Assignment 1

Manan Patel

September 8, 2024

A. Analyzing problem

1: Accurately define the problem by specifying the valid inputs and outputs, as well as their relationship.

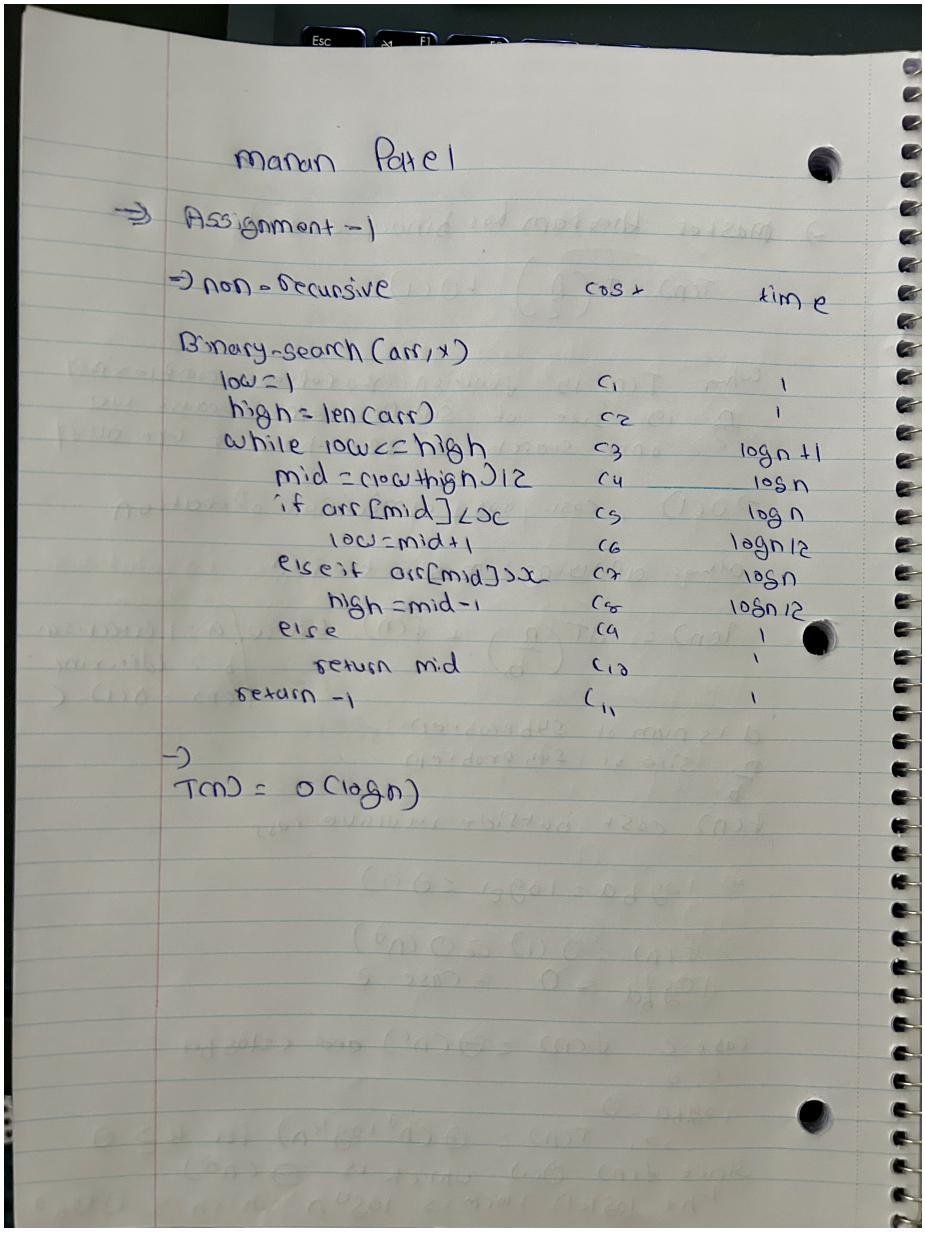
Problem: Find the location of x in a sorted array using the binary search algorithm. **Input:**

- **array:** A sorted array of integers or floats, ordered in non-decreasing (increasing) order and containing no duplicates.
- **x:** The target integer or float to find within the array.
- **low:** The starting index of the array (used in recursive implementations).
- **high:** The ending index of the array (used in recursive implementations).

