

2303A51504

Batch : 25

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

```

new.py
1 #Write a Python program to sort student exam scores using Bubble Sort with an early termination condition.
2 # Show sample input/output and explain time complexity (best, average, worst).
3 def bubble_sort(scores):
4     n = len(scores)
5     for i in range(n):
6         swapped = False
7         for j in range(0, n - i - 1):
8             if scores[j] > scores[j + 1]:
9                 scores[j], scores[j + 1] = scores[j + 1], scores[j]
10            swapped = True
11        if not swapped:
12            break
13    return scores
14
15 scores = list(map(int, input("Enter student scores: ").split()))
16 sorted_scores = bubble_sort(scores)
17
18 print("Sorted Scores:", sorted_scores)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Chinnari\Downloads> & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\Chinnari\vscode\extensions\ms-python.python-2025.18.0-win32\lib\site-packages\debugpy\launcher' '49686' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter student scores: 45 62 78 84 98
Sorted Scores: [45, 62, 78, 84, 98]

PS C:\Users\Chinnari\Downloads>

BREAKPOINTS

- Raised Exceptions
- Uncaptured Exceptions
- User Uncaptured Exceptions

Ln 18, Col 39 Spaces:4 UTF-8 CR/LF Python Python 3.11

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 give code and prompt for this remove comments

```

new.py
-----
1 #Compare Bubble Sort and Insertion Sort for nearly sorted student roll numbers. Suggest the better algorithm, generate both Python
2 #and show why Insertion Sort performs better on nearly sorted data with sample input/output.
3 def bubble_sort(arr):
4     n = len(arr)
5     for i in range(n):
6         swapped = False
7         for j in range(0, n - i - 1):
8             if arr[j] > arr[j + 1]:
9                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
10            swapped = True
11    if not swapped:
12        break
13 return arr
14
15 data = list(map(int, input("Enter roll numbers: ").split()))
16 print("Bubble Sorted:", bubble_sort(data.copy()))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Chinnari\Downloads> & 'C:\Users\chinnari\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\Chinnari\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '49686' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter student scores: 78 45 90 62 84
Sorted Scores: [45, 62, 78, 84, 90]
PS C:\Users\Chinnari\Downloads> c:; cd 'c:\Users\chinnari\Downloads'; & 'C:\Users\chinnari\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\chinnari\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62495' '--' 'c:\Users\Chinnari\Downloads\new.py'
Enter roll numbers: 1 2 3 5 4 6 7
Bubble Sorted: [1, 2, 3, 4, 5, 6, 7]
PS C:\Users\Chinnari\Downloads>

BREAKPOINTS

- Raised Exceptions
- Uncaught Exceptions
- User Uncaught Exceptions

Task 3: Searching Student Records in a Database

Database

Scenario

You are developing a student information portal where users search for student records by roll number.

Task Description

- Implement:
 - Linear Search for unsorted student data
 - Binary Search for sorted student data
- Use AI to:
 - Add docstrings explaining parameters and return values
 - Explain when Binary Search is applicable
 - 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

give code and prompt for this and remove comments

```
new.py
RUN AND DEBUG
RUN
File Edit Selection View Go Run Terminal Help ← → Q Search
C:\> Users > Chinnari > Downloads > new.py > ...
1 #Implement Linear Search and Binary Search in Python for student roll number search. Add docstrings explaining parameters and
2 # return values. Explain when Binary Search can be used, compare time complexity, and show sample input/output with observations on
3 def linear_search(data, target):
4     for i in range(len(data)):
5         if data[i] == target:
6             return i
7     return -1
8
9 def binary_search(data, target):
10    left = 0
11    right = len(data) - 1
12    while left <= right:
13        mid = (left + right) // 2
14        if data[mid] == target:
15            return mid
16        elif data[mid] < target:
17            left = mid + 1
18        else:
19            right = mid - 1
20    return -1
21
22 records = list(map(int, input("Enter roll numbers: ").split()))
23 target = int(input("Enter roll number to search: "))
24
25 lin_result = linear_search(records, target)
26 print("Linear Search:", "Found at index" if lin_result != -1 else "Not found", lin_result)
27
28 sorted_records = sorted(records)
29 bin_result = binary_search(sorted_records, target)
30 print("Binary Search (on sorted data):", "Found at index" if bin_result != -1 else "Not found", bin_result)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Chinnari\Downloads> & 'C:\Users\chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\chinnari\vscode\extensions\vs-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55845' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter roll numbers: 105 101 109 103 107
Enter roll number to search: 103
Linear Search: Found at index 3
Binary Search (on sorted data): Found at index 1
PS C:\Users\Chinnari\Downloads>

Ln 30, Col 108 Spaces: 4 UTF-8 CRLF () Python Python 3.13
```

```
new.py
RUN AND DEBUG
RUN
File Edit Selection View Go Run Terminal Help ← → Q Search
C:\> Users > Chinnari > Downloads > new.py > ...
8
9 def binary_search(data, target):
10    left = 0
11    right = len(data) - 1
12    while left <= right:
13        mid = (left + right) // 2
14        if data[mid] == target:
15            return mid
16        elif data[mid] < target:
17            left = mid + 1
18        else:
19            right = mid - 1
20    return -1
21
22 records = list(map(int, input("Enter roll numbers: ").split()))
23 target = int(input("Enter roll number to search: "))
24
25 lin_result = linear_search(records, target)
26 print("Linear Search:", "Found at index" if lin_result != -1 else "Not found", lin_result)
27
28 sorted_records = sorted(records)
29 bin_result = binary_search(sorted_records, target)
30 print("Binary Search (on sorted data):", "Found at index" if bin_result != -1 else "Not found", bin_result)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Chinnari\Downloads> & 'C:\Users\chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\chinnari\vscode\extensions\vs-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55845' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter roll numbers: 105 101 109 103 107
Enter roll number to search: 103
Linear Search: Found at index 3
Binary Search (on sorted data): Found at index 1
PS C:\Users\Chinnari\Downloads>

Ln 30, Col 108 Spaces: 4 UTF-8 CRLF () Python Python 3.13
```

