Signal Processing System Design Laboratory (EE69205)

LAB REPORT

by

YASH ANAND

(Roll No.- **23EE65R22**)

SIGNAL PROCESSING AND MACHINE LEARNING

Department of ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
Kharagpur - 721302, India
August, 2023

|| Experiment 1 ||
|| Search and Sorting in Arrays ||

1.0 Search in Arrays

AIM:
Estimation of polynomial order of complexity of searching (linear and binary) implemented in Python and their profiling for execution complexity estimation.

FUNCTION USED:

- **time.perf_counter()**: This function returns the current time in seconds, as measured by the performance counter.
- **append(size)**:This function will place new items in the available space so that we can add any object to a given list.

OBSERVATION:

★ The average search time for binary search increases linearly with the array size, while the average search time for linear search increases quadratically with the array size. This is because binary search divides the array in half at each step, so the number of steps required to search the array is proportional to the logarithm of the array size. Linear search, on the other hand, compares each element of the array to the target element, so the number of steps required to search the array is proportional to the square of the array size.

★ The difference in performance between binary search and linear search becomes more pronounced as the array size increases. This is because the logarithmic growth of binary search is much slower than the quadratic growth of linear search.

★ The performance of both binary search and linear search is affected by the randomness of the array elements. If the array elements are sorted, then binary search will be much faster than linear search. However, if the array elements are randomly distributed, then the difference in performance between the two algorithms will be smaller.

Here is a table that summarizes the time complexity of binary search and linear search:

Algorithm	Best Case	Worst Case
Linear Search	$O(1)$	$O(n)$
Binary Search	$O(1)$	$O(\log n)$

Table 1: Time Complexity of Linear Search and Binary Search

OUTPUT:

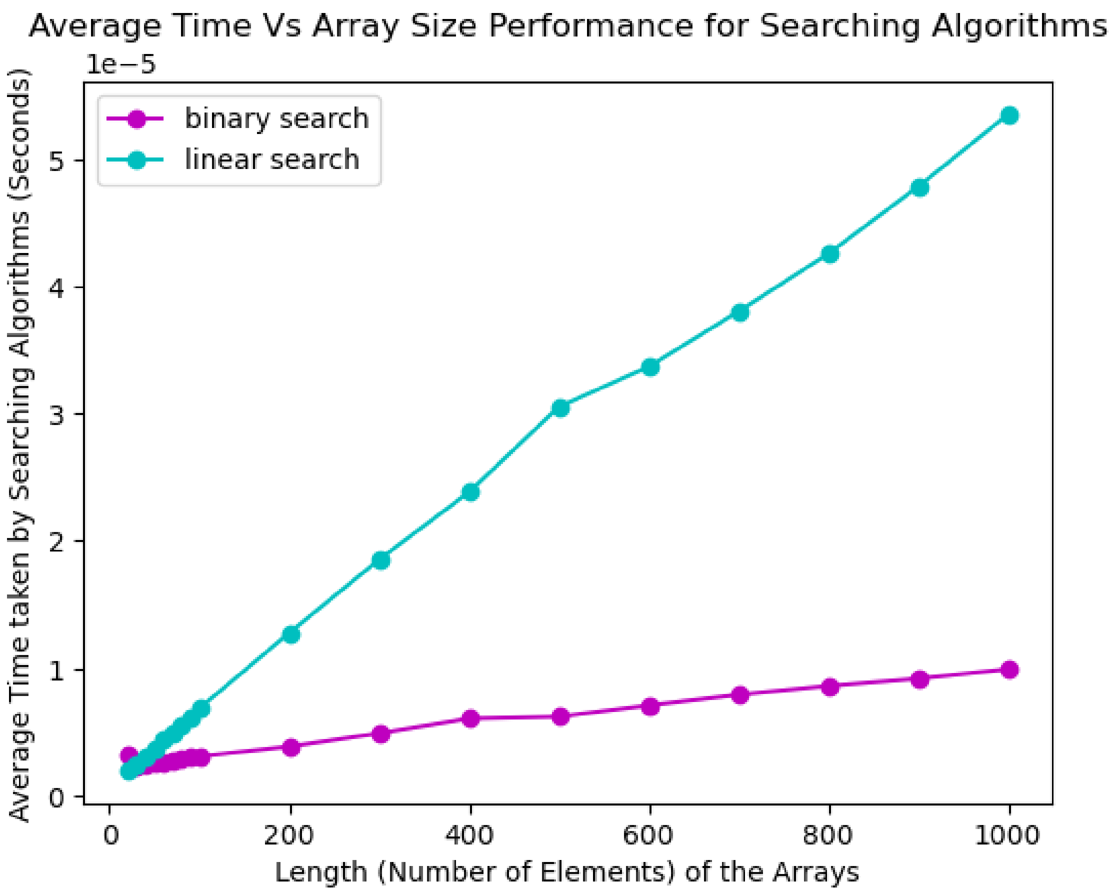

Figure 1: **Binary and Linear search time complexity comparison.**

2.0 Sorting in Arrays

AIM:
Estimation of polynomial order of complexity of Sorting (linear and bubble) implemented in Python and their profiling for execution complexity estimation.

