

ASSIGNMENT-12.4

2303A510E3

BATCH-14

Task 1: Bubble Sort for Ranking Exam

Scores

Scenario

You are working on a college result processing system where a small

list of student scores needs to be sorted after every internal assessment.

Task Description

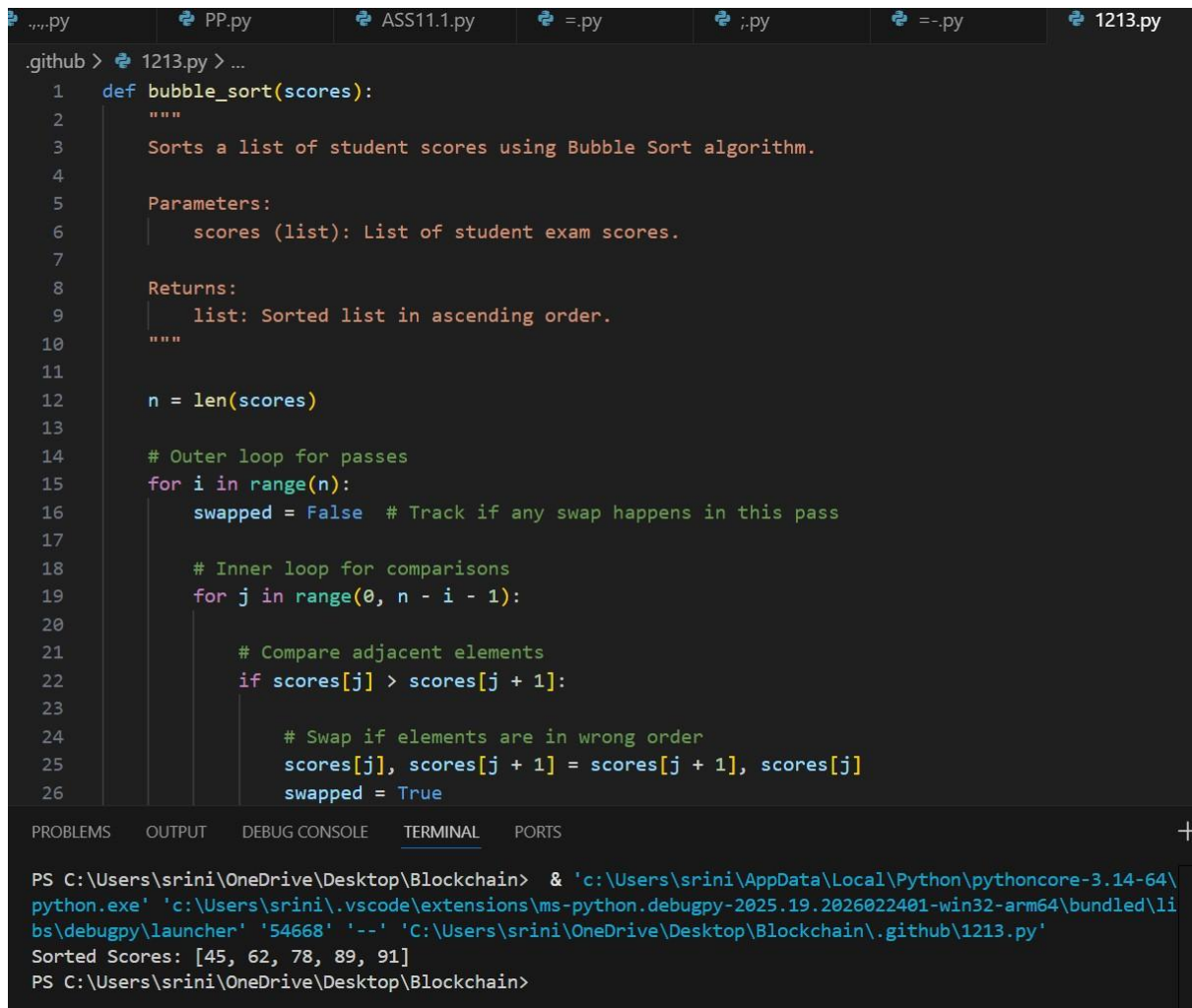
- Implement Bubble Sort in Python to sort a list of student scores.
- Use an AI tool to:
 - o Insert inline comments explaining key operations such as comparisons, swaps, and iteration passes
 - o Identify early-termination conditions when the list becomes sorted
 - o Provide a brief time complexity analysis

