

## AI ASSISTED CODING

### LAB ASSIGNMENT: 12.4

NAME: SK.NAHEEMA

H.NO : 2403A52059

BATCH NO : 03

#### TASK 1:

Implementing Bubble Sort with AI Comments

- Task: Write a Python implementation of Bubble Sort.
- Instructions:
  - Students implement Bubble Sort normally.
  - Ask AI to generate inline comments explaining key logic (like swapping, passes, and termination).
  - Request AI to provide time complexity analysis.

#### PROMPT:

Generate python code and implementing Bubble Sort with AI Comments

- Task: Write a Python implementation of Bubble Sort.
- Instructions:
  - Students implement Bubble Sort normally.
  - Ask AI to generate inline comments explaining key

logic (like swapping, passes, and termination)

o Request AI to provide time complexity analysis.

## CODE & OUTPUT:

The screenshot shows the VS Code interface with the following details:

- File Explorer:** On the left, showing icons for RUN, RUN DEBUG, and other project files.
- Code Editor:** The main area displays Python code for implementing bubble sort. The code is as follows:

```
def bubble_sort(arr):
    """
    Sorts the list "arr" in-place using Bubble Sort algorithm.
    """
    n = len(arr)
    # Outer loop for each pass over the array
    for i in range(n):
        # We assume no swaps this pass (to detect if array already sorted)
        swapped = False
        # For each pair of adjacent elements in positions [0 .. n-2]
        # (since last i elements already bubbled to correct positions)
        for j in range(0, n - i - 1):
            # If the element at j is greater than the one at j+1, swap them
            if arr[j] > arr[j + 1]:
                # Swap the two elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True # mark that a swap happened in this pass
        # If no two elements were swapped in the inner loop, the list is already sorted
        if not swapped:
            break # Early termination - no further passes needed
```

- Terminal:** Below the editor, the terminal window shows the execution of the script:

```
PS C:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl> 8 <C:\Users\kadal\AppData\Local\Microsoft\WindowsApps\python3.11.exe> 'C:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl\12.4 AI LAB T1.py'
Before: [5, 1, 4, 2, 8]
After: [1, 2, 4, 5, 8]
PS C:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl>
```

- Bottom Status Bar:** Shows connection status (No connection), system temperature (25°C Heavy rain), and system information (ENG IN, 11:31 25-09-2025).

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure including 'RUN AND DEBUG' and 'RUN'.
- Search Bar:** Contains the word 'Search'.
- Code Editor:** Displays a Python script named 'bubble\_sort.py'. The code implements a bubble sort algorithm and includes example usage at the bottom.
- Terminal:** Shows the output of running the script. It prints 'Before' and 'After' states of a list [5, 1, 4, 2, 8] and ends with the command 'PS C:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532\_0\Lib\site-packages\alembic\ddl>'.
- Python Debug Console:** Shows the command 'python -m alembic -v <file>' being run.
- Bottom Status Bar:** Includes icons for file status (30), connection (No connection), and system information (25°C Heavy rain).

## EXPLANATION:

- **Outer loop** (`for i in range(n)`): Each pass moves the next largest (or smallest, depending on sort direction) element into its correct position at the end. After `i` passes, the last `i` elements are already sorted and in place.
  - **Inner loop** (`for j in range(0, n - i - 1)`): Compares adjacent pairs `arr[j]` and `arr[j + 1]`, and swaps if they are out of order. We don't go all the way to the end every pass because the tail end is already sorted.
  - **Swapping logic:** `arr[j], arr[j + 1] = arr[j + 1], arr[j]` performs the exchange in one line.
  - **Early termination check** (`if not swapped: break`): If in a full pass we made no swaps, the list is already sorted — so we break to avoid unnecessary passes (optimization).
- 

### Time Complexity Analysis

- **Worst-case time complexity:**  
In the worst case (the input is in reverse order), the algorithm will perform all possible comparisons and swaps in every pass. That leads to
$$\sum_{i=0}^{n-1} (n - i - 1) = \frac{n(n - 1)}{2} = O(n^2)$$
So  $O(n^2)$ .
- **Best-case time complexity:**  
If the array is already sorted, then in the first pass no swaps will be made, so `swapped` remains `False` and we break out early. The inner loop runs once with  $\sim n$  comparisons, so it is  $O(n)$  in the best case (with the early termination optimization).
- **Best-case time complexity:**  
If the array is already sorted, then in the first pass no swaps will be made, so `swapped` remains `False` and we break out early. The inner loop runs once with  $\sim n$  comparisons, so it is  $O(n)$  in the best case (with the early termination optimization).
- **Average-case time complexity:**  
Generally also  $O(n^2)$ .
- **Space complexity:**  
Bubble Sort is in-place (only a constant amount of extra memory for indices and flag), so  $O(1)$  additional space.

## TASK 2:

### Optimizing Bubble Sort → Insertion Sort

- Task: Provide Bubble Sort code to AI and ask it to suggest a more efficient algorithm for partially sorted arrays.

- Instructions:
  - Students implement Bubble Sort first.
  - Ask AI to suggest an alternative (Insertion Sort).
  - Compare performance on nearly sorted input.

#### PROMPT:

Generate python code and optimizing Bubble Sort →  
Insertion Sort

- Task: Provide Bubble Sort code to AI and ask it to suggest a more efficient algorithm for partially sorted arrays.

- Instructions:

- Students implement Bubble Sort first.
- Ask AI to suggest an alternative (Insertion Sort).
- Compare performance on nearly sorted input.

#### CODE & OUTPUT:

The screenshot shows a Jupyter Notebook environment with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Search.
- Run and Debug:** LAB 11.1.py, LAB 11.1.T8.py, WEB LAB 4 HTML.html, WEB LAB 5 HTML.html, web 2.py+, 12.4 AI LAB T1.py, AI T 12.4 LAB.py X, web lab.py.
- Run Section:** RUN, Run and Debug.
- Text Area:** Code for bubble sort and insertion sort.

```
def insertion_sort(arr):  
    for i in range(1, len(arr)): # loop through array  
        key = arr[i]  
        j = i - 1  
        # Shift elements that are greater than key to one position ahead  
        while j >= 0 and arr[j] > key:  
            arr[j + 1] = arr[j]  
            j -= 1  
        # Place key in its correct spot  
        arr[j + 1] = key  
    return arr  
  
def make_almost_sorted(n, num_swaps=5):  
    """  
    Return a list of size n that is nearly sorted,  
    by starting with [0,1,2,...,n-1] and performing a few random adjacent swaps.  
    """  
    lst = list(range(n))  
    for _ in range(num_swaps):  
        i = random.randint(0, n - 2)  
        # swap adjacent pair  
        lst[i], lst[i + 1] = lst[i + 1], lst[i]  
    return lst
```

- Bottom Navigation:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, QUERY RESULTS.
- Python Debug Console:** Shows execution results for bubble sort and insertion sort on arrays of size 1000, 5000, 10000, and 100000.
- Bottom Status Bar:** Spaces: 4, UTF-8, Python, 3.11.9 (Microsoft Store), 11:08:02, ENG IN, 25-09-2023.