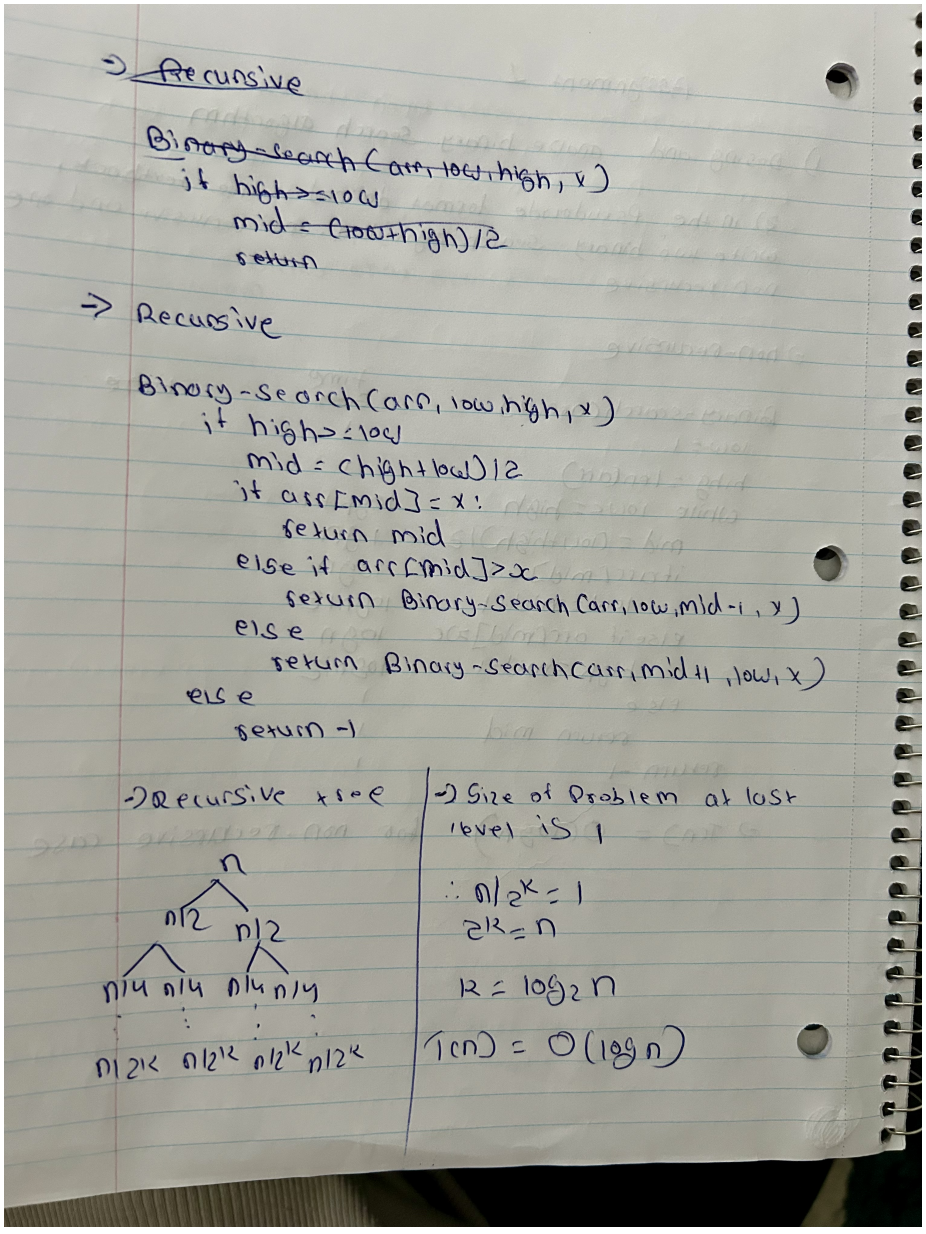
Output:

- **Index of x:** If x is found in the array, return the index where x is located.
- **-1 or 'not found':** If x is not present in the array, return -1 or the message 'not found'.

