

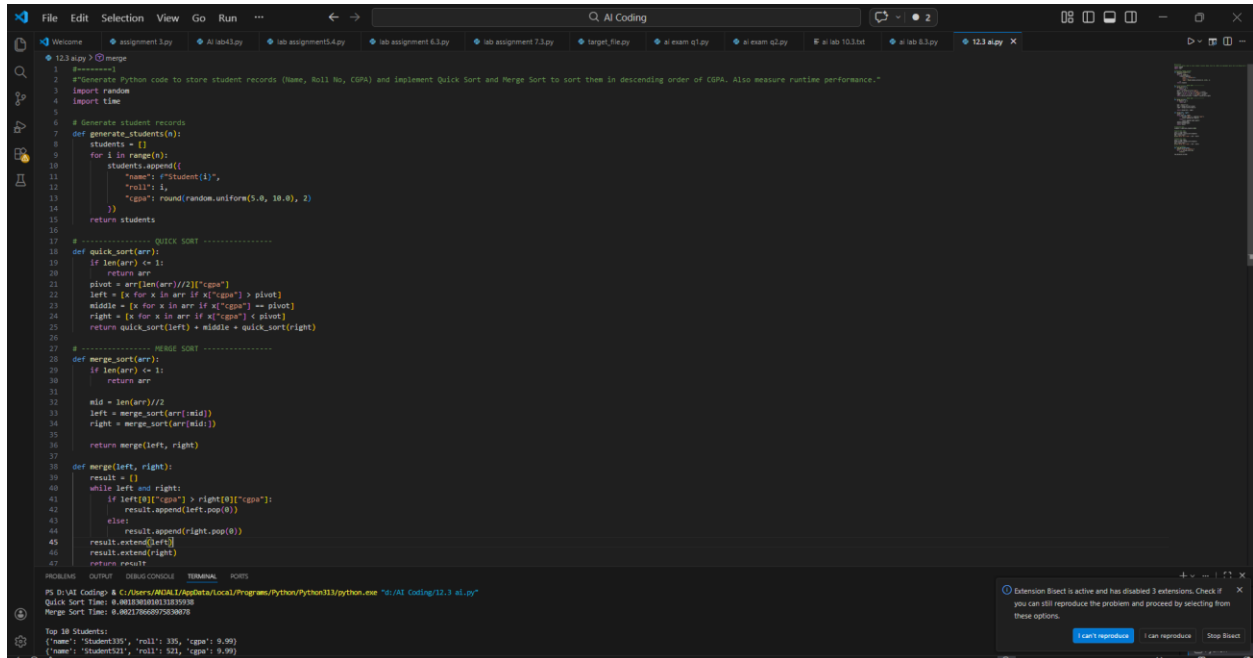
2303A51739

Lab assignment-12.3

Task 1: Sorting Student Records for Placement Drive

Scenario

SR University's Training and Placement Cell needs to shortlist candidates efficiently during campus placements. Student records must be sorted by CGPA in descending order.