The screenshot shows a Windows desktop with a Python development environment set up in Visual Studio Code (VS Code). The code editor has a dark theme and displays a script named `LAB 11.1.py`. The script contains functions for benchmarking sorting algorithms and printing test parameters. The terminal below the editor shows the output of running the script, comparing bubble sort and insertion sort on arrays of size 1000, 5000, and 10000. The status bar at the bottom right indicates "Airplane mode off".

```
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

C:\Users\kadal>anaconda3> pkgs> alembic 1.16.4-py313haa95532_0> lib> site-packages> alembic> ddl> AI T 2 12.4 LAB.py > ...

52
53
54 def benchmark(sort_fn, arr):
55     """
56     Time how long sort_fn takes to sort a *copy* of arr.
57     Returns the duration in seconds.
58     """
59
60     arr_copy = arr.copy()
61     start = time.perf_counter()
62     sort_fn(arr_copy)
63     end = time.perf_counter()
64     return end - start
65
66 def main():
67     # testing parameters
68     test_sizes = [1000, 5000, 10000]
69     swap_counts = [1, 5, 20] # how many adjacent swaps to create disorder
70
71     print("Comparing Bubble Sort vs Insertion Sort on nearly sorted arrays")
72     for n in test_sizes:
73         for swaps in swap_counts:
74             print(f"n = {n}, swaps = {swaps} : bubble: {bubble_time}, insertion: {insertion_time}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Python Debug Console + □ ... x

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off

File Edit Selection View Go Run Terminal Help ↺ → Search
RUN AND DEBUG ... LAB 11.1.py AI LAB 11.1 T8.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py+ 12.4 AI LAB T1.py AIT 2 12.4 LAB.py web lab.py

RUN
Run and Debug

To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Comparing bubble Sort vs Insertion Sort on nearly sorted arrays

n = 1000, swaps = 1 + bubble: 0.003613s, insertion: 0.001743s
n = 1000, swaps = 20 + bubble: 0.004632s, insertion: 0.005344s
n = 10000, swaps = 1 + bubble: 0.019424s, insertion: 0.012826s
n = 10000, swaps = 5 + bubble: 0.008574s, insertion: 0.016755s
n = 10000, swaps = 20 + bubble: 0.010897s, insertion: 0.011559s

Sample (nearly sorted) before: [0, 2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 12, 13, 15, 16, 17, 18, 19]
Sorted by bubble : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Sorted by insertion: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
PS C:\Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\binc\ddls

Airplane mode off
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a folder structure with files like LAB 11.1.py, AI LAB 11.1 T8.py, WEB LAB 4 HTML.html, WEB LAB 5 HTML.html, web 2.py, 12.4 AI LAB T1.py, AI T 2 12.4 LAB.py, and web lab.py.
- Run View:** The "RUN" tab is selected, showing the "Run and Debug" configuration. It includes a note: "To customize Run and Debug, open a folder and create a launch.json file." Below it, a note says: "Debug using a terminal command or in an interactive chat."
- Code Editor:** The main editor area contains Python code for sorting algorithms. The code defines a `main` function that generates nearly sorted arrays, benchmarks bubble sort and insertion sort, and prints the results. It also demonstrates how to check if an array is nearly sorted by comparing it with its sorted version.
- Terminal:** The terminal window at the bottom shows the output of running the script, comparing the performance of bubble sort and insertion sort on nearly sorted arrays of size 1000, 5000, 10000, and 100000. The output includes swap counts and execution times.
- Bottom Status Bar:** Shows the current workspace has 30 files, no connection, and the weather is 25°C Heavy rain. It also displays the number of spaces used (Spaces: 4), the current tab (UTF-8), the Python extension icon, the Microsoft Store icon, and the date and time (25-09-2023).

## **EXPLANATION:**

- **Bubble Sort** is implemented with the usual double loop and an early-exit `swapped` flag to stop if no swaps occur in a pass.
  - **Insertion Sort** is used as the alternative, especially effective when the array is already "almost sorted."
  - `make_almost_sorted(n, num_swaps)` creates test data that is nearly sorted by applying a few adjacent swaps to a sorted list of size `n`.
  - In `main()`, we run benchmarks for various sizes and disorder levels, and print timings that show where insertion sort overtakes bubble sort.

You can run this script and observe that:

- For large  $n$  and **very few swaps**, insertion sort tends to be *much faster* than bubble sort.
  - For more disorder (lots of swaps), both may degrade toward  $O(n^2)$  behavior, but insertion sort's constant factors are often better in practice for partially sorted data.

If you like, I can also generate a **detailed log / step-by-step** version (printing each swap or pass) for teaching purposes, or visualize the time complexity difference (e.g. a plot). Do you want me to produce that?

### TASK 3:

# Binary Search vs Linear Search

- Task: Implement both Linear Search and Binary Search.

- Instructions:
  - o Use AI to generate docstrings and performance notes.
  - o Test both algorithms on sorted and unsorted data.
  - o Ask AI to explain when Binary Search is preferable.

#### PROMPT:

Generate python code and binary Search vs Linear Search

- Task: Implement both Linear Search and Binary Search.
- Instructions:
  - o Use AI to generate docstrings and performance notes.
  - o Test both algorithms on sorted and unsorted data.
  - o Ask AI to explain when Binary Search is preferable.

#### CODE & OUTPUT:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a folder structure for "RUN DEBUG" and "RUN".
- Search Bar:** Contains the word "Search".
- Code Editor:** Displays Python code for a linear search algorithm. The code includes imports for random, time, and typing modules, a docstring explaining the function, arguments, and return type, and a performance analysis section.
- Python Debug Console:** Shows the output of running the code. It includes:
  - Execution command: PS C:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532\_0\lib\site-packages\alembic\ddl.py & c:\Users\kadal\AppData\Local\Microsoft\WindowsApps\python3.11.exe "c:\Users\kadal\vscodeextensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher" "55118" ... c:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532\_0\lib\site-packages\alembic\ddl\T1 12.4 LAB.py"
  - Testing on Unsorted Data: Linear search time (unsorted): 0.000293 s, Binary search time (unsorted, invalid) : 0.000034 s (result may be wrong)
  - Testing on Sorted Data: Linear search time (sorted list) : 0.025805 s, Binary search time (sorted list) : 0.000036 s
  - Correctness check: target = 49895, Linear search found at index: 50000
- Bottom Status Bar:** Shows connection status (No connection), file count (311), encoding (UTF-8), language (Python), and version (3.11.9 Microsoft Store). It also displays system icons for battery, signal, and network.

The screenshot shows the Microsoft Visual Studio Code interface with the Python extension installed. The left sidebar displays the 'RUN' and 'Run and Debug' sections, along with a note about launching a JSON file. The main editor area contains a Python script for a binary search algorithm, with detailed docstrings explaining the code's behavior. The terminal at the bottom shows the execution of the script, including imports, variable definitions, and the output of the search function. The status bar at the bottom right indicates the current workspace has 4 spaces, 4 tabs open, and is using Python 3.11.9.

