis of profits, stock prices, and market trends. For example, investors frequently monitor the peak stock value over a given period to inform strategic investment decisions. Similarly, businesses employ this analysis to ascertain maximum revenue or profit.

I. **Machine Learning and Data Science:**
In machine learning, finding the maximum element is frequently used in pre-processing steps such as normalization and feature scaling. It helps in transforming data into a standardized range, which improves the performance and accuracy of machine learning models. Additionally, it is used in algorithms that rely on selecting the best or most significant feature.

II. **Optimization and Decision-Making Systems**
Many optimization problems require identifying the best possible outcome from a set of values. The maximum element often represents the optimal solution, such as the highest efficiency, maximum output, or best choice among alternatives.

3. EXISTING SYSTEM FOR LOCATING THE MAXIMUM ELEMENT

Traditionally, the task of identifying the maximum element within a vector is addressed through a manual algorithm referred to as the linear search method. This technique entails traversing each element in the dataset in a sequential manner and consistently updating the maximum value based on comparisons. It is regarded as one of the most straightforward and essential algorithms introduced in computer science, leading to its widespread application in fundamental programming and educational settings.

Example

Vector = {3, 7, 2, 9, 5}

Steps Involved:

- Assume the first element as the initial maximum
- Compare this value with each subsequent element in the vector
- If a larger element is discovered, update the maximum value
- Continue this procedure until all elements have been evaluated
- The final value obtained is the maximum element

This method guarantees that every element in the dataset is scrutinized, rendering it dependable and precise for all varieties of input data.

4. PROPOSED SYSTEM: BUILT-IN FUNCTION APPROACH

Programmers can effectively find the greatest element in a vector without the need for manual implementation thanks to built-in methods in modern programming languages. These features are intended to guarantee excellent performance and dependability while streamlining the coding process. Developers can utilize these predefined functions directly to accomplish the desired outcome rather than explicitly implementing the logic for comparison and iteration.

For Instance

- Python → max() max_element() in C++
- Collections.max() in Java

(a) **Principle of Operation**

Internal operations that resemble the linear search method are carried out via built-in functions. But their application is where the main distinction is found. These functions are embedded into the language runtime or standard libraries and written in low-level languages (like C or optimized libraries). They therefore run more quickly and effectively than code that is developed by hand.

(b) **Benefits of Integrated Features:**

• Less Work in Coding

With just one line of code, programmers can determine the maximum value, saving time and effort throughout development.

• Enhanced Readability

Maintainability is enhanced by shorter, clearer, and simpler code.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

• Excellent Performance Particularly in high-level languages like Python, built-in functions frequently outperform human implementations due to their speed and efficiency optimization.
• Reduced Chance of Error
The likelihood of programming errors is much decreased because the logic is pre-defined and tested.
• Improved Use of Resources
These functions are appropriate for huge datasets since they are made to use CPU and memory resources effectively.

(c) **Python-Specific Benefit**

The max() function in Python can run more quickly than a manually written loop because it is implemented in C. Furthermore, Python's max() method offers additional functionalities like:
• Capacity to manage various data kinds
• Custom comparison utilizing the key parameter is supported.
• Effective management of big datasets
Because of their adaptability, built-in functions are more potent than straightforward implementations of linear search.

5. METHODOLOGY: IMPLEMENTATION

This section explains how to utilize both the built-in function technique and the linear search method to locate the greatest element in a vector. The approach is made to be easy to use, effective, and compatible with a variety of programming languages.

Common Algorithm (Accepted by all languages)

The fundamental reasoning behind figuring out the maximum element is represented by the following procedure. Any computer language can be used to build this universal algorithm.

FUNCTION FIND_MAX(V)
1. *Take first element as max*
2. *Check each element in V*
3. *If element > max, update max*
4. *Repeat until end*
5. *Return max*

The technique starts by assuming that the vector's first member is its maximum. Then, one by one, it compares this value with every vector element. The maximum is updated whenever a higher value is discovered. The maximum element is represented by the final value kept in the variable after every element has been verified.
This approach guarantees that every factor is taken into account, making it precise and dependable.

6. PYTHON IMPLEMENTATION

The aforementioned algorithm can be implemented in Python using both built-in and manual methods.