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 give prompt and code for this and remove comments

```

1 #Complete recursive Quick Sort and Merge Sort functions in Python with docstrings. Explain recursion in both algorithms,
2 #test them on random, sorted, and reverse-sorted data, and compare time complexities and practical use cases.
3 def quick_sort(arr):
4     if len(arr) <= 1:
5         return arr
6     pivot = arr[len(arr) // 2]
7     left = [x for x in arr if x < pivot]
8     middle = [x for x in arr if x == pivot]
9     right = [x for x in arr if x > pivot]
10    return quick_sort(left) + middle + quick_sort(right)
11
12 def merge_sort(arr):
13     if len(arr) <= 1:
14         return arr
15     mid = len(arr) // 2
16     left = merge_sort(arr[:mid])
17     right = merge_sort(arr[mid:])
18     return merge(left, right)
19 def merge(left, right):
20     result = []
21     i = j = 0
22     while i < len(left) and j < len(right):
23         if left[i] < right[j]:
24             result.append(left[i])
25             i += 1
26         else:
27             result.append(right[j])
28             j += 1
29     result.extend(left[i:])
30     result.extend(right[j:])
31     return result
32
33
34 data = list(map(int, input("Enter numbers: ").split()))
35
36 print("Quick Sort on Random Data:", quick_sort(data))
37 print("Merge Sort on Random Data:", merge_sort(data))
38
39 sorted_data = sorted(data)
40 reverse_data = sorted(data, reverse=True)
41
42 print("Quick Sort on Sorted Data:", quick_sort(sorted_data))
43 print("Merge Sort on Sorted Data:", merge_sort(sorted_data))
44
45 print("Quick Sort on Reverse Data:", quick_sort(reverse_data))
46 print("Merge Sort on Reverse Data:", merge_sort(reverse_data))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Enter numbers: 8 3 1 7 0 10 2

Quick Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]

PS C:\Users\Chinnari\Downloads> []

```

19 def merge(left, right):
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
30
31
32
33
34 data = list(map(int, input("Enter numbers: ").split()))
35
36 print("Quick Sort on Random Data:", quick_sort(data))
37 print("Merge Sort on Random Data:", merge_sort(data))
38
39 sorted_data = sorted(data)
40 reverse_data = sorted(data, reverse=True)
41
42 print("Quick Sort on Sorted Data:", quick_sort(sorted_data))
43 print("Merge Sort on Sorted Data:", merge_sort(sorted_data))
44
45 print("Quick Sort on Reverse Data:", quick_sort(reverse_data))
46 print("Merge Sort on Reverse Data:", merge_sort(reverse_data))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Enter numbers: 8 3 1 7 0 10 2

Quick Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Random Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Sorted Data: [0, 1, 2, 3, 7, 8, 10]
Quick Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
Merge Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]

PS C:\Users\Chinnari\Downloads> []

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:
 - 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 give prompt and code for this and remove comments

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

- File Explorer:** Shows a file named "new.py".
- Code Editor:** Displays Python code for implementing duplicate detection. It includes two functions: `brute_force_duplicates` (brute-force approach) and `optimized_duplicates` (using sets). The code also reads user input for IDs and prints the results.
- Terminal:** Shows the command-line output of running the script. It merges sort data from a file, enters user IDs, and prints the results for both brute-force and optimized approaches.
- Sidebar:** Includes sections for RUN, BREAKPOINTS, and a Python configuration dropdown.

```

new.py
C:\> Users > Chinnari > Downloads > new.py > ...
1 #Implement duplicate detection in Python using a brute-force nested loop and an optimized approach with a set.
2 #Compare time complexity and explain why the optimized version performs better for large datasets with sample input/output.
3 def brute_force_duplicates(data):
4     duplicates = []
5     n = len(data)
6     for i in range(n):
7         for j in range(i + 1, n):
8             if data[i] == data[j] and data[i] not in duplicates:
9                 duplicates.append(data[i])
10    return duplicates
11
12
13 def optimized_duplicates(data):
14     seen = set()
15     duplicates = set()
16     for item in data:
17         if item in seen:
18             duplicates.add(item)
19         else:
20             seen.add(item)
21     return list(duplicates)
22
23
24 data = list(map(int, input("Enter user IDs: ").split()))
25
26 print("Brute Force Duplicates:", brute_force_duplicates(data))
27 print("Optimized Duplicates:", optimized_duplicates(data))

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
Merge Sort on Reverse Data: [0, 1, 2, 3, 7, 8, 10]
PS C:\Users\Chinnari\Downloads> cd 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe'
'C:\Users\Chinnari\vscode\extensions\ms-python.debugger-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60117' '--' 'C:\Users\Chinnari\Downloads\new.py'
Enter user IDs: 101 205 101 309 205 410
Brute Force Duplicates: [101, 205]
Optimized Duplicates: [205, 101]
PS C:\Users\Chinnari\Downloads>

```