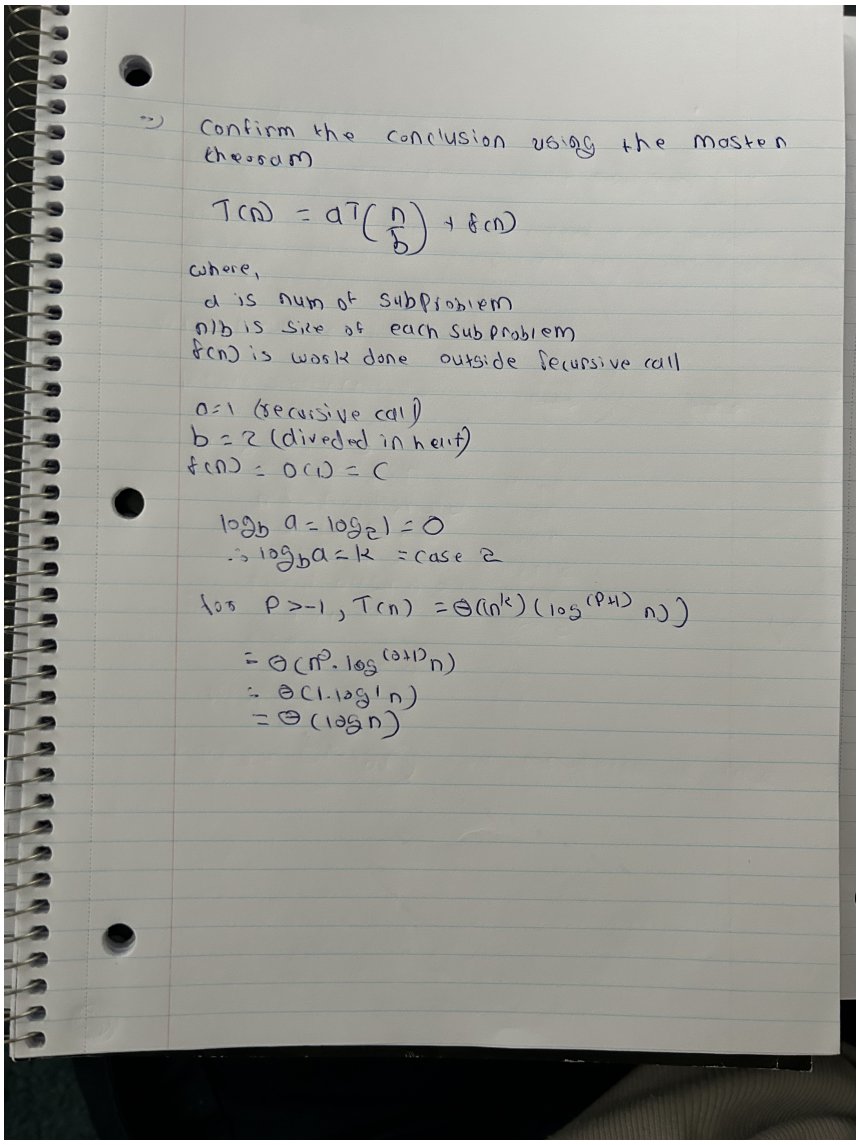
2-3: In the pseudo code format defined in the textbook, write two binary search algorithms, one recursive and the other non-recursive Decide the time function T(n) of the non-recursive algorithm like how Insertion Sort is analyzed.



2-4 Guess the time function T(n) of the recursive algorithm using a recursion tree, then confirm the conclusion using the master method



master method



B. Programming problem

3-Collect and display the testing results to show the difference of the two algorithms on time complexity.

Testing results

Array Size: 10

Test 1:
Insertion Sort Time (ns): 11500.00
Insertion Sort Calls: 13
Merge Sort Time (ns): 26700.00
Merge Sort Merge Calls: 9

Test 2:
Insertion Sort Time (ns): 5800.00
Insertion Sort Calls: 32
Merge Sort Time (ns): 10599.99
Merge Sort Merge Calls: 9

Test 3:
Insertion Sort Time (ns): 4000.00
Insertion Sort Calls: 21
Merge Sort Time (ns): 9000.00
Merge Sort Merge Calls: 9

Test 4:
Insertion Sort Time (ns): 3100.01
Insertion Sort Calls: 15
Merge Sort Time (ns): 8900.01
Merge Sort Merge Calls: 9

Test 5:
Insertion Sort Time (ns): 4000.00
Insertion Sort Calls: 24
Merge Sort Time (ns): 8700.01
Merge Sort Merge Calls: 9

Array Size: 100

Test 1:
Insertion Sort Time (ns): 333900.00
Insertion Sort Calls: 2656

Merge Sort Time (ns): 124000.00
Merge Sort Merge Calls: 99

Test 2:
Insertion Sort Time (ns): 273300.00
Insertion Sort Calls: 2163
Merge Sort Time (ns): 123800.00
Merge Sort Merge Calls: 99

