ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 9| Issue: 12| December 2023|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2023: 8.224 || ISI Value: 1.188

FINDING AN INDEX OF GIVEN ELEMENT OF A VECTOR USING *find_vect()* SEQUENTIAL SEARCH METHODOLOGY WITH FINDING AN ELEMENT USING *index() method* OF A LIST. FURTHER COMPARISON BETWEEN find_*vect()* OF SEARCHING AN INDEX WITH FINDING AN ELEMENT USING *index()* METHOD OF LIST. - A CASE STUDY

6837 – Cadet A Vishnukumar[1]
Class- XII 2021-22, Sainik School Amaravathinagar
Post: Amaravathinagar,Udumalpet Taluka,Tirupur Dt,Tamilnadu State

ABSTRACT

The abstract of the comparison between find_vect() sequential search methodology and the index() method of a list for finding the index of a given element is as follows:

Sequential search is a commonly used method to search for an element in a vector. The find_vect() function employs sequential search to locate the index of a given element within a vector. On the other hand, the index() method of a list in Python provides a direct way to find the index of an element.

This comparative analysis focuses on the efficiency and performance of these two methodologies in terms of finding an element's index. The find_vect() function iteratively searches through the vector until it locates the element, while the index() method directly returns the index of the element from the list.

In terms of time complexity, the find_vect() sequential search methodology has a time complexity of O(n), where n is the size of the vector. This means that as the size of the vector increases, the searching process becomes slower due to having to iterate through each element. On the other hand, the index() method of a list has an average time complexity of O(n), but it can perform significantly faster, especially for large lists, due to its internal implementation.

However, the index() method is only available for lists in Python, whereas the find_vect() function can be implemented for both vectors and lists. Additionally, if the vector or list is unsorted, the find_vect() method can still find the index of the element, while the index() method assumes the list is sorted.

In conclusion, the find_vect() sequential search methodology provides a versatile approach for finding the index of a given element, suitable for unsorted vectors or lists. On the other hand, the index() method of a list offers a direct and potentially faster option, especially for large lists or sorted data. The choice between these methods depends on the specific requirements and characteristics of the data being searched.

KEYWORDS: ***find_vect(), sequential search, index(), vector and list.***

1. INTRODUCTION

In this analysis, we aim to compare two methodologies for finding an index of a given element in a vector or a list. The first methodology is a sequential search approach called find_vect() specifically designed for vectors, while the second methodology involves the use of the index() method of a list to find an element

2. SEQUENTIAL SEARCH METHODOLOGY USING FIND_VECT()

Sequential search is a simple and straightforward algorithm that involves iterating through each element of a vector to find the desired element. The custom function `find_vect()` implements this methodology by comparing each element sequentially until a match is found. The index of the matched element is then returned. If the element is not found, a specific value (e.g., -1) can be returned to indicate the absence of the element in the vector

3. COMPARISON WITH `INDEX()` METHOD OF A LIST:

Python's built-in `index()` method provides a convenient way to find the index of an element in a list. It returns the index of the first occurrence of the element, and if the element is not found, it raises a `ValueError`. This method internally applies a similar sequential search technique to locate the desired element efficiently.

4. PERFORMANCE & CONSIDERATIONS

While both `find_vect()` and the `index()` method achieve the same goal, their performance may vary depending on the specific use case. `index()` is optimized for lists and offers a simpler syntax, which makes it a popular choice for locating elements. On the other hand, `find_vect()` is a custom function that can be adapted for other data structures like vectors.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 9| Issue: 12| December 2023|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2023: 8.224 || ISI Value: 1.188

PYTHON PROGRAM TO FIND INDEX OF A NUMBER IN A GIVEN LIST (CODE)

```
x=eval(input("enter a list="))
y=0
z=int(input("enter element to find="))
for i in x:
    if z==i:
        print(y)
    y+=1
          b=p
```

PYTHON PROGRAM TO FIND INDEX OF A GIVEN NUMBER IN THE LIST(GENETALISED APPROACH)

```
x=eval(input("enter a list="))
y=int(input("index of the number you want="))
z=x.index(y)
print("index of the number is",z)
```

5. PYTHON'S BUILT IN FUNCTION

Python's built-in index() function is a useful tool for finding the index of a specific element in a sequence. This function takes an argument representing the value to search for and returns the index of the first occurrence of that value in the sequence.

6. ALORITHM TO FIND INDEX OF A LIST

1. Initialize a variable `index` to -1 as the default value if the number is not found.
2. Iterate through the list and compare each element with the target number.
3. If the element matches the target number, set `index` to the current index and break out of the loop.
4. After the loop completes, check if `index` is still -1. If so, the number was not found in the list.
5. Return the value of `index`.