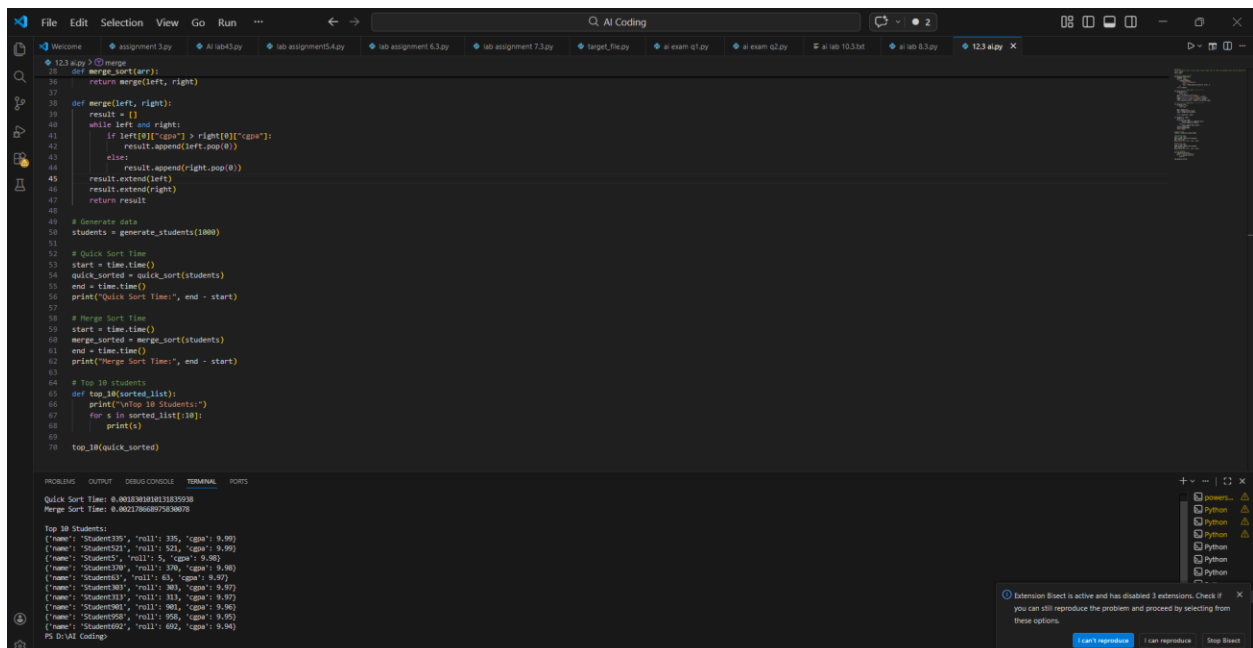
```
1 #merge
2 """Generate Python code to store student records (Name, Roll No, CGPA) and implement Quick Sort and Merge Sort to sort them in descending order of CGPA. Also measure runtime performance."""
3 import random
4 import time
5
6 # Generate student records
7 def generate_students(n):
8     students = []
9     for i in range(n):
10         student.append({
11             "name": f"Student{i+1}",
12             "roll": i,
13             "cgpa": round(random.uniform(5.0, 10.0), 2)
14         })
15     return students
16
17 # ----- QUICK SORT -----
18 def quick_sort(arr):
19     if len(arr) <= 1:
20         return arr
21     pivot = arr[len(arr)//2]["cgpa"]
22     left = [x for x in arr if x["cgpa"] > pivot]
23     middle = [x for x in arr if x["cgpa"] == pivot]
24     right = [x for x in arr if x["cgpa"] < pivot]
25     return quick_sort(left) + middle + quick_sort(right)
26
27 # ----- MERGE SORT -----
28 def merge_sort(arr):
29     if len(arr) <= 1:
30         return arr
31
32     mid = len(arr)//2
33     left = merge_sort(arr[:mid])
34     right = merge_sort(arr[mid:])
35
36     return merge(left, right)
37
38 def merge(left, right):
39     result = []
40     while left and right:
41         if left[0]["cgpa"] > right[0]["cgpa"]:
42             result.append(left.pop(0))
43         else:
44             result.append(right.pop(0))
45     result.extend(left)
46     result.extend(right)
47     return result
48
49 # Generate data
50 students = generate_students(1000)
51
52 # Quick Sort Time
53 start = time.time()
54 quick_sorted = quick_sort(students)
55 end = time.time()
56 print("Quick Sort Time:", end - start)
57
58 # Merge Sort Time
59 start = time.time()
60 merge_sorted = merge_sort(students)
61 end = time.time()
62 print("Merge Sort Time:", end - start)
63
64 # Top 10 students
65 def top_10(sorted_list):
66     print("Top 10 Students:")
67     for i in sorted_list[:10]:
68         print(i)
69
70 top_10(quick_sorted)
```

PS D:\AI Coding > C:\Users\NADAL\AppData\Local\Programs\Python\Python311\python.exe "D:/AI Coding/12.3 ai.py"