```
File Edit Selection View Go Run Terminal Help ← → Search RUN AND DEBUG ... AI LAB 11.1 LAB.py WEB LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py 9+ 12.4 AI LAB T1.py AIT 12.4 LAB.py x AI T3 12.4 LAB.py x we ▾ ... RUN Run and Debug To customize Run and Debug, open a folder and create a launch.json file. Debug using a terminal command or in an interactive chat. 27 28 def binary_search(arr: List[Any], target: Any) -> Optional[int]: 29     """ 30         Binary search (assumes arr is sorted in ascending order). 31     Args: 32         arr: A sorted list of comparable items. 33         target: The value to search for. 34     Returns: 35         The index of target in arr if found, else None. 36     """ 37     # Performance: 38     # - Worst-case time complexity: O(log n) 39     # - Average-case: O(log n) 40     # - Best-case: O(1) (if target is at mid initially) 41     # - Space complexity: O(1) (iterative) 42     # Notes: 43     # - Works only on sorted lists. 44     # - Each iteration halves the search interval. 45     """ 46     :param arr: List of comparable items, sorted in ascending order. 47     :param target: Value to search for. 48     :return: Index of target if found, else None. 49     low = 0 50 51     while low <= high: 52         mid = (low + high) // 2 53         if arr[mid] == target: 54             return mid 55         elif arr[mid] < target: 56             low = mid + 1 57         else: 58             high = mid - 1 59     return None 60 61 if __name__ == "__main__": 62     arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 63     target = 5 64     print(binary_search(arr, target)) 65 66 # Output: 4 67 68 PS C:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\lib\site-packages\alembic\ddl\ & 'c:\users\kadal\appdata\local\microsoft\windowsApps\python3.11.6\python.exe' 'c:\users\kadal\anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\lib\site-packages\alembic\ddl\ait\ait 12.4 LAB.py' 69 == Testing on Unsorted Data == 70 Linear search time (unsorted): 0.000293 s 71 Binary search time (unsorted, invalid): 0.000034 s (result may be wrong) 72 73 == Testing on Sorted Data == 74 Linear search time (sorted list) : 0.025805 s 75 Binary search time (sorted list) : 0.000036 s 76 77 Correctness check: target = 49805 78 Linear search found at index: 50000 79 Spaces: 4 UTF-8 () Python 3.11.9 (Microsoft Store) 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 319 320 321 322 323 324 325 326 327 328 329 329 330 331 332 333 334 335 336 337 338 339 339 340 341 342 343 344 345 346 347 348 349 349 350 351 352 353 354 355 356 357 358 359 359 360 361 362 363 364 365 366 367 368 369 369 370 371 372 373 374 375 376 377 378 379 379 380 381 382 383 384 385 386 387 388 389 389 390 391 392 393 394 395 396 397 398 399 399 400 401 402 403 404 405 406 407 408 409 409 410 411 412 413 414 415 416 416 417 418 419 419 420 421 422 423 424 425 426 426 427 428 429 429 430 431 432 433 434 435 436 436 437 438 439 439 440 441 442 443 444 445 445 446 447 448 449 449 450 451 452 453 454 455 455 456 457 458 458 459 460 461 462 462 463 464 465 465 466 467 468 468 469 470 471 472 472 473 474 475 475 476 477 478 478 479 479 480 481 482 482 483 484 485 485 486 487 488 488 489 489 490 491 492 492 493 494 495 495 496 497 498 498 499 499 500 501 502 502 503 504 505 505 506 507 508 508 509 510 511 511 512 513 513 514 515 515 516 517 517 518 518 519 519 520 521 521 522 523 523 524 525 525 526 526 527 527 528 528 529 529 530 531 531 532 532 533 533 534 534 535 535 536 536 537 537 538 538 539 539 540 540 541 541 542 542 543 543 544 544 545 545 546 546 547 547 548 548 549 549 550 550 551 551 552 552 553 553 554 554 555 555 556 556 557 557 558 558 559 559 560 560 561 561 562 562 563 563 564 564 565 565 566 566 567 567 568 568 569 569 570 570 571 571 572 572 573 573 574 574 575 575 576 576 577 577 578 578 579 579 580 580 581 581 582 582 583 583 584 584 585 585 586 586 587 587 588 588 589 589 590 590 591 591 592 592 593 593 594 594 595 595 596 596 597 597 598 598 599 599 600 600 601 601 602 602 603 603 604 604 605 605 606 606 607 607 608 608 609 609 610 610 611 611 612 612 613 613 614 614 615 615 616 616 617 617 618 618 619 619 620 620 621 621 622 622 623 623 624 624 625 625 626 626 627 627 628 628 629 629 630 630 631 631 632 632 633 633 634 634 635 635 636 636 637 637 638 638 639 639 640 640 641 641 642 642 643 643 644 644 645 645 646 646 647 647 648 648 649 649 650 650 651 651 652 652 653 653 654 654 655 655 656 656 657 657 658 658 659 659 660 660 661 661 662 662 663 663 664 664 665 665 666 666 667 667 668 668 669 669 670 670 671 671 672 672 673 673 674 674 675 675 676 676 677 677 678 678 679 679 680 680 681 681 682 682 683 683 684 684 685 685 686 686 687 687 688 688 689 689 690 690 691 691 692 692 693 693 694 694 695 695 696 696 697 697 698 698 699 699 700 700 701 701 702 702 703 703 704 704 705 705 706 706 707 707 708 708 709 709 710 710 711 711 712 712 713 713 714 714 715 715 716 716 717 717 718 718 719 719 720 720 721 721 722 722 723 723 724 724 725 725 726 726 727 727 728 728 729 729 730 730 731 731 732 732 733 733 734 734 735 735 736 736 737 737 738 738 739 739 740 740 741 741 742 742 743 743 744 744 745 745 746 746 747 747 748 748 749 749 750 750 751 751 752 752 753 753 754 754 755 755 756 756 757 757 758 758 759 759 760 760 761 761 762 762 763 763 764 764 765 765 766 766 767 767 768 768 769 769 770 770 771 771 772 772 773 773 774 774 775 775 776 776 777 777 778 778 779 779 780 780 781 781 782 782 783 783 784 784 785 785 786 786 787 787 788 788 789 789 790 790 791 791 792 792 793 793 794 794 795 795 796 796 797 797 798 798 799 799 800 800 801 801 802 802 803 803 804 804 805 805 806 806 807 807 808 808 809 809 810 810 811 811 812 812 813 813 814 814 815 815 816 816 817 817 818 818 819 819 820 820 821 821 822 822 823 823 824 824 825 825 826 826 827 827 828 828 829 829 830 830 831 831 832 832 833 833 834 834 835 835 836 836 837 837 838 838 839 839 840 840 841 841 842 842 843 843 844 844 845 845 846 846 847 847 848 848 849 849 850 850 851 851 852 852 853 853 854 854 855 855 856 856 857 857 858 858 859 859 860 860 861 861 862 862 863 863 864 864 865 865 866 866 867 867 868 868 869 869 870 870 871 871 872 872 873 873 874 874 875 875 876 876 877 877 878 878 879 879 880 880 881 881 882 882 883 883 884 884 885 885 886 886 887 887 888 888 889 889 890 890 891 891 892 892 893 893 894 894 895 895 896 896 897 897 898 898 899 899 900 900 901 901 902 902 903 903 904 904 905 905 906 906 907 907 908 908 909 909 910 910 911 911 912 912 913 913 914 914 915 915 916 916 917 917 918 918 919 919 920 920 921 921 922 922 923 923 924 924 925 925 926 926 927 927 928 928 929 929 930 930 931 931 932 932 933 933 934 934 935 935 936 936 937 937 938 938 939 939 940 940 941 941 942 942 943 943 944 944 945 945 946 946 947 947 948 948 949 949 950 950 951 951 952 952 953 953 954 954 955 955 956 956 957 957 958 958 959 959 960 960 961 961 962 962 963 963 964 964 965 965 966 966 967 967 968 968 969 969 970 970 971 971 972 972 973 973 974 974 975 975 976 976 977 977 978 978 979 979 980 980 981 981 982 982 983 983 984 984 985 985 986 986 987 987 988 988 989 989 990 990 991 991 992 992 993 993 994 994 995 995 996 996 997 997 998 998 999 999 1000 1000 1001 1001 1002 1002 1003 1003 1004 1004 1005 1005 1006 1006 1007 1007 1008 1008 1009 1009 1010 1010 1011 1011 1012 1012 1013 1013 1014 1014 1015 1015 1016 1016 1017 1017 1018 1018 1019 1019 1020 1020 1021 1021 1022 1022 1023 1023 1024 1024 1025 1025 1026 1026 1027 1027 1028 1028 1029 1029 1030 1030 1031 1031 1032 1032 1033 1033 1034 1034 1035 1035 1036 1036 1037 1037 1038 1038 1039 1039 1040 1040 1041 1041 1042 1042 1043 1043 1044 1044 1045 1045 1046 1046 1047 1047 1048 1048 1049 1049 1050 1050 1051 1051 1052 1052 1053 1053 1054 1054 1055 1055 1056 1056 1057 1057 1058 1058 1059 1059 1060 1060 1061 1061 1062 1062 1063 1063 1064 1064 1065 1065 1066 1066 1067 1067 1068 1068 1069 1069 1070 1070 1071 1071 1072 1072 1073 1073 1074 1074 1075 1075 1076 1076 1077 1077 1078 1078 1079 1079 1080 1080 1081 1081 1082 1082 1083 1083 1084 1084 1085 1085 1086 1086 1087 1087 1088 1088 1089 1089 1090 1090 1091 1091 1092 1092 1093 1093 1094 1094 1095 1095 1096 1096 1097 1097 1098 1098 1099 1099 1100 1100 1101 1101 1102 1102 1103 1103 1104 1104 1105 1105 1106 1106 1107 1107 1108 1108 1109 1109 1110 1110 1111 1111 1112 1112 1113 1113 1114 1114 1115 1115 1116 1116 1117 1117 1118 1118 1119 1119 1120 1120 1121 1121 1122 1122 1123 1123 1124 1124 1125 1125 1126 1126 1127 1127 1128 1128 1129 1129 1130 1130 1131 1131 1132 1132 1133 1133 1134 1134 1135 1135 1136 1136 1137 1137 1138 1138 1139 1139 1140 1140 1141 1141 1142 1142 1143 1143 1144 1144 1145 1145 1146 1146 1147 1147 1148 1148 1149 1149 1150 1150 1151 1151 1152 1152 1153 1153 1154 1154 1155 1155 1156 1156 1157 1157 1158 1158 1159 1159 1160 1160 1161 1161 1162 1162 1163 1163 1164 1164 1165 1165 1166 1166 1167 1167 1168 1168 1169 1169 1170 1170 1171 1171 1172 1172 1173 1173 1174 1174 1175 1175 1176 1176 1177 1177 1178 1178 1179 1179 1180 1180 1181 1181 1182 1182 1183 1183 1184 1184 1185 1185 1186 1186 1187 1187 1188 1188 1189 1189 1190 1190 1191 1191 1192 1192 1193 1193 1194 1194 1195 1195 1196 1196 1197 1197 1198 1198 1199 1199 1200 1200 1201 1201 1202 1202 1203 1203 1204 1204 1205 1205 1206 1206 1207 1207 1208 1208 1209 1209 1210 1210 1211 1211 1212 1212 1213 1213 1214 1214 1215 1215 1216 1216 1217 1217 1218 1218 1219 1219 1220 1220 1221 1221 1222 1222 1223 1223 1224 1224 1225 1225 1226 1226 1227 1227 1228 1228 1229 1229 1230 1230 1231 1231 1232 1232 1233 1233 1234 1234 1235 1235 1236 1236 1237 1237 1238 1238 1239 1239 1240 1240 1241 1241 1242 1242 1243 1243 1244 1244 1245 1245 1246 1246 1247 1247 1248 1248 1249 1249 1250 1250 1251 1251 1252 1252 1253 1253 1254 1254 1255 1255 1256 1256 1257 1257 1258 1258 1259 1259 1260 1260 1261 1261 1262 1262 1263 1263 1264 1264 1265 1265 1266 1266 1267 1267 1268 1268 1269 1269 1270 1270 1271 1271 1272 1272 1273 1273 1274 1274 1275 1275 1276 1276 1277 1277 1278 1278 1279 1279 1280 1280 1281 1281 1282 1282 1283 1283 1284 1284 1285 1285 1286 1286 1287 1287 1288 1288 1289 1289 1290 1290 1291 1291 1292 1292 1293 1293 1294 1294 1295 1295 1296 1296 1297 1297 1298 1298 1299 1299 1300 1300 1301 1301 1302 1302 1303 1303 1304 1304 1305 1305 1306 1306 1307 1307 1308 1308 1309 1309 1310 1310 1311 1311 1312 1312 1313 1313 1314 1314 1315 1315 1316 1316 1317 1317 1318 1318 1319 1319 1320 1320 1321 1321 1322 1322 1323 1323 1324 1324 1325 1325 1326 1326 1327 1327 1328 1328 1329 1329 1330 1330 1331 1331 1332 1332 1333 1333 1334 1334 1335 1335 1336 1336 1337 1337 1338 1338 1339 1339 1340 1340 1341 1341 1342 1342 1343 1343 1344 1344 1345 1345 1346 1346 1347 1347 1348 1348 1349 1349 1350 1350 1351 1351 1352 1352 1353 1353 1354 1354 1355 1355 1356 1356 1357 1357 1358 1358 1359 1359 1360 1360 1361 1361 1362 1362 1363 1363 1364 1364 1365 1365 1366 1366 1367 1367 1368 1368 1369 1369 1370 1370 1371 1371 1372 1372 1373 1373 1374 1374 1375 1375 1376 1376 1377 1377 1378 1378 1379 1379 1380 1380 1381 1381 1382 1382 1383 1383 1384 1384 1385 1385 1386 1386 1387 1387 1388 1388 1389 1389 1390 1390 1391 1391 1392 1392 1393 1393 1394 1394 1395 1395 1396 1396 1397 1397 1398 1398 1399 1399 1400 1400 1401 1401 1402 1402 1403 1403 1404 1404 1405 1405 1406 1406 1407 1407 1408 1408 1409 1409 1410 1410 1411 1411 1412 1412 1413 1413 1414 1414 1415 1415 1416 1416 1417 1417 1418 1418 1419 1419 1420 1420 1421 1421 1422 1422 1423 1423 1424 1424 1425 1425 1426 1426 1427 1427 1428 1428 1429 1429 1430 1430 1431 1431 1432 1432 1433 1433 1434 1434 1435 1435 1436 1436 1437 1437 1438 1438 1439 1439 1440 1440 1441 1441 1442 1442 1443 1443 1444 1444 1445 1445 1446 1446 1447 1447 1448 1448 1449 1449 1450 1450 1451 1451 1452 1452 1453 1453 1454 1454 1455 1455 1456 1456 1457 1457 1458 1458 1459 1459 1460 1460 1461 1461 1462 1462 1463 1463 1464 1464 1465 1465 1466 1466 1467 1467 1468 1468 1469 1469 1470 1470 1471 1471 1472 1472 1473 1473 1474 1474 1475 1475 1476 1476 1477 1477 1478 1478 1479 1479 1480 1480 1481 1481 1482 1482 1483 1483 1484 1484 1485 1485 1486 1486 1487 1487 1488 1488 1489 1489 1490 1490 1491 1491 1492 1492 1493 1493 1494 1494 1495 1495 1496 1496 1497 1497 1498 1498 1499 1499 1500 1500 1501 1501 1502 1502 1503 1503 1504 1504 1505 1505 1506 1506 1507 1507 1508 1508 1509 1509 1510 1510 1511 1511 1512 1512 1513 1513 1514 1514 1515 1515 1516 1516 1517 1517 1518 1518 1519 1519 1520 1520 1521 1521 1522 1522 1523 1523 1524 1524 1525 1525 1526 1526 1527 1527 1528 1528 1529 1529 1530 1530 1531 1531 1532 1532 1533 1533 1534 1534 1535 1535 1536 1536 1537 1537 1538 1538 1539 1539 1540 1540 1541 1541 1542 1542 1543 1543 1544 1544 1545 1545 1546 1546 1547 1547 1548 1548 1549 1549 1550 1550 1551 1551 1552 1552 1553 1553 1554 1554 1555 1555 1556 1556 1557 1557 1558 1
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like AI LAB 11.1 TB.py, WEB LAB 4 HTML.html, WEB LAB 5 HTML.html, web 2.py, 12.4 AI LAB T1.py, AI T 2 12.4 LAB.py, and AI T 3 12.4 LAB.py.
- Code Editor:** Displays a Python script for binary search. The code defines a function `binary_search` that takes a sorted list and a target value. It uses a while loop to find the target, adjusting the search range based on whether the mid-point value is less than or greater than the target. A helper function `make_random_list` generates a list of random integers.
- Terminal:** Shows the command line output of running the script. It includes timing measurements for linear and binary search on both sorted and unsorted data sets, along with a correctness check.
- Bottom Status Bar:** Shows system information like temperature (25°C), battery level (30%), connection status (No connection), and system time (25-09-2025).