Expected Outcome

- A Bubble Sort implementation with:
 - o AI-generated comments explaining the logic
 - o Clear explanation of best, average, and worst-case

complexity

o Sample input/output showing sorted scores



```
1 def bubble_sort(scores):
2     """
3     Sorts a list of student scores using Bubble Sort algorithm.
4
5     Parameters:
6     | scores (list): List of student exam scores.
7
8     Returns:
9     | list: Sorted list in ascending order.
10    """
11
12    n = len(scores)
13
14    # Outer loop for passes
15    for i in range(n):
16        swapped = False # Track if any swap happens in this pass
17
18        # Inner loop for comparisons
19        for j in range(0, n - i - 1):
20
21            # Compare adjacent elements
22            if scores[j] > scores[j + 1]:
23
24                # Swap if elements are in wrong order
25                scores[j], scores[j + 1] = scores[j + 1], scores[j]
26                swapped = True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\srini\OneDrive\Desktop\Blockchain> & 'c:\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\srini\.vscode\extensions\ms-python.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '54668' '--' 'C:\Users\srini\OneDrive\Desktop\Blockchain\.github\1213.py'
Sorted Scores: [45, 62, 78, 89, 91]
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
```

Task 2: Improving Sorting for Nearly Sorted

Attendance Records

Scenario

You are maintaining an attendance system where student roll numbers

are already almost sorted, with only a few late updates.

Task Description

- Start with a Bubble Sort implementation.

- Ask AI to:

- o Review the problem and suggest a more suitable sorting algorithm

- o Generate an Insertion Sort implementation

- o Explain why Insertion Sort performs better on nearly sorted data

- Compare execution behavior on nearly sorted input

Expected Outcome

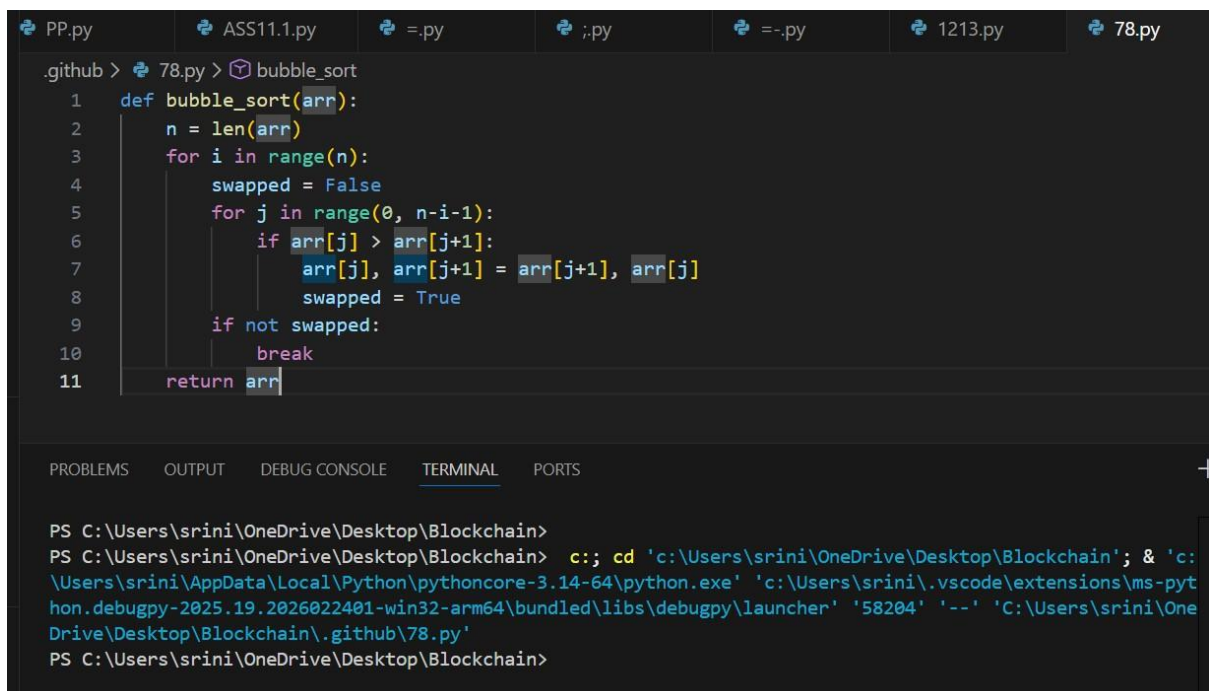
- Two sorting implementations:

- o Bubble Sort

- o Insertion Sort

- AI-assisted explanation highlighting efficiency differences for partially sorted datasets

Bubble Sort



```
PP.py  ASS11.1.py  =.py  ;.py  =-.py  1213.py  78.py

.github > 78.py > bubble_sort
1  def bubble_sort(arr):
2      n = len(arr)
3      for i in range(n):
4          swapped = False
5          for j in range(0, n-i-1):
6              if arr[j] > arr[j+1]:
7                  arr[j], arr[j+1] = arr[j+1], arr[j]
8                  swapped = True
9          if not swapped:
10             break
11     return arr

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\srini\OneDrive\Desktop\Blockchain>
PS C:\Users\srini\OneDrive\Desktop\Blockchain> c::; cd 'c:\Users\srini\OneDrive\Desktop\Blockchain'; & 'c:
\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '58204' '--' 'C:\Users\srini\One
Drive\Desktop\Blockchain\.github\78.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
```

Insertion Sort Implementation

```
.github > 0-.py > ...
1  def insertion_sort(arr):
2      """
3      Sorts list using Insertion Sort.
4      Efficient for nearly sorted data.
5      """
6
7      for i in range(1, len(arr)):
8          key = arr[i] # Current element
9          j = i - 1
10
11         # Shift elements greater than key
12         while j >= 0 and arr[j] > key:
13             arr[j + 1] = arr[j]
14             j -= 1
15
16         arr[j + 1] = key # Insert at correct position
17
18     return arr
19
20
21 # Nearly sorted input
22 roll_numbers = [101, 102, 103, 105, 104]
23
24 print("Insertion Sort Result:", insertion_sort(roll_numbers))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Drive\Desktop\Blockchain\.github\0-.py'
Insertion Sort Result: [101, 102, 103, 104, 105]
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
```

Task 3: Searching Student Records in a Database

Scenario

You are developing a student information portal where users search for

student records by roll number.

Task Description

- Implement:
 - o Linear Search for unsorted student data
 - o Binary Search for sorted student data
- Use AI to:
 - o Add docstrings explaining parameters and return values
 - o Explain when Binary Search is applicable
 - o Highlight performance differences between the two searches

Expected Outcome

- Two working search implementations with docstrings
- AI-generated explanation of:
 - o Time complexity
 - o Use cases for Linear vs Binary Search
- A short student observation comparing results on sorted vs
unsorted lists

Linear Search Implementation

.github > YU.py > linear_search

```
1 def linear_search(data, target):
2     """
3     Searches for target roll number in an unsorted list.
4
5     Parameters:
6         data (list): List of student roll numbers.
7         target (int): Roll number to search.
8
9     Returns:
10        int: Index of target if found, otherwise -1.
11    """
12
13    for i in range(len(data)):
14        if data[i] == target:
15            return i
16    return -1
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\sринi\OneDrive\Desktop\Blockchain>
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> c;; cd 'c:\Users\sринi\OneDrive\Desktop\Blockchain'; & 'c:
\Users\sринi\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\sринi\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '63196' '--' 'C:\Users\sринi\One
Drive\Desktop\Blockchain\.github\YU.py'
PS C:\Users\sринi\OneDrive\Desktop\Blockchain>
```