Quick Sort Time: 0.0018108118135908
Merge Sort Time: 0.0021766897583078

Top 10 Students:

```
{'name': 'Student100', 'roll': 335, 'cgpa': 9.99}
{'name': 'Student101', 'roll': 521, 'cgpa': 9.99}
{'name': 'Student5', 'roll': 5, 'cgpa': 9.96}
{'name': 'Student130', 'roll': 370, 'cgpa': 9.88}
{'name': 'Student64', 'roll': 64, 'cgpa': 9.97}
{'name': 'Student388', 'roll': 388, 'cgpa': 9.97}
{'name': 'Student131', 'roll': 33, 'cgpa': 9.97}
{'name': 'Student961', 'roll': 181, 'cgpa': 9.96}
{'name': 'Student958', 'roll': 958, 'cgpa': 9.95}
{'name': 'Student62', 'roll': 62, 'cgpa': 9.94}
```



```
1 #merge
2 """Generate Python code to store student records (Name, Roll No, CGPA) and implement Quick Sort and Merge Sort to sort them in descending order of CGPA. Also measure runtime performance."""
3 import random
4 import time
5
6 # Generate student records
7 def generate_students(n):
8     students = []
9     for i in range(n):
10         student.append({
11             "name": f"Student{i+1}",
12             "roll": i,
13             "cgpa": round(random.uniform(5.0, 10.0), 2)
14         })
15     return students
16
17 # ----- QUICK SORT -----
18 def quick_sort(arr):
19     if len(arr) <= 1:
20         return arr
21     pivot = arr[len(arr)//2]["cgpa"]
22     left = [x for x in arr if x["cgpa"] > pivot]
23     middle = [x for x in arr if x["cgpa"] == pivot]
24     right = [x for x in arr if x["cgpa"] < pivot]
25     return quick_sort(left) + middle + quick_sort(right)
26
27 # ----- MERGE SORT -----
28 def merge_sort(arr):
29     if len(arr) <= 1:
30         return arr
31
32     mid = len(arr)//2
33     left = merge_sort(arr[:mid])
34     right = merge_sort(arr[mid:])
35
36     return merge(left, right)
37
38 def merge(left, right):
39     result = []
40     while left and right:
41         if left[0]["cgpa"] > right[0]["cgpa"]:
42             result.append(left.pop(0))
43         else:
44             result.append(right.pop(0))
45     result.extend(left)
46     result.extend(right)
47     return result
48
49 # Generate data
50 students = generate_students(1000)
51
52 # Quick Sort Time
53 start = time.time()
54 quick_sorted = quick_sort(students)
55 end = time.time()
56 print("Quick Sort Time:", end - start)
57
58 # Merge Sort Time
59 start = time.time()
60 merge_sorted = merge_sort(students)
61 end = time.time()
62 print("Merge Sort Time:", end - start)
63
64 # Top 10 students
65 def top_10(sorted_list):
66     print("Top 10 Students:")
67     for i in sorted_list[:10]:
68         print(i)
69
70 top_10(quick_sorted)
```

PS D:\AI Coding > C:\Users\NADAL\AppData\Local\Programs\Python\Python311\python.exe "D:/AI Coding/12.3 ai.py"

Quick Sort Time: 0.0018108118135908
Merge Sort Time: 0.0021766897583078

Top 10 Students:

```
{'name': 'Student100', 'roll': 335, 'cgpa': 9.99}
{'name': 'Student101', 'roll': 521, 'cgpa': 9.99}
{'name': 'Student5', 'roll': 5, 'cgpa': 9.96}
{'name': 'Student130', 'roll': 370, 'cgpa': 9.88}
{'name': 'Student64', 'roll': 64, 'cgpa': 9.97}
{'name': 'Student388', 'roll': 388, 'cgpa': 9.97}
{'name': 'Student131', 'roll': 33, 'cgpa': 9.97}
{'name': 'Student961', 'roll': 181, 'cgpa': 9.96}
{'name': 'Student958', 'roll': 958, 'cgpa': 9.95}
{'name': 'Student62', 'roll': 62, 'cgpa': 9.94}
```

Task 2: Implementing Bubble Sort with AI Comments

- Task: Write a Python implementation of Bubble Sort

```
1 #####
2 #Add inline comments explaining Bubble Sort logic and provide time complexity.
3 def bubble_sort(arr):
4     n = len(arr)
5
6     # Traverse through all elements
7     for i in range(n):
8
9         # Flag to detect if swap happens
10        swapped = False
11
12        # Last i elements are already sorted
13        for j in range(0, n-i-1):
14
15            # Compare adjacent elements
16            if arr[j] > arr[j+1]:
17
18                # Swap if elements are in wrong order
19                arr[j], arr[j+1] = arr[j+1], arr[j]
20                swapped = True
21
22        # If no swapping happened, array is sorted
23        if not swapped:
24            break
25
26    return arr
27
28 data = [5, 3, 6, 4, 2]
29 print("Sorted:", bubble_sort(data))
```