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like AI LAB 11.1 TB.py, WEB LAB 4 HTML.html, WEB LAB 5 HTML.html, web 2.py, 12.4 AI LAB T1.py, AI T 2 12.4 LAB.py, and AI T 3 12.4 LAB.py.
- Code Editor:** Displays a Python script for benchmarking search functions. It defines a `make_sorted_list` function to generate a sorted list of integers and a `benchmark` function to measure the time taken to find a target value in an array, repeated multiple times. A `main` function tests these with a list size of 100,000.
- Terminal:** Shows the command line output of running the script. It includes timing measurements for linear and binary search on both sorted and unsorted data sets, along with a correctness check.
- Bottom Status Bar:** Shows system information like temperature (25°C), battery level (30%), connection status (No connection), and system time (25-09-2025).

```

C:\> Users > kadal > anaconda3 > pkgs > alembic-1.16.4-py313haa95532_0 > Lib > site-packages > alembic > ddl > AI T 12.4 LAB.py > ...
90 def main():
91     # Create unsorted list and sorted list
92     unsorted = make_random_list(n)
93     sorted_list = sorted(unsorted)
94     # Or simply: sorted_list = make_sorted_list(n)
95
96     # Pick a target (choose something that is in the list)
97     # For safety, pick from sorted_list
98     target_value = sorted_list[n // 2]
99
100    print("== Testing on Unsorted Data ==")
101    t_lin = benchmark(linear_search, unsorted, target_value, repeat=5)
102    t_bin_unsorted = None
103
104    try:
105        # We can "call" binary search on unsorted data, but it will give incorrect result
106        t_bin_unsorted = benchmark(binary_search, unsorted, target_value, repeat=5)
107    except Exception as e:
108        print("Binary search on unsorted data failed!: ", e)
109    print(f"Linear search time (unsorted) : {t_lin:.6f} s")
110    if t_bin_unsorted is not None:
111        print(f"Binary search time (unsorted, invalid) : {t_bin_unsorted:.6f} s (result may be wrong)")

112    == Testing on Sorted Data ==
113    Linear search time (unsorted) : 0.000093 s
114    Binary search time (unsorted, invalid) : 0.000034 s (result may be wrong)

115    == Testing on Sorted Data ==
116    Linear search time (sorted list) : 0.025805 s
117    Binary search time (sorted list) : 0.000036 s
118
119    Correctness check: target = 49805
120    Linear search found at index: 50000