.github > BH.py > binary_search

```
1 def binary_search(data, target):
2     """
3     Searches for target roll number in a sorted list.
4
5     Parameters:
6         data (list): Sorted list of roll numbers.
7         target (int): Roll number to search.
8
9     Returns:
10        int: Index if found, else -1.
11    """
12
13    left = 0
14    right = len(data) - 1
15
16    while left <= right:
17        mid = (left + right) // 2
18
19        if data[mid] == target:
20            return mid
21        elif data[mid] < target:
22            left = mid + 1
23        else:
24            right = mid - 1
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\sринi\OneDrive\Desktop\Blockchain>
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> c;; cd 'c:\Users\sринi\OneDrive\Desktop\Blockchain'; & 'c:
\Users\sринi\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\sринi\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '63959' '--' 'C:\Users\sринi\One
Drive\Desktop\Blockchain\.github\BH.py'
PS C:\Users\sринi\OneDrive\Desktop\Blockchain>
```

Task 4: Choosing Between Quick Sort and Merge Sort for Data Processing

Scenario

You are part of a data analytics team that needs to sort large datasets

received from different sources (random order, already sorted, and reverse sorted).

Task Description

- Provide AI with partially written recursive functions for:

- o Quick Sort

- o Merge Sort

- Ask AI to:

- o Complete the recursive logic

- o Add meaningful docstrings

- o Explain how recursion works in each algorithm

- Test both algorithms on:

- o Random data

- o Sorted data

- o Reverse-sorted data

Expected Outcome

- Fully functional Quick Sort and Merge Sort implementations

- AI-generated comparison covering:

- o Best, average, and worst-case complexities
- o Practical scenarios where one algorithm is preferred over the other

Quick Sort Implementation

```
.github > JI.py > quick_sort
1  def quick_sort(arr):
2      """
3      Sorts list using Quick Sort (Divide and Conquer).
4      """
5
6      if len(arr) <= 1:
7          return arr
8
9      pivot = arr[len(arr) // 2]
10
11     left = [x for x in arr if x < pivot]
12     middle = [x for x in arr if x == pivot]
13     right = [x for x in arr if x > pivot]
14
15     return quick_sort(left) + middle + quick_sort(right)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sринi\OneDrive\Desktop\Blockchain>
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> c::; cd 'c:\Users\sринi\OneDrive\Desktop\Blockchain'; & 'c:
\Users\sринi\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\sринi\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundle\libs\debugpy\launcher' '63959' '--' 'C:\Users\sринi\One
Drive\Desktop\Blockchain\.github\BH.py'
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> ^C
Drive\Desktop\Blockchain\.github\BH.py'
Drive\Desktop\Blockchain\.github\BH.py'
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\sринi\OneDrive\Desktop\Blockchain>
PS C:\Users\sринi\OneDrive\Desktop\Blockchain> c::; cd 'c:\Users\sринi\OneDrive\Desktop\Blockchain'; & 'c:
\Users\sринi\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\sринi\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundle\libs\debugpy\launcher' '53060' '--' 'C:\Users\sринi\One
Drive\Desktop\Blockchain\.github\JI.py'
PS C:\Users\sринi\OneDrive\Desktop\Blockchain>
```

