```
Function convertToVector(data) -> vector
    Initialize vector as an empty collection
    Try
        Convert data to a vector and store it in vector
    Except
        If an exception of type TypeError occurs:
            Print "Data Type Mismatch! Please supply a list of integers as a parameter."
            Exit the function
        Else:
            Raise the exception
    End Try

    Return vector

Function linearSearch(array, searchValue: int) -> vector
    Set array to the result of convertToVector(array)
    Initialize count to 0
    Initialize an empty list found_indexes
    For each element in array with index i
        If element is equal to searchValue
            Append count to found_indexes
            If the length of found_indexes is equal to 2
                Return a vector with found_indexes
            End If
        End If
        Increment count by 1
    End For

 Return a vector containing -1  # Sentinel value if not found twice

Function main()
   Initialize a vector_variable
   Initialize an integer search_value
    LinearSearch(vector_variable, search_value)
```

Program Snippet

```
def linear_search(array, search_value):
    array = convert_to_vector(array)  # c1, 1 does not depend upon the size of data
    count = 0  # c2 , 1
    found_indexes = []  # c3 ,1

    for i in range(len(array)):  #c4, n times(each loop of i in the array generated by range) + 1 (range function has constant time complexity)
        if array[i] == search_value:  # c5 n times
            found_indexes.append(count)  # c6, n times
            if len(found_indexes) == 2:  # c7, n times + 1 (len function has constant time complexity)
                return np.array([found_indexes])  # c8, 1
        count += 1  # c9, n times

    return np.array([-1])  # c10, 1
```

Cost = c1 + c2 + c3 + c4 * (n + 1) + c5 * n + c6 * n + c7 * (n + 1) + c8 + c9 * n + c10 + c11

= (c1 + c2 + c3 + c8 + c10 + c11) + (c4 + c7) * (n + 1) + (c5 + c6 + c9) * n

= 1 + 1 + 1 + 1 + 1 + 1 + 2*(n+1) +3n

= 6 + 2n + 2 + 3n

= 8 + 5n

T(n) <= c*f(n) when n>n0

in this case,

T(n) = 8 + 5n

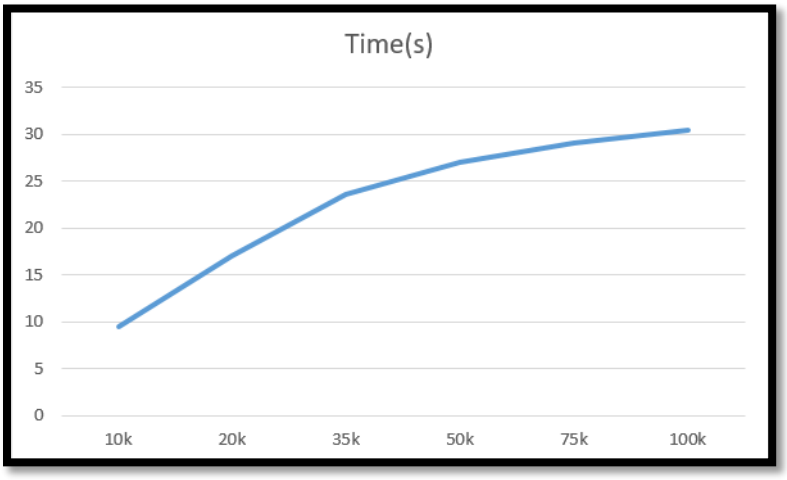
c >= 5

f(n) = n

when n>n0| n>1 | Since, n0=1

The time complexity for the algorithm is O(n)

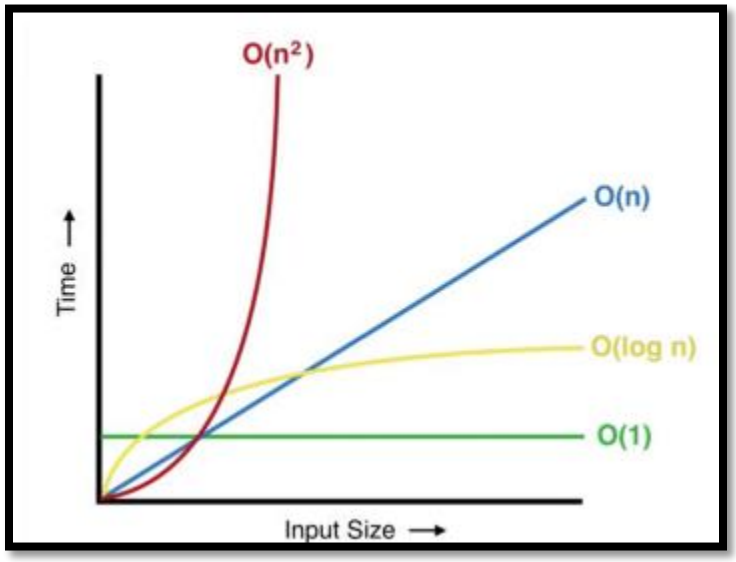
Comparison Report



Input Size	Hits	Misses	Min Steps	Average	Time(s)
10k	147	853	607	9439	9.439
20k	375	625	963	17043	17.043
35k	695	305	471	23630	23.63
50k	846	154	763	27074	27.074
75k	959	41	519	29101	29.101
100k	986	14	621	30419	30.419

The figure above was constructed in Microsoft Excel

Comparing our figure with the figure below..



The graph that was obtained by our program resembles the time complexity that is between O(n) and O(log n). Since the calculated time complexity of this program is O(n) and considering the fact that we are dealing with thousands of random values it is not as accurate as it could have been. But we can clearly see that the program is not o(n^2) or O(1) from our graph.