```

```

C:\> Users > kadal > anaconda3 > pkgs > alembic-1.16.4-py313haa95532_0 > Lib > site-packages > alembic > ddl > AI T 12.4 LAB.py > ...
99 def main():
100    print("== Testing on Sorted Data ==")
101    t_lin_sorted = benchmark(linear_search, sorted_list, target_value, repeat=5)
102    t_bin_sorted = benchmark(binary_search, sorted_list, target_value, repeat=5)
103    print(f"Linear search time (sorted list) : {t_lin_sorted:.6f} s")
104    print(f"Binary search time (sorted list) : {t_bin_sorted:.6f} s")

105    # Validate correctness
106    idx_lin = linear_search(sorted_list, target_value)
107    idx_bin = binary_search(sorted_list, target_value)
108    print(f"Correctness check: target = {target_value}")
109    print(f"Linear search found at index: {idx_lin}")
110    print(f"Binary search found at index: {idx_bin}")

111    print("== Summary & When to Prefer Binary Search ==")
112    print("1. Linear search works whether or not the data is sorted; binary search requires sorted data.")
113    print("2. Linear search is O(n), while binary search is O(log n) on sorted data.")
114    print("3. For small arrays or one-time searches, the overhead of binary search may not pay off.")
115    print("4. If you perform many searches on the same sorted list, binary search is much more efficient overall.")
116    print("5. If the data is unsorted and you only search once, linear search may be preferable (to avoid sorting).")
117    print("6. In practice, for large n and many searches, binary search is the better choice. (Binary search is preferred for large sorted datasets.)")

118    if __name__ == "__main__":
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700

```

Correctness check: target = 49805  
 Linear search found at index: 50000  
 Binary search found at index: 50000

== Summary & When to Prefer Binary Search ==

1. Linear search works whether or not the data is sorted; binary search requires sorted data.
2. Linear search is  $O(n)$ , while binary search is  $O(\log n)$  on sorted data.
3. For small arrays or one-time searches, the overhead of binary search may not pay off.
4. If you perform many searches on the same sorted list, binary search is much more efficient overall.
5. If the data is unsorted and you only search once, linear search may be preferable (to avoid sorting).
6. In practice, for large  $n$  and many searches, binary search is the better choice. (Binary search is preferred for large sorted datasets.)