Test 3:
Insertion Sort Time (ns): 289700.01
Insertion Sort Calls: 2467
Merge Sort Time (ns): 122899.99
Merge Sort Merge Calls: 99

Test 4:
Insertion Sort Time (ns): 324500.00
Insertion Sort Calls: 2811
Merge Sort Time (ns): 181500.00
Merge Sort Merge Calls: 99

Test 5:
Insertion Sort Time (ns): 309200.00
Insertion Sort Calls: 2292
Merge Sort Time (ns): 145000.00
Merge Sort Merge Calls: 99

Array Size: 1000

Test 1:
Insertion Sort Time (ns): 29492700.01
Insertion Sort Calls: 249619
Merge Sort Time (ns): 1330900.01
Merge Sort Merge Calls: 999

Test 2:
Insertion Sort Time (ns): 25699600.00
Insertion Sort Calls: 244418
Merge Sort Time (ns): 1730299.99
Merge Sort Merge Calls: 999

Test 3:
Insertion Sort Time (ns): 25797600.00
Insertion Sort Calls: 249882

Merge Sort Time (ns): 1374499.99
Merge Sort Merge Calls: 999

Test 4:
Insertion Sort Time (ns): 23762700.00
Insertion Sort Calls: 236486
Merge Sort Time (ns): 1387000.00
Merge Sort Merge Calls: 999

Test 5:
Insertion Sort Time (ns): 27501399.99
Insertion Sort Calls: 252732
Merge Sort Time (ns): 1354099.99
Merge Sort Merge Calls: 999

Analysis

I conducted 5 tests for each of the following array sizes: 10, 100, and 1000.

For an array size of 10, insertion sort performs faster than merge sort.
For an array size of 100, merge sort outperforms insertion sort.
For an array size of 1000, merge sort is significantly faster than insertion sort.

From these results, it is evident that insertion sort only outperforms merge sort for very small array sizes (10). As the array size increases, merge sort consistently performs better, with a larger margin as the size grows.

Comparing with Theoretical Analysis

Theoretical analysis indicates that merge sort has a worst-case time complexity of $\theta(n \log n)$, whereas insertion sort has a worst-case time complexity of $\theta(n^2)$. In practice, insertion sort may be faster for small arrays due to lower constant factors.

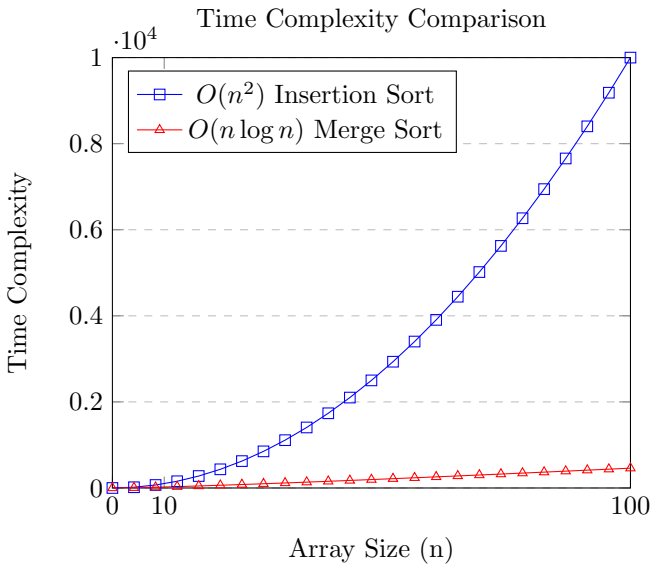


Figure 1: Time Complexity Comparison of Insertion Sort and Merge Sort

Our experimental data aligns with this theoretical analysis, as illustrated in Figure 1. The graph visually demonstrates that insertion sort has quadratic time complexity, growing rapidly with increasing input sizes, whereas merge sort grows much slower in comparison, due to its logarithmic factor.

In the case of small arrays (around $n = 10$), insertion sort is faster due to the lower constant factors associated with it. However, as the array size grows larger (closer to $n = 100$ and beyond), merge sort's $\theta(n \log n)$ complexity leads to much faster performance compared to insertion sort's $\theta(n^2)$.

In conclusion, the performance of sorting algorithms depends on the array size. For very small arrays, insertion sort can be preferable. However, for larger arrays, merge sort generally provides better performance.