WHEN THE CODE IS

```
def find_index(number, lst):
    index = -1
    for i in range(len(lst)):
        if lst[i] == number:
            index = i
            break
    return index
```

7. COMPLEXITY OF ALGORITHM

In computer science, analysis of algorithms is a very crucial part. It is important to find the most efficient algorithm for solving a problem. It is possible to have many algorithms to solve a problem, but the challenge here is to choose the most efficient one.[1]

There are multiple ways to design an algorithm, or considering which one to implement in an application. When thinking through this, it's crucial to consider the algorithm's **time complexity** and **space complexity**.[2]

8. SPACE COMPLEXITY

The space complexity of an algorithm is the amount of space (or memory) taken by the algorithm to run as a function of its input length, n. Space complexity includes both auxiliary space and space used by the input.[2]

Auxiliary space is the temporary or extra space used by the algorithm while it is being executed. Space complexity of an algorithm is commonly expressed using **Big (O(n))** notation.[2]

The Space complexity is ignored in this research paper, since the space complexity of particular problem is not considered so important.

9. TIME COMPLEXITY

The time complexity of an algorithm is the amount of time taken by the algorithm to complete its process as a function of its input length, n. The time complexity of an algorithm is commonly expressed using asymptotic notations:[2]

Big O - O(n)
Big Theta - Θ(n)
Big Omega - Ω(n)

It's valuable for a programmer to learn how to compare performances of different algorithms and choose the best time-space complexity to solve a particular problem in the most efficient way possible.[2]

Big O specifically defines the worst-case scenario of an algorithm, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm. here O stands for order of growth.

Big Theta(Θ) is used to represent the average case scenario of an algorithm and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

Big Omega (Ω)is used to represent the best case scenario of an algorithm and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

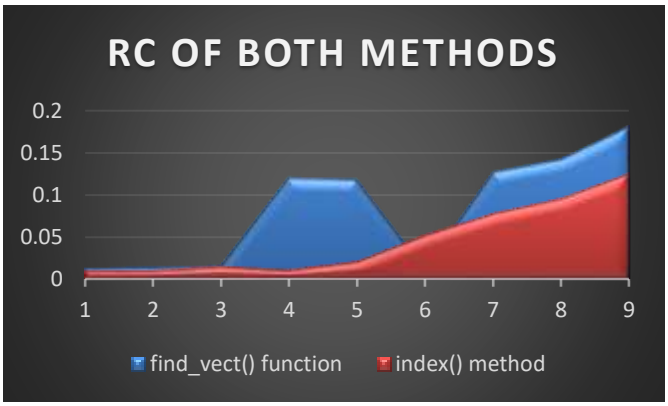
These three methods are the most common and very popular methods of design and analysis of an algorithm which are used for finding the efficiency of the program.

10. RUNTIME COMPLEXITY OF FINDING INDEX OF NUMBER IN A LIST

Input (No of Digits)	*find_vect() function*	*index() method*
100	0.012575492	0.010575492
1000	0.013789515	0.010695421
10000	0.015637874	0.015637874
50000	0.119718418	0.010993957
100000	0.118007764	0.020989698
200000	0.011995544	0.052467088
300000	0.126982545	0.078321424
400000	0.141989564	0.095634537
500000	0.180964345	0.125586376

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 9| Issue: 12| December 2023|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2023: 8.224 || ISI Value: 1.188

GRAPHICAL REPRESENTATION OF RUNTIME COMPLEXITY OF BOTH THE METHODS



11. CONCLUSION

In conclusion, if you are looking for the index of a given element in a list, using the `index()` method is a straightforward and efficient way to achieve it. However, if you are working specifically with vectors (implemented as arrays) and do not have access to a list or its methods, implementing a custom `find_vect()` function would be necessary. Keep in mind that `find_vect()` would have similar performance characteristics to the `index()` method, but it would only work with vectors and require separate implementation.

12.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 luminary **The Principal Capt. (IN) K Manikandan, Sainik School Amaravathinagar** who has been continuously motivating and extending their helping hand to us.

I express my sincere thanks to the academician The **Vice Principal Wg Cdr Deepti Upadhyay, Sainik School Amaravathinagar**, for constant encouragement and the guidance provided during this research.

I express my earnest thanks to the academician The **Administrative Officer Lt Col Deepu K, Sainik School Amaravathinagar**, for constant encouragement and the guidance provided during this research.

My sincere thanks to **Mr. Praveen Kumar MurigeppaJigajinni**, 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.

12. REFERENCES

1. *https://www.freecodecamp.org/news/time-complexity-of-algorithms/*
2. *https://www.educative.io/edpresso/time-complexity-vs-space-complexity*