## EXPLAINATION:

**Linear Search** is simple and works on **any** list (sorted or unsorted).

- It scans each element until it finds the target or reaches the end.
- Time complexity is  $O(n)$  in the worst and average cases. [Wikipedia](#)
- It has very low overhead, which may make it competitive for small arrays or simple uses.

**Binary Search** is much faster if the array is sorted.

- It repeatedly divides the search space in half. [Wikipedia](#)
- Time complexity is  $O(\log n)$  in worst and average cases. [Wikipedia +1](#)
- But it **requires** the data to be sorted first.
- If you have to sort the list just to use binary search once, sorting time ( $O(n \log n)$ ) may dominate.

**When Binary Search is preferable:**

1. Your data is already sorted (or you can afford keeping it sorted).
2. You will perform **many** searches on the same dataset (then the one-time sorting cost is amortized).
3. The dataset is large, so the logarithmic reduction in comparisons is meaningful.
4. For small arrays or one-off searches, linear search's simplicity and minimal overhead may beat binary search in practice. (Some benchmarks show that for small  $n$ , linear search can be faster because binary search has more branch logic, etc.) [dirtyhandsCoding... +1](#)

## TASK 4:

### Quick Sort and Merge Sort Comparison

- Task: Implement Quick Sort and Merge Sort using recursion.
- Instructions:
  - o Provide AI with partially completed functions for recursion.
  - o Ask AI to complete the missing logic and add docstrings.
  - o Compare both algorithms on random, sorted, and reverse-sorted lists.

## PROMPT:

Generate python code and quick Sort and Merge Sort Comparison

- Task: Implement Quick Sort and Merge Sort using recursion.
- Instructions:
  - o Provide AI with partially completed functions for recursion.
  - o Ask AI to complete the missing logic and add docstrings.
  - o Compare both algorithms on random, sorted, and reverse-sorted lists.

## CODE & OUTPUT:

The screenshot shows the Microsoft Visual Studio Code interface with the Python extension installed. The code editor displays two files: `web 2.py` and `AI T 12.4 LAB.py`. The `AI T 12.4 LAB.py` file contains the following Python code:

```
C:\> Users > kadal > anaconda3 > pkgs > alembic-1.16.4-py313hae95532_0 > Lib > site-packages > alembic > ddl > AI T 12.4 LAB.py > ...
1 import random
2 import time
3
4 # --- Quick Sort Implementation ---
5 def quick_sort_recursive(arr, low, high):
6     """
7         Recursively sorts an array using the Quick Sort algorithm.
8     """
9     if low < high:
10         pi = partition(arr, low, high)
11         quick_sort_recursive(arr, low, pi - 1)
12         quick_sort_recursive(arr, pi + 1, high)
13
14 def partition(arr, low, high):
15     """
16         Partitions the array and returns the pivot index.
17     """
18     pivot = arr[high]
19     i = low - 1
20     for j in range(low, high):
21         if arr[j] <= pivot:
22             i += 1
23             arr[i], arr[j] = arr[j], arr[i]
```

The terminal window below the code editor shows the execution of the code and the results of a performance comparison:

```
crosssoft\WindowsApps\python.11.exe "c:\Users\kadal\vscode\extensions\ms-python.debugger-2025.10.0-win32-x64\bundled\libs\debug\launcher" "55214" -- "c:\Users\kadal\anaconda3\pkgs\alembic-1.16.4-py313hae95532_0\lib\site-packages\alembic\ddl\VAE 12.4 LAB.py"
--- Performance Comparison (List Size: 50000) ---
--- random List ---
Quick Sort time: 1.0666 seconds
Merge Sort time: 1.4798 seconds
--- Sorted List ---
```

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure with files like LAB 4 HTML.html, WEB LAB 5 HTML.html, web 2.py, AI T 2 12.4 LAB.py, AI T 3 12.4 LAB.py, AI T 4 12.4 LAB.py, and web lab.py.
- Search Bar:** At the top center.
- Run and Debug:** A dropdown menu at the top left.
- VARIABLES:** A sidebar showing variable values: arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], len(arr) = 10, i = 0, j = 0, k = 0.
- WATCH:** A sidebar showing watched variables.
- Code Editor:** The main area contains Python code for sorting algorithms:

```
def quick_sort(arr):
    """
    Wrapper function for Quick Sort.
    """
    quick_sort_recursive(arr, 0, len(arr) - 1)

# --- Merge Sort Implementation ---
def merge_sort(arr):
    """
    Sorts an array using the Merge Sort algorithm.
    """
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0
```
- PROBLEMS:** Shows 1 error: "c:\Users\kadal\site-packages\sqlalchemy\util\compat.py:11: DeprecationWarning: 'c:\Users\kadal\vscode\extensions\ms-python.python\2025.10.0-win32-x64\bundled\libs\debug\launcher' is being soft-imported by a module (ai\_t\_4\_12\_4\_lab.py)"
- OUTPUT:** Shows the output of the Python Debug Console:

```
crossref\windows\deps\python3.11.exe: 'c:\Users\kadal\vscode\extensions\ms-python.python\2025.10.0-win32-x64\bundled\libs\debug\launcher' '55234' '---' 'C:\Users\kadal\an
bconda\pkgs\alab[<1.16->]\haa9532\0\lib\site-packages\sqlalchemy\util\compat.py'
--- Performance Comparison (List size: 50000) ---
```
- TERMINAL:** Shows the terminal output:

```
--- Random List ---
Quick Sort time: 1.0666 seconds
Merge Sort time: 1.4798 seconds

--- Sorted List ---
```
- CALL STACK:** Shows the call stack for the current process.
- BREAKPOINTS:** Shows the current breakpoints.
- STATUS BAR:** Shows connection status, file paths, and system information.

File Edit Selection View Go Run Terminal Help ← → Search RUN AND DEBUG ... LAB 4 HTML.html WEB LAB 5 HTML.html web 2.py 9+ AI T 2 12.4 LAB.py AI T 3 12.4 LAB.py AI T 4 12.4 LAB.py x web lab.py ...

VARIABLES

WATCH

```
48     # Merge the two halves
49     while i < len(left_half) and j < len(right_half):
50         if left_half[i] < right_half[j]:
51             arr[k] = left_half[i]
52             i += 1
53         else:
54             arr[k] = right_half[j]
55             j += 1
56         k += 1
57
58     # Copy any remaining elements
59     while i < len(left_half):
60         arr[k] = left_half[i]
61         i += 1
62         k += 1
63
64     while j < len(right_half):
65         arr[k] = right_half[j]
66         j += 1
67         k += 1
68
69 # --- Comparison ---
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

crossfire\Windows\appdata\local\temp\1\pip-req-build-11\python\2.7\lib\site-packages\alibic\lab.py

--- Performance Comparison (List Size: 50000) ---

CALL STACK Running

--- Random List ---  
Quick Sort time: 1.0666 seconds  
Merge Sort time: 1.4798 seconds

--- Sorted List ---

Breakpoints

10 5 No connection

Search

11:29:31 ENG IN 25-09-2025

The screenshot shows the Visual Studio Code interface with the Python extension installed. The code editor displays a script named `AI T4 12.4 LAB.py` containing a function `run_comparison()` which compares Quick Sort and Merge Sort on different list types. The Python Debug Console at the bottom shows the execution of the script and its output, including timing results for each sort type on random and sorted lists.