(a) Using Built-in Function(Simple Method):

```
number =[3,7,2,9,5]
print(max(numbers))
```

Reasons
• Makes use of the built-in max() function in Python
• Just one line of code is needed.
• Improved performance through internal optimization

(b) Using Linear Search(Manual Method)

```
numbers = [3, 7, 2, 9, 5]
max_val = numbers[0]
for num in numbers:
    if num > max_val:
        max_val = num
print(max_val)
```

Reason
• Initially, the first figure is assumed to be the maximum
•Iterates over every component
• When a higher value is discovered, the maximum is updated.

7. COMPLEXITY OF ALGORITHM

Analysing an algorithm's complexity is crucial to assessing its effectiveness and performance in computer science. It aids in comprehending how an algorithm responds to growing input sizes and how well it makes use of available system resources. When choosing the best method to solve a problem, complexity analysis is essential, particularly when working with enormous datasets. Two crucial elements are taken into account when solving the problem of locating the maximum element in a vector:

- Time complexity
- Space complexity

8. SPACE COMPLEXITY

The term "space complexity" describes how much memory an algorithm needs to run. It comprises any extra memory utilized for processing, such as variables, temporary storage, and auxiliary data structures, in addition to the memory required to hold the input data. Space complexity analysis is crucial because effective memory use directly affects an algorithm's scalability and performance, particularly when working with big datasets or constrained system resources.

Both the built-in function approach and the linear search method are very memory-efficient when determining a vector's greatest element. To store the current maximum value throughout execution, these methods only need one variable. This variable is modified each time a larger element is met throughout the algorithm's progression. Other than this variable, no complicated data structures or extra memory allocation are needed.

The algorithm uses very little memory overall since it processes the input data without making copies or utilizing additional storage. Because of this, the method is appropriate for situations where memory optimization is crucial.

Therefore, the space complexity of both approaches can be written as follows:

O(1)

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

This constant space complexity shows that the algorithm's memory use stays constant and doesn't grow as the input size does. The approach requires the same amount of extra memory whether the vector has a modest number of elements or a very large dataset.

As a result, both the built-in function and linear search approaches are thought to be very effective at using space, which makes them workable and scalable solutions for actual applications.

9. TIME COMPLEXITY

The term "time complexity" describes how long an algorithm takes to finish running as a function of the size of the input. It is among the most crucial metrics for assessing an algorithm's effectiveness and performance. We may comprehend how the execution time rises as the input dataset's element count increases by examining temporal complexity.

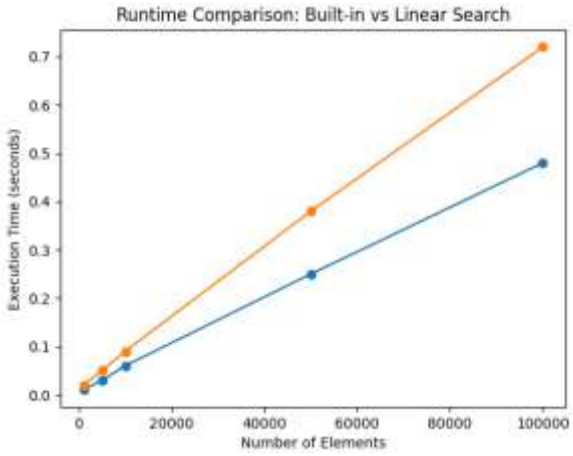
Both the built-in function technique and the linear search method use a similar procedure to discover the greatest element in a vector. To guarantee that the maximum value is accurately determined, each element in the vector must be looked at at least once. This indicates that the dataset is traversed sequentially by the algorithm.

The number of operations rises in direct proportion to the size of the input because each element is examined exactly once. Consequently, the time complexity of both methods is: O(n)
where n is the vector's total number of entries.

10. RUNTIME COMPLEXITY

Number of Elements(n)	Built-in Function Time(sec)	Linear Search Time(sec)
1000	0.01	0.02
5000	0.03	0.05
10000	0.06	0.09
50000	0.25	0.38
1000000	0.48	0.72

Graphical Representation of Runtime complexity of both the methods



The graph shows how the built-in function technique and the linear search method compare in terms of execution time for various input sizes. The vertical axis (Y-axis) displays the execution time in seconds, while the horizontal axis (X-axis) displays the number of elements in the vector.

It is evident from the graph that both approaches exhibit a consistent rise in execution time as the number of elements rises. This suggests that both methods exhibit a linear development pattern, validating their O(n) time complexity.