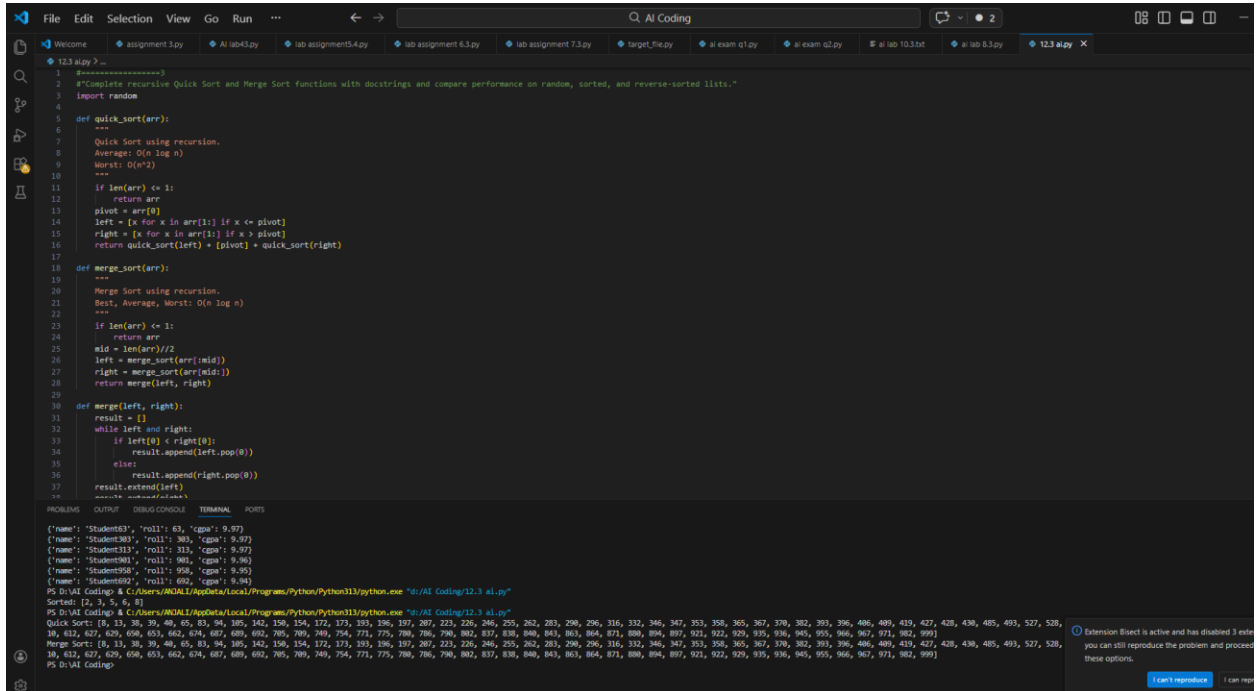
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
{'name': 'Student1', 'roll': 63, 'gpa': 9.97}
{'name': 'Student2', 'roll': 84, 'gpa': 9.97}
{'name': 'Student3', 'roll': 33, 'gpa': 9.97}
{'name': 'Student4', 'roll': 98, 'gpa': 9.96}
{'name': 'Student5', 'roll': 68, 'gpa': 9.96}
{'name': 'Student6', 'roll': 69, 'gpa': 9.94}
PS D:\AI Coding & C\Users\NDAI\AppData\Local\Programs\Python\Python113\python.exe "D:/AI Coding/12.3 ai.py"
Sorted: [2, 3, 5, 6, 9]
PS D:\AI Coding\
```

Extension Bisect is active and has disabled 3 extensions. Check if you can still reproduce the problem and proceed by selecting from Best options.

Task 3: Quick Sort and Merge Sort Comparison

• Task: Implement Quick Sort and Merge Sort using recursion

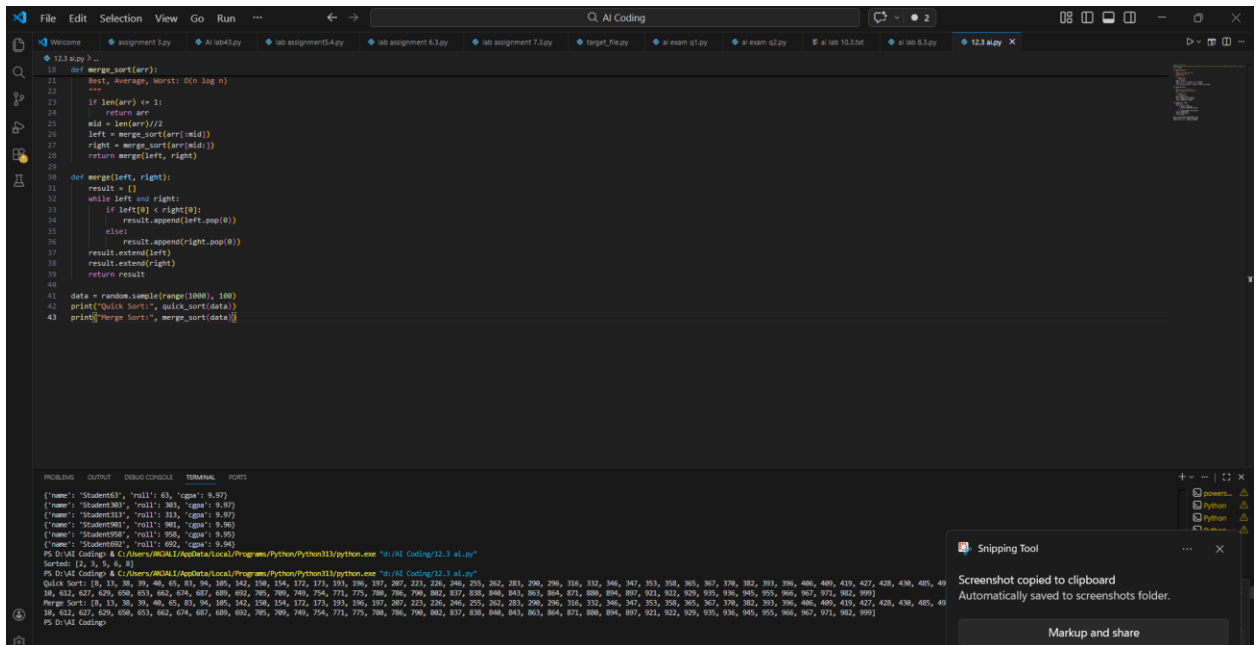