```
C:\> Users > kadal > anaconda3 > pkgs > alembic-1.16.4-py313haa95532_0 > Lib > site-packages > alembic > ddl > AI T4 12.4 LAB.py > ...
70 def run_comparison():
71     """
72         Compares Quick Sort and Merge Sort on different list types.
73     """
74     list_size = 50000
75
76     # Test cases
77     random_list = [random.randint(0, list_size) for _ in range(list_size)]
78     sorted_list = sorted(random_list)
79     reverse_sorted_list = sorted(random_list, reverse=True)
80
81     print("---- Performance Comparison (List Size: {}) ----".format(list_size))
82
83     # Test on Random List
84     print("\n--- Random List ---")
85
86     # Quick Sort
87     quick_list = list(random_list)
88     start_time = time.time()
89     quick_sort(quick_list)
90     end_time = time.time()
91     print("Quick Sort time: {:.4f} seconds".format(end_time - start_time))
92
93     # Merge Sort
94     merge_list = list(random_list)
95     start_time = time.time()
96     merge_sort(merge_list)
97     end_time = time.time()
98     print("Merge Sort time: {:.4f} seconds".format(end_time - start_time))
99
100    # Test on Sorted List
101    print("\n--- Sorted List ---")
102
103    # Quick Sort
104    quick_list = list(sorted_list)
105    start_time = time.time()
106    quick_sort(quick_list)
107    end_time = time.time()
108    print("Quick Sort time: {:.4f} seconds".format(end_time - start_time))
109
110    # Merge Sort
111    merge_list = list(sorted_list)
112    start_time = time.time()
113    merge_sort(merge_list)
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2198
```

```

70 def run_comparison():
115     print("Merge Sort time: {:.4f} seconds".format(end_time - start_time))
116
117     # Test on Reverse Sorted List
118     print("\n--- Reverse Sorted List ---")
119
120     # Quick Sort
121     quick_list = list(reverse_sorted_list)
122     start_time = time.time()
123     quick_sort(quick_list)
124     end_time = time.time()
125     print("Quick Sort time: {:.4f} seconds".format(end_time - start_time))
126
127     # Merge Sort
128     merge_list = list(reverse_sorted_list)
129     start_time = time.time()
130     merge_sort(merge_list)
131     end_time = time.time()
132     print("Merge Sort time: {:.4f} seconds".format(end_time - start_time))
133
134 if __name__ == "__main__":
135     run_comparison()

```

PROBLEMS 35 OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

```

crossoft\WindowsApps\python3.11.exe' 'c:\Users\kadal\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debu
aconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl\AI_T4_12.4_LAB.py'
--- Performance Comparison (List Size: 50000) ---

--- Random List ---
Quick Sort time: 1.0666 seconds
Merge Sort time: 1.4798 seconds

```

## EXPLANATION:

### Time Complexity (average / worst / best)

- **Quick Sort:** average-case  $O(n \log n)$ , but worst-case  $O(n^2)$  (e.g. if pivot choices are poor).

[Wikipedia +2](#)

- **Merge Sort:**  $O(n \log n)$  in all cases (best, average, worst).

[Wikipedia +2](#)

### Space Complexity

- Quick Sort (this version) uses extra lists for partitioning, so not truly in-place; but an in-place variant (Lomuto / Hoare partition) can be done with  $O(\log n)$  stack usage.
- Merge Sort needs  $O(n)$  extra space for merging (the merged list) plus recursion overhead.

[Wikipedia +1](#)

### Stability

- Merge Sort is stable (does not reorder equal elements).
- Quick Sort is generally **not stable** (depends on partition scheme).

[Wikipedia +1](#)

[GeeksforGeeks +1](#)

### Partition Behavior vs Splitting

- Quick Sort partitions around a pivot; the partitions may be unbalanced, introducing risk of worst-case.
- Merge Sort always splits evenly (midpoint), so its recursion depth is predictable.

[GeeksforGeeks +2](#)

[Wikipedia +2](#)

#### Practical Performance & Use Cases

- Quick Sort often outperforms Merge Sort for in-memory arrays because of lower constant factors, better cache locality, and no large temporary arrays. Stack Overflow +2
- Merge Sort is preferable when you require stability, or when worst-case guarantees matter (e.g. real-time systems).
- Merge Sort is often used in external sorting (on disk) because its merging approach accesses data sequentially. GeeksforGeeks +1
- For linked lists, Merge Sort is often a better choice since you can merge without needing additional arrays and random access is expensive. GeeksforGeeks +1

## TASK 5:

### AI-Suggested Algorithm Optimization

- Task: Give AI a naive algorithm (e.g.,  $O(n^2)$  duplicate search).
  - Instructions:
    - Students write a brute force duplicate-finder.
    - Ask AI to optimize it (e.g., by using sets/dictionaries with  $O(n)$  time).
    - Compare execution times with large input sizes.

## PROMPT:

### Generate python code and AI-Suggested Algorithm Optimization

- Task: Give AI a naive algorithm (e.g.,  $O(n^2)$  duplicate search).
  - Instructions:
    - Students write a brute force duplicate-finder.

- o Ask AI to optimize it (e.g., by using sets/dictionaries with  $O(n)$  time).
  - o Compare execution times with large input sizes.

## CODE & OUTPUT:

The screenshot shows a Python development environment with the following details:

- File Explorer:** Shows a file tree with files like `ml`, `WEB LAB 5 HTML.html`, `web 2.py`, `12.4 AI LAB.py`, `.py`, `AI T3 12.4 LAB.py`, `AI T4 12.4 LAB.py`, and `AI T5 12.4 LAB.py`.
- Run and Debug:** A dropdown menu showing options like `RUN`, `DEBUG`, `LAUNCH`, etc.
- VARIABLES:** A sidebar showing variable definitions for `find_duplicates_bruteforce`.