However, for all input sizes, the built-in function consistently takes less time to execute than the linear search approach. For instance, the built-in function performs noticeably faster than the manual linear search method when the number of elements hits 100,000. The implementation of the built-in function in low-level optimized code, which lowers computational overhead and speeds up execution, is primarily responsible for its enhanced performance. The linear search approach, on the other hand, takes a little longer to execute because it is carried out step-by-step in high-level code.

The fact that the difference between the two approaches widens as the input size increases is another crucial finding. This demonstrates how the built-in function is more scalable and appropriate for managing big datasets.

Overall, the graph makes it abundantly evident that although the theoretical time complexity of both approaches is equal, the built-in function offers superior practical performance, making it the recommended option for real-world applications.

11. ANALYSIS OF EXECUTION TIME

When assessing an algorithm's effectiveness and performance, execution time is crucial, particularly when handling big datasets. It establishes the speed at which an algorithm may finish a task and generate results. Analyzing execution time aids in comprehending the useful distinctions between the built-in function technique and the linear search strategy when determining the maximum element in a vector.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

(a) Built-in Function Approach:

The maximum element in a dataset can be found quickly and easily using Python's built-in max() function. This function can run more quickly than manually written code because it is internally implemented in a low-level language (C).

The integrated function offers optimal performance and lowers the expense of repeated interpretation. It makes use of memory access patterns and effective iteration strategies, which greatly increase execution performance. Because of this, it works well even with big datasets.

(b) Linear search method

In the linear search approach, each vector element is manually iterated through and compared to the current maximum value. Python code is used to carry out this procedure step-by-step.
This approach is slower than the built-in function, despite being straightforward and easy to comprehend. This is due to the fact that every operation is carried out via the Python interpreter, which adds extra overhead. The execution time grows proportionately to the number of items.

(c) Comparative Analysis

Both methods execute in a linear fashion, which means that the time needed rises in direct proportion to the vector's element count. This demonstrates that the temporal complexity of both approaches is (O(n)). But according to the graph and experimental findings:
• Compared to linear search, the built-in function **max()** consistently performs faster.
• As the amount of the dataset rises, the performance disparity widens.
• Built-in features are more scalable and effective.
• Linear search offers more comprehension and adaptability.

12. CONCLUSION

The study comes to the conclusion that both built-in functions and linear search techniques are dependable and efficient ways to locate a vector's greatest element. Each approach ensures accurate findings by following a linear traverse of the dataset, which makes them appropriate for a variety of applications.

However, a thorough comparison shows that they differ significantly in terms of functionality, performance, and usefulness. The speed, ease of use, and efficiency of built-in functions, like Python's max(), are substantial advantages. These functions are very appropriate for real-world applications, particularly when working with huge datasets or performance-critical systems, because they are internally optimized and require less code.

The study also shows that although both methods have the same theoretical temporal complexity of $O(n)$ O(n), implementation-level optimizations cause differences in their actual performance. Because built-in functions handle memory more effectively and have less computational overhead than manual implementations, they perform better. Consequently, it can be said that:

- Built-in features are quicker, more effective, and simpler to use.
- Linear search is more adaptable, personalized, and instructive. Built-in functions are advised in real-world situations to improve performance and minimize development effort.
- Nonetheless, linear search is still a useful method for understanding basic ideas and in circumstances requiring specialized reasoning.

In general, the decision between the two methods relies on the particular needs of the application, striking a balance between flexibility and efficiency.

13.ACKNOWLEDGEMENT

Apart from the efforts of me, the success of any work or project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this research paper.

I express deep sense of gratitude to almighty God for giving me strength for the successful completion of the research paper.

I express my heartfelt gratitude to my parents for constant encouragement while carrying out this research paper.

I express my deep sense of gratitude to **The Principal, The Vice Principal and The Administrative Officer** who have been continuously motivating and extending their helping hand to us.

My sincere thanks to **Mr. Praveen Kumar Murigeppa Jigajinni**, Master In-charge, A guide, Mentor and great motivator, who critically reviewed my paper and helped in solving each and every problem, occurred during implementation of this research paper.

14. REFERENCES

[1] T. H. Cormen, Introduction to Algorithms, MIT Press.
[2] Python Documentation – Built-in Functions