```
1 #=====
2 #Complete recursive Quick Sort and Merge Sort Functions with docstrings and compare performance on random, sorted, and reverse-sorted lists.
3 import random
4
5 def quick_sort(arr):
6     """
7     Quick Sort using recursion.
8     Average: O(n log n)
9     Worst: O(n^2)
10    """
11    if len(arr) <= 1:
12        return arr
13    pivot = arr[0]
14    left = [x for x in arr[1:] if x <= pivot]
15    right = [x for x in arr[1:] if x > pivot]
16    return quick_sort(left) + [pivot] + quick_sort(right)
17
18 def merge_sort(arr):
19     """
20     Merge Sort using recursion.
21     Best, Average, Worst: O(n log n)
22    """
23    if len(arr) <= 1:
24        return arr
25    mid = len(arr)//2
26    left = merge_sort(arr[:mid])
27    right = merge_sort(arr[mid:])
28    return merge(left, right)
29
30 def merge(left, right):
31     result = []
32     while left and right:
33         if left[0] < right[0]:
34             result.append(left.pop(0))
35         else:
36             result.append(right.pop(0))
37     result.extend(left)
38     result.extend(right)
39     return result
40
41 data = random.sample(range(1000), 1000)
42 print("Quick Sort:", quick_sort(data))
43 print("Merge Sort:", merge_sort(data))
```

Sorted: [2, 3, 5, 6, 8]

Quick Sort: [8, 13, 38, 39, 40, 65, 83, 94, 105, 142, 150, 154, 172, 173, 183, 196, 197, 207, 223, 226, 246, 255, 262, 283, 289, 296, 316, 332, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 438, 439, 485, 493, 527, 528, 562, 627, 629, 650, 653, 662, 674, 687, 689, 692, 705, 709, 740, 754, 771, 775, 780, 786, 790, 802, 837, 838, 840, 843, 863, 864, 871, 880, 894, 897, 921, 922, 929, 935, 936, 945, 955, 966, 967, 971, 982, 999]

Merge Sort: [8, 13, 38, 39, 40, 65, 83, 94, 105, 142, 150, 154, 172, 173, 183, 196, 197, 207, 223, 226, 246, 255, 262, 283, 289, 296, 316, 332, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 438, 439, 485, 493, 527, 528, 562, 627, 629, 650, 653, 662, 674, 687, 689, 692, 705, 709, 740, 754, 771, 775, 780, 786, 790, 802, 837, 838, 840, 843, 863, 864, 871, 880, 894, 897, 921, 922, 929, 935, 936, 945, 955, 966, 967, 971, 982, 999]