```
C:\> Users > kadal > anaconda3\pkgs\alembic-1.16.4-py313ha95532_0\lib\site-packages\alembic\ddl.py
1 import random
2 import time
3 from typing import List, Any, Optional
4
5 def find_duplicates_bruteforce(lst: List[Any]) -> List[Any]:
6     """
7         Naive duplicate finder (brute force).
8
9     Args:
10        lst: List of items (can be any hashable or comparable values).
11    Returns:
12        A list of values that appear more than once in lst.
13        May include duplicates in output if the same duplicate is found multiple times.
14
15    Time complexity: O(n^2) in worst and average cases (two nested loops).
16    Space complexity: O(n) for the output list (plus whatever the input uses).
17    """
18
19    n = len(lst)
20    duplicates = []
21    for i in range(n):
22        for j in range(i + 1, n):
23            if lst[i] == lst[j]:
24                # If not already recorded, add to duplicates
```
- WATCH:** A sidebar showing watched variables.
- PROBLEMS:** A sidebar showing no problems.
- OUTPUT:** A tab showing the command line output of the script execution.
- DEBUT CONSOLE:** A tab showing the debug console.
- TERMINAL:** A tab showing the terminal.
- PORTS:** A tab showing ports.
- QUERY RESULTS:** A tab showing query results.
- Python Debug Console:** A panel showing the execution results of the script, including benchmarking information.
- CALL STACK:** A sidebar showing the call stack.
- BREAKPOINTS:** A sidebar showing breakpoints.
- Bottom Status Bar:** Shows system status (25°C), network connection (No connection), and system info (Spaces: 4, UFT: 8, Python 3.11.9 (Microsoft Store)).

The screenshot shows the VS Code interface with the Python extension installed. The code editor displays a Python script with two functions: `find_duplicates_bruteforce` and `find_duplicates_optimized`. The `find_duplicates_optimized` function uses a set to track seen items, resulting in a time complexity of  $O(n)$ . The `Python Debug Console` tab shows the execution of the script for three different input sizes (n=1000, n=2000, n=5000) and compares the execution times for both methods. The output indicates that the optimized method is significantly faster for larger inputs.

```
C:\> Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl & "c:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/kadal/.vscode/extensions/ms-python.python.debug-2025.10.0-win32-x64/bundled/libs/debugpy\launcher" "55274" "--" "C:/Users/kadal/anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl\AI T5 12.4 LAB.py"
Benchmark: brute force vs optimized duplicate finder

n = 1000 | brute: 0.223449 s | optimized: 0.000631 s
n = 2000 | brute: 0.950304 s | optimized: 0.001714 s
n = 5000 | brute: 6.414864 s | optimized: 0.003992 s
```

This screenshot shows the same VS Code environment with a different Python script. The code defines a `make_random_list` function that generates a list of random integers. It also includes a `benchmark` function that measures the execution time of a provided function `fn` over multiple iterations. The `Python Debug Console` output shows the execution times for the `make_random_list` function and the `benchmark` function for various input sizes (n=1000, n=2000, n=5000).

```
C:\> Users\kadal>anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl & "c:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/kadal/.vscode/extensions/ms-python.python.debug-2025.10.0-win32-x64/bundled/libs/debugpy\launcher" "55274" "--" "C:/Users/kadal/anaconda3\pkgs\alembic-1.16.4-py313haa95532_0\Lib\site-packages\alembic\ddl\AI T5 12.4 LAB.py"
Benchmark: brute force vs optimized duplicate finder

n = 1000 | brute: 0.223449 s | optimized: 0.000631 s
n = 2000 | brute: 0.950304 s | optimized: 0.001714 s
n = 5000 | brute: 6.414864 s | optimized: 0.003992 s
```

```

C:\> Users\kadal> anaconda3\pkgs> alembic-1.16.4-py313haa95532_0> Lib> site-packages> alembic> ddl> AI T5 12.4 LAB.py > ...
63 def benchmark(fn, data: List[Any], repeat: int = 1) -> float:
    total = 0.0
    for _ in range(repeat):
        arr = data.copy()
        start = time.perf_counter()
        fn(arr)
        end = time.perf_counter()
        total += (end - start)
    return total / repeat

78 def main():
    # Test sizes
    sizes = [1000, 2000, 5000, 10000]
    # For each size, generate a random list with many duplicates (smaller range)
    print("Benchmark: brute force vs optimized duplicate finder")
    for n in sizes:
        lst = make_random_list(n, value_range = max(10, n // 10))
        t_brute = benchmark(find_duplicates_bruteforce, lst, repeat=3)
        t_opt = benchmark(find_duplicates_optimized, lst, repeat=3)
        print(f"\n{n} | brute: {t_brute:.6f} s | optimized: {t_opt:.6f} s")

89 # Demonstrate correctness
90 sample = [5, 1, 3, 5, 2, 3, 8, 5]
91 print("\nSample list:", sample)
92 print("Brute duplicates : ", find_duplicates_bruteforce(sample))
93 print("Optimized duplicates: ", find_duplicates_optimized(sample))

96 if __name__ == "__main__":
97     main()
98

```

Python Debug Console

```

n = 1000 | brute: 0.223449 s | optimized: 0.000631 s
n = 2000 | brute: 0.950304 s | optimized: 0.001714 s
n = 5000 | brute: 6.414864 s | optimized: 0.003992 s
n = 10000 | brute: 43.288533 s | optimized: 0.007612 s

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

```

C:\> Users\Kadal> anaconda3\pkgs> alembic-1.16.4-py313haa95532_0> Lib> site-packages> alembic> ddl> AI T5 12.4 LAB.py > ...
78 def main():
    for n in sizes:
        lst = make_random_list(n, value_range = max(10, n // 10))
        t_brute = benchmark(find_duplicates_bruteforce, lst, repeat=3)
        t_opt = benchmark(find_duplicates_optimized, lst, repeat=3)
        print(f"\n{n} | brute: {t_brute:.6f} s | optimized: {t_opt:.6f} s")

89 # Demonstrate correctness
90 sample = [5, 1, 3, 5, 2, 3, 8, 5]
91 print("\nSample list:", sample)
92 print("Brute duplicates : ", find_duplicates_bruteforce(sample))
93 print("Optimized duplicates: ", find_duplicates_optimized(sample))

96 if __name__ == "__main__":
97     main()
98

```

Python Debug Console

```

n = 1000 | brute: 0.223449 s | optimized: 0.000631 s
n = 2000 | brute: 0.950304 s | optimized: 0.001714 s
n = 5000 | brute: 6.414864 s | optimized: 0.003992 s
n = 10000 | brute: 43.288533 s | optimized: 0.007612 s

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

```

n = 1000 | brute: 0.223449 s | optimized: 0.000631 s
n = 2000 | brute: 0.950304 s | optimized: 0.001714 s
n = 5000 | brute: 6.414864 s | optimized: 0.003992 s
n = 10000 | brute: 43.288533 s | optimized: 0.007612 s

Sample list: [5, 1, 3, 5, 2, 3, 8, 5]
Brute duplicates : [5, 3]
Optimized duplicates: [3, 5]

```

## EXPLANATION:

- The **brute force version** uses nested loops (`i` and `j`) to compare every pair. That gives  $O(n^2)$  time complexity. It also checks `lst[i] not in duplicates` to avoid adding the same duplicate multiple times, but that check itself is  $O(n)$  in worst case, making it even slower in practice.
- The **optimized version** uses two sets:
  - `seen` to record which items have already been encountered.
  - `duplicates` to record items that appear more than once (without repetition).
  - For each item, you do `if item in seen: duplicates.add(item) else seen.add(item)`.
  - Both set membership / insertion operations are average  $O(1)$ , so total is  $O(n)$  average time.
- We convert `duplicates` set to list at the end for output; you could also return a set directly.
- The benchmarking compares the two methods on random lists with many duplicates (using a relatively small `value_range`) so collisions / duplicates are common — that better highlights the speed difference.

You'll observe that for moderate `n` (e.g. 5,000+), the brute force method becomes very slow, while the optimized one scales linearly.