FUNCTION USED:

- **np.random.randint()**: This function generates a random integer in a specified range.
- **plt.legend()**:This function I have used to display a legend on a plot. A legend is a table that explains the different symbols or colors used in a plot. It is used to make the plot more readable and understandable.

OBSERVATION:

★ Bubble Sort's average time complexity grows significantly as the length of the array increases. Bubble Sort has a worst-case time complexity of O(n$\hat{2}$), which makes it inefficient for larger datasets. This is evident from the steep increase in the graph's curve for Bubble Sort.

★ Insertion Sort's average time complexity also increases as the array length grows, but its growth is not as steep as Bubble Sort's. Insertion Sort has a worst-case time complexity of O(n$\hat{2}$) as well. However, its performance can be better for smaller datasets due to its adaptive nature.

★ For both sorting algorithms, the average time complexity increases as the array length increases. This is expected since the number of comparisons and swaps increase as the number of elements to be sorted increases

Here is a table that summarizes the time complexity of bubble sort and linear sort:

Algorithm	Best Case	Worst Case
Insertion Sort	$O(n)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$

Table 2: Time Complexity of Insertion Sort and Bubble Sort

OUTPUT:

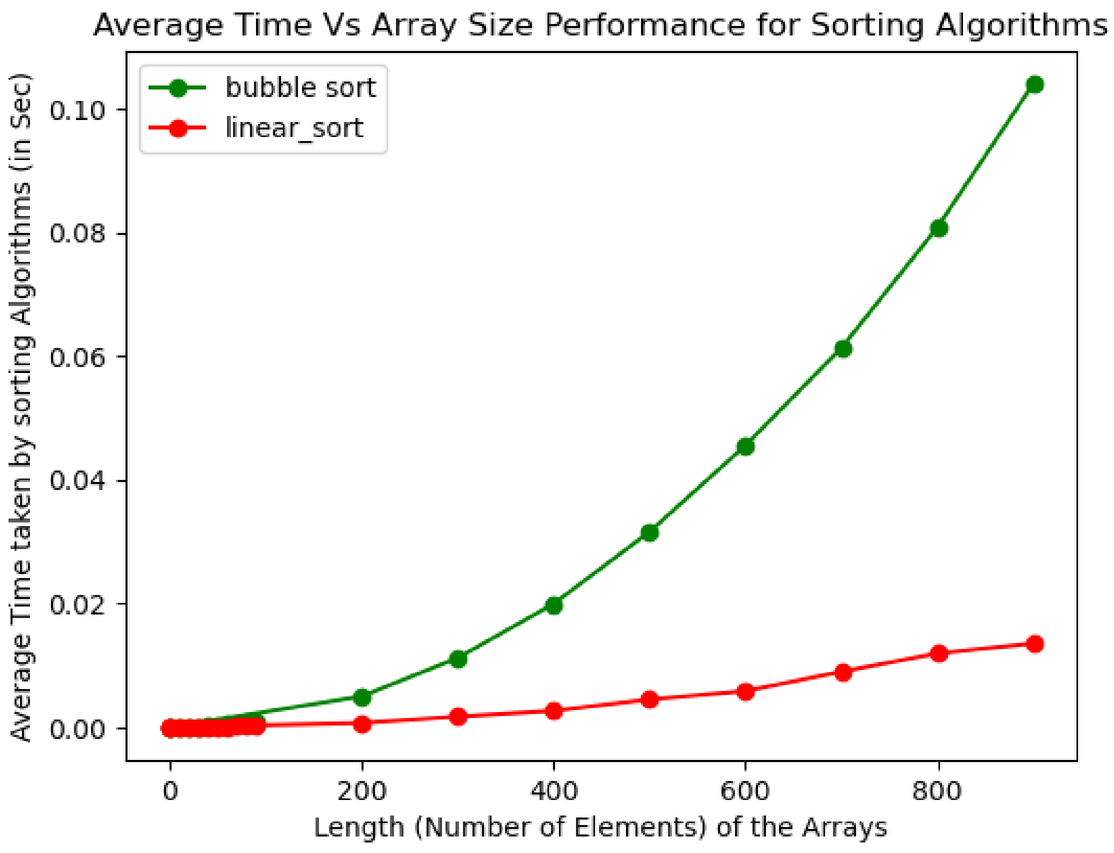

FIGURE 2: **Bubble and Linear sort time complexity comparison.**