```
18 def merge_sort(arr):
19     """
20     Merge Sort using recursion.
21     Best, Average, Worst: O(n log n)
22    """
23    if len(arr) <= 1:
24        return arr
25    mid = len(arr)//2
26    left = merge_sort(arr[:mid])
27    right = merge_sort(arr[mid:])
28    return merge(left, right)
29
30 def merge(left, right):
31     result = []
32     while left and right:
33         if left[0] < right[0]:
34             result.append(left.pop(0))
35         else:
36             result.append(right.pop(0))
37     result.extend(left)
38     result.extend(right)
39     return result
40
41 data = random.sample(range(1000), 1000)
42 print("Quick Sort:", quick_sort(data))
43 print("Merge Sort:", merge_sort(data))
```

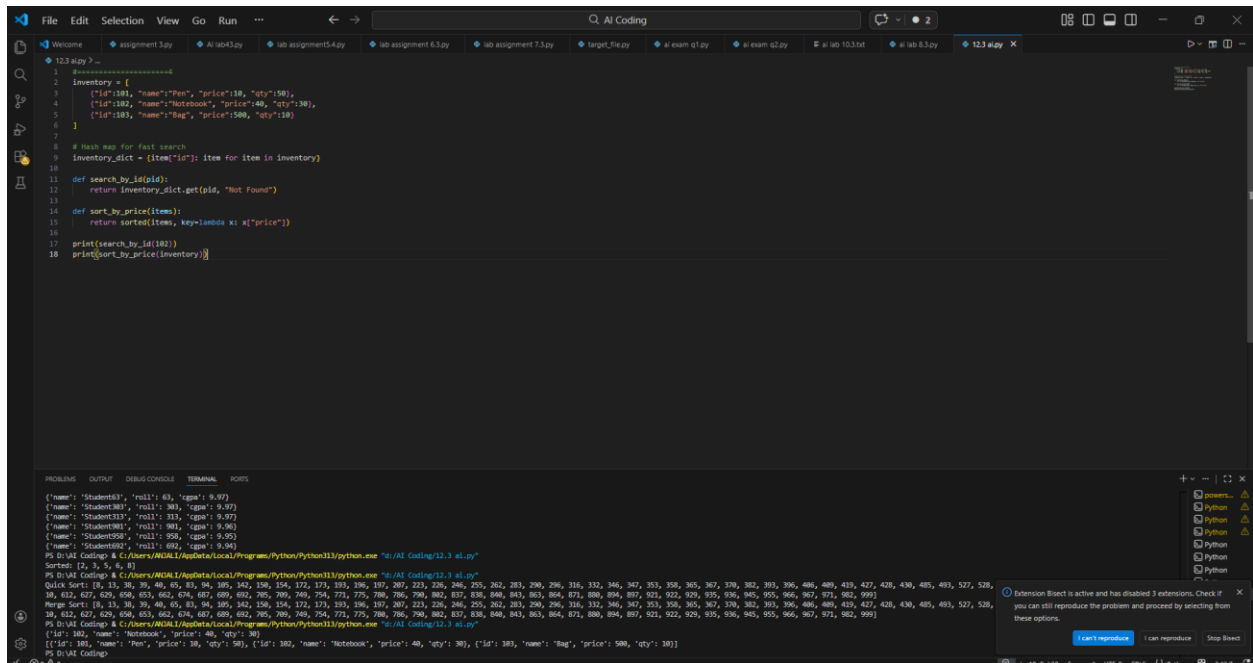
Sorted: [2, 3, 5, 6, 8]

Quick Sort: [8, 13, 38, 39, 40, 65, 83, 94, 105, 142, 150, 154, 172, 173, 183, 196, 197, 207, 223, 226, 246, 255, 262, 283, 289, 296, 316, 332, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 438, 439, 485, 493, 527, 528, 562, 627, 629, 650, 653, 662, 674, 687, 689, 692, 705, 709, 740, 754, 771, 775, 780, 786, 790, 802, 837, 838, 840, 843, 863, 864, 871, 880, 894, 897, 921, 922, 929, 935, 936, 945, 955, 966, 967, 971, 982, 999]

Merge Sort: [8, 13, 38, 39, 40, 65, 83, 94, 105, 142, 150, 154, 172, 173, 183, 196, 197, 207, 223, 226, 246, 255, 262, 283, 289, 296, 316, 332, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 438, 439, 485, 493, 527, 528, 562, 627, 629, 650, 653, 662, 674, 687, 689, 692, 705, 709, 740, 754, 771, 775, 780, 786, 790, 802, 837, 838, 840, 843, 863, 864, 871, 880, 894, 897, 921, 922, 929, 935, 936, 945, 955, 966, 967, 971, 982, 999]

Task 4 (Real-Time Application – Inventory Management System)

Scenario: A retail store's inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff



```
File Edit Selection View Go Run ... Q AI Coding
12.3 alpy 2
1 # =====
2 inventory = [
3     {'id': 1001, 'name': 'Pen', 'price': 10, 'qty': 100},
4     {'id': 1002, 'name': 'Notebook', 'price': 40, 'qty': 10},
5     {'id': 1003, 'name': 'Bag', 'price': 500, 'qty': 10}
6 ]
7
8 # Hash map for fast search
9 inventory_dict = {item['id']: item for item in inventory}
10
11 def search_by_id(pid):
12     return inventory_dict.get(pid, "Not Found")
13
14 def sort_by_price(items):
15     return sorted(items, key=lambda x: x['price'])
16
17 print(search_by_id(1002))
18 print(sort_by_price(inventory))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
{'name': 'Student001', 'roll': 01, 'gpa': 9.97}
{'name': 'Student001', 'roll': 01, 'gpa': 9.97}
{'name': 'Student001', 'roll': 01, 'gpa': 9.97}
{'name': 'Student001', 'roll': 01, 'gpa': 9.97}
{'name': 'Student001', 'roll': 01, 'gpa': 9.97}
{'name': 'Student001', 'roll': 01, 'gpa': 9.97}
PS D:\AI Coding > C:\Users\ANAL\AppData\Local\Programs\Python\Python113\python.exe "d:/AI Coding/12.3 al.py"
Sorted: [2, 3, 5, 6, 8]
PS D:\AI Coding > C:\Users\ANAL\AppData\Local\Programs\Python\Python113\python.exe "d:/AI Coding/12.3 al.py"
Quick Sort: [0, 15, 38, 39, 40, 60, 63, 96, 100, 145, 150, 154, 172, 173, 183, 186, 187, 207, 223, 226, 246, 255, 262, 283, 286, 286, 316, 332, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 410, 427, 428, 430, 436, 485, 483, 527, 528,
10, 612, 627, 629, 650, 653, 662, 674, 687, 689, 692, 705, 709, 740, 754, 773, 775, 780, 786, 790, 802, 837, 838, 840, 843, 863, 864, 871, 880, 894, 897, 921, 922, 929, 935, 936, 945, 955, 966, 967, 971, 982, 999]
Range Sort: [0, 15, 38, 39, 40, 60, 63, 96, 100, 145, 150, 154, 172, 173, 183, 186, 187, 207, 223, 226, 246, 255, 262, 283, 286, 286, 316, 332, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 410, 427, 428, 430, 436, 485, 483, 527, 528,
10, 612, 627, 629, 650, 653, 662, 674, 687, 689, 692, 705, 709, 740, 754, 773, 775, 780, 786, 790, 802, 837, 838, 840, 843, 863, 864, 871, 880, 894, 897, 921, 922, 929, 935, 936, 945, 955, 966, 967, 971, 982, 999]
PS D:\AI Coding > C:\Users\ANAL\AppData\Local\Programs\Python\Python113\python.exe "d:/AI Coding/12.3 al.py"
[{'id': 1002, 'name': 'Notebook', 'price': 40, 'qty': 10}
[{'id': 1002, 'name': 'Pen', 'price': 10, 'qty': 100}, {'id': 1002, 'name': 'Notebook', 'price': 40, 'qty': 10}, {'id': 1003, 'name': 'Bag', 'price': 500, 'qty': 10}]
PS D:\AI Coding >
```

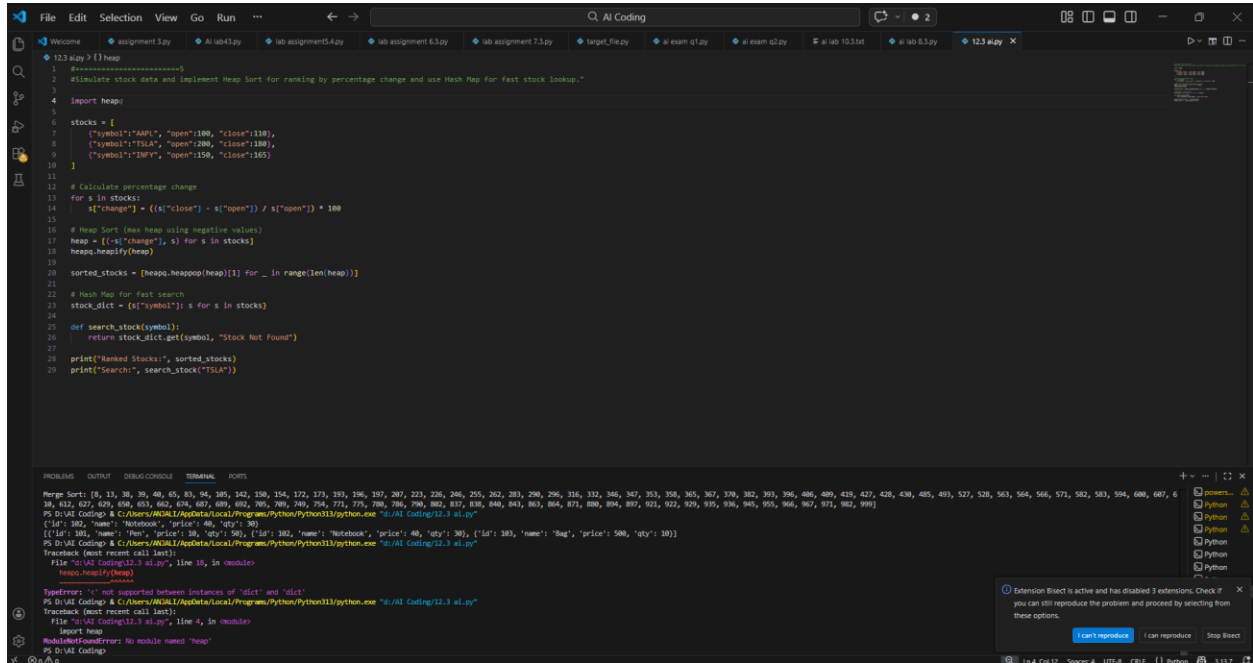
Extension Bisect is active and has disabled 3 extensions. Check if you can still reproduce the problem and proceed by selecting from these options.

[Can't reproduce] [Can reproduce] [Stop Bisect]

Task 5: Real-Time Stock Data Sorting & Searching

Scenario:

An AI-powered FinTech Lab at SR University is building a tool for analyzing stock price movements. The requirement is to quickly sort stocks by daily gain/loss and search for specific stock symbols efficiently.