Merge Sort Implementation


```
.github > NH.py > merge
1  def merge_sort(arr):
2      """
3      Sorts list using Merge Sort algorithm.
4      """
5
6      if len(arr) <= 1:
7          return arr
8
9      mid = len(arr) // 2
10
11     left = merge_sort(arr[:mid])
12     right = merge_sort(arr[mid:])
13
14     return merge(left, right)
15
16
17 def merge(left, right):
18     result = []
19     i = j = 0
20     while i < len(left) and j < len(right):
21         if left[i] < right[j]:
22             result.append(left[i])
23             i += 1
24         else:
25             result.append(right[j])
26             j += 1
27     result.extend(left[i:])
28     result.extend(right[j:])
29     return result
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Drive\Desktop\Blockchain\.github\BH.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
PS C:\Users\srini\OneDrive\Desktop\Blockchain> c::; cd 'c:\Users\srini\OneDrive\Desktop\Blockchain'; & 'c:
\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '53060' '--' 'C:\Users\srini\One
Drive\Desktop\Blockchain\.github\JI.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
PS C:\Users\srini\OneDrive\Desktop\Blockchain> c::; cd 'c:\Users\srini\OneDrive\Desktop\Blockchain'; & 'c:
\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '58147' '--' 'C:\Users\srini\One
Drive\Desktop\Blockchain\.github\NH.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
```

Task 5: Optimizing a Duplicate Detection

Algorithm

Scenario

You are building a data validation module that must detect duplicate

user IDs in a large dataset before importing it into a system.

Task Description

- Write a naive duplicate detection algorithm using nested loops.
- Use AI to:
 - o Analyze the time complexity

- o Suggest an optimized approach using sets or dictionaries
- o Rewrite the algorithm with improved efficiency
- Compare execution behavior conceptually for large input sizes

Expected Outcome

- Two versions of the algorithm:
 - o Brute-force ($O(n^2)$)
 - o Optimized ($O(n)$)
- AI-assisted explanation showing how and why performance

Improved

Naive Approach

```

.github > DF.py > find_duplicates_naive
1  def find_duplicates_naive(data):
2      """
3      Detect duplicates using nested loops.
4      Time Complexity: O(n^2)
5      """
6
7      duplicates = []
8
9      for i in range(len(data)):
10         for j in range(i + 1, len(data)):
11             if data[i] == data[j] and data[i] not in duplicates:
12                 duplicates.append(data[i])
13
14     return duplicates
  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '58147' '--' 'C:\Users\srini\One
Drive\Desktop\Blockchain\github\NH.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
PS C:\Users\srini\OneDrive\Desktop\Blockchain> c:: cd 'c:\Users\srini\OneDrive\Desktop\Blockchain'; & 'c:
\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '58855' '--' 'C:\Users\srini\One
Drive\Desktop\Blockchain\github\DF.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
  
```

Complexity Analysis

- Two nested loops $\rightarrow O(n^2)$

- Very slow for large datasets

Optimized Approach Using Set

```
.github > GH.py > find_duplicates_optimized
1 def find_duplicates_optimized(data):
2     """
3     Detect duplicates using a set.
4     Time Complexity: O(n)
5     """
6
7     seen = set()
8     duplicates = set()
9
10    for item in data:
11        if item in seen:
12            duplicates.add(item)
13        else:
14            seen.add(item)
15
16    return list(duplicates)
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
PS C:\Users\srini\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
PS C:\Users\srini\OneDrive\Desktop\Blockchain> c;; cd 'c:\Users\srini\OneDrive\Desktop\Blockchain'; & 'c:\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\srini\.vscode\extensions\ms-python.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '58855' '--' 'C:\Users\srini\OneDrive\Desktop\Blockchain\.github\DF.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain> ^C
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
PS C:\Users\srini\OneDrive\Desktop\Blockchain> c;; cd 'c:\Users\srini\OneDrive\Desktop\Blockchain'; & 'c:\Users\srini\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\srini\.vscode\extensions\ms-python.debugpy-2025.19.2026022401-win32-arm64\bundled\libs\debugpy\launcher' '54708' '--' 'C:\Users\srini\OneDrive\Desktop\Blockchain\.github\GH.py'
PS C:\Users\srini\OneDrive\Desktop\Blockchain>
```

Why Performance Improved?

- Set lookup is $O(1)$.
- Eliminated nested loops.
- Single traversal of dataset.

• Critical Evaluation of AI Suggestions

- ✓ AI correctly suggested Insertion Sort for nearly sorted data
- ✓ AI improved Bubble Sort using early termination
- ✓ AI recommended set-based optimization for duplicate detection
- ✓ AI provided correct complexity analysis