```
1 #!/usr/bin/env python3
2 # Simulate stock data and implement Heap Sort for ranking by percentage change and use Hash Map for fast stock lookup.
3
4 import heapq
5
6 stocks = [
7     {"symbol": "AAPL", "open": 180, "close": 185},
8     {"symbol": "TSLA", "open": 200, "close": 190},
9     {"symbol": "INFY", "open": 150, "close": 165}
10 ]
11
12 # Calculate percentage change
13 for s in stocks:
14     s["change"] = ((s["close"] - s["open"]) / s["open"]) * 100
15
16 # Heap Sort (use neg change using negative values)
17 heap = [(-s["change"], s) for s in stocks]
18 heapq.heapify(heap)
19
20 sorted_stocks = [heapq.heappop(heap)[1] for _ in range(len(heap))]
21
22 # Hash Map for fast search
23 stock_dict = {s["symbol"]: s for s in stocks}
24
25 def search_stock(symbol):
26     return stock_dict.get(symbol, "Stock Not Found")
27
28 print("Sorted Stocks:", sorted_stocks)
29 print("Search:", search_stock("TSLA"))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Merge Sort: [8, 13, 38, 39, 40, 65, 83, 94, 105, 142, 150, 154, 172, 173, 193, 196, 197, 207, 223, 226, 246, 255, 262, 283, 290, 296, 316, 332, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 428, 430, 485, 493, 527, 528, 563, 564, 566, 571, 582, 583, 594, 600, 607, 610, 622, 627, 629, 630, 633, 662, 674, 687, 689, 692, 705, 709, 740, 754, 772, 775, 780, 786, 790, 802, 837, 838, 840, 843, 863, 864, 871, 880, 894, 897, 921, 922, 925, 935, 936, 945, 955, 966, 967, 971, 982, 990]

PS D:\AI Coding & C:\Users\NAGALI\AppData\Local\Programs\Python\Python311\python.exe "D:/AI Coding/12.3 ai.py"

{'id': 102, 'name': 'Netflix', 'price': 40, 'qty': 30}

[{'id': 100, 'name': 'Tesla', 'price': 190, 'qty': 50}, {'id': 102, 'name': 'Netflix', 'price': 40, 'qty': 30}, {'id': 103, 'name': 'Bag', 'price': 500, 'qty': 10}]

Traceback (most recent call last):

File "D:\AI Coding\12.3 ai.py", line 10, in <module>

import heapq

ModuleNotFoundError: No module named 'heap'

PS D:\AI Coding >