

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.

The screenshot shows the VS Code interface with the file `12.3 ai.py` open. The code implements two sorting algorithms: Quick Sort and Merge Sort, and a function to print the top 10 students from a list of generated student records.

```
File Edit Selection View Go Run ... ← → Q AI Coding
1 Welcome 2 assignment1.py 3 lab4.py 4 lab assignment5.4.py 5 lab assignment 6.3.py 6 lab assignment 7.3.py 7 target_file.py 8 ai exam 9.py 9 ai exam 10.py 10 ai lab 10.3.txt 11 ai lab 8.3.py 12 12.3 ai.py 13
1 # 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.
2
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 = {
11             "name": f"Student{i+1}",
12             "roll": i,
13             "cgpa": round(random.uniform(5.0, 10.0), 2)
14         }
15         students.append(student)
16     return students
17
18 # ..... QUICK SORT .....
19 def quick_sort(arr):
20     if len(arr) <= 1:
21         return arr
22     pivot = arr[len(arr)//2]["cgpa"]
23     left = [x for x in arr if x["cgpa"] >= pivot]
24     middle = [x for x in arr if x["cgpa"] == pivot]
25     right = [x for x in arr if x["cgpa"] < pivot]
26     return quick_sort(left) + middle + quick_sort(right)
27
28 # ..... MERGE SORT .....
29 def merge_sort(arr):
30     if len(arr) <= 1:
31         return arr
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
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\AI\AI Coding & C:\Users\ACER\AppData\Local\Programs\Python\Python311\python.exe "D:\AI Coding\12.3 ai.py"
PS D:\AI\AI Coding & C:\Users\ACER\AppData\Local\Programs\Python\Python311\python.exe "D:\AI Coding\12.3 ai.py"
Merge Sort Time: 0.002178668975830078
Top 10 Students:
('name': 'Student335', 'roll': 335, 'cgpa': 9.99)
('name': 'Student521', 'roll': 521, 'cgpa': 9.99)
('name': 'Student330', 'roll': 330, 'cgpa': 9.99)
('name': 'Student320', 'roll': 320, 'cgpa': 9.98)
('name': 'Student63', 'roll': 63, 'cgpa': 9.97)
('name': 'Student303', 'roll': 303, 'cgpa': 9.97)
('name': 'Student313', 'roll': 313, 'cgpa': 9.97)
('name': 'Student981', 'roll': 981, 'cgpa': 9.96)
('name': 'Student958', 'roll': 958, 'cgpa': 9.95)
('name': 'Student692', 'roll': 692, 'cgpa': 9.96)
PS D:\AI\AI Coding
```

The screenshot shows the VS Code interface with the file `12.3 ai.py` open. The code implements two sorting algorithms: Quick Sort and Merge Sort, and a function to print the top 10 students from a list of generated student records. This version includes timing measurements for both sorting methods.

```
File Edit Selection View Go Run ... ← → Q AI Coding
1 Welcome 2 assignment1.py 3 lab4.py 4 lab assignment5.4.py 5 lab assignment 6.3.py 6 lab assignment 7.3.py 7 target_file.py 8 ai exam 9.py 9 ai exam 10.py 10 ai lab 10.3.txt 11 ai lab 8.3.py 12 12.3 ai.py 13
1 # 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.
2
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 = {
11             "name": f"Student{i+1}",
12             "roll": i,
13             "cgpa": round(random.uniform(5.0, 10.0), 2)
14         }
15         students.append(student)
16     return students
17
18 # Quick Sort Time
19 start = time.time()
20 quick_sorted = quick_sort(students)
21 end = time.time()
22 print("Quick Sort Time:", end - start)
23
24 # Merge Sort Time
25 start = time.time()
26 merge_sorted = merge_sort(students)
27 end = time.time()
28 print("Merge Sort Time:", end - start)
29
30 # Top 10 students
31 def top_10(sorted_list):
32     print("Top 10 Students:")
33     for i in sorted_list[:10]:
34         print(i)
35
36 top_10(quick_sorted)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Quick Sort Time: 0.0018004913839593
Merge Sort Time: 0.002178668975830078
Top 10 Students:
('name': 'Student335', 'roll': 335, 'cgpa': 9.99)
('name': 'Student521', 'roll': 521, 'cgpa': 9.99)
('name': 'Student330', 'roll': 330, 'cgpa': 9.99)
('name': 'Student320', 'roll': 320, 'cgpa': 9.98)
('name': 'Student63', 'roll': 63, 'cgpa': 9.97)
('name': 'Student303', 'roll': 303, 'cgpa': 9.97)
('name': 'Student313', 'roll': 313, 'cgpa': 9.97)
('name': 'Student981', 'roll': 981, 'cgpa': 9.96)
('name': 'Student958', 'roll': 958, 'cgpa': 9.95)
('name': 'Student692', 'roll': 692, 'cgpa': 9.96)
PS D:\AI\AI Coding
```

Task 2: Implementing Bubble Sort with AI Comments

- Task: Write a Python implementation of Bubble Sort

The screenshot shows a code editor window titled "AI Coding". The file being edited is "12.3 ai.py". The code implements the bubble sort algorithm with detailed inline comments explaining the logic and time complexity.

```
1 #=====2 1 #Add inline comments explaining Bubble Sort logic and provide time complexity.3 2 def bubble_sort(arr):4 3     n = len(arr)5 6     # Traverse through all elements6     for i in range(n):7 8         # Flag to detect if swap happens8         swapped = False9 10        # Last i elements are already sorted10        for j in range(0, n-i-1):11 12            # Compare adjacent elements12            if arr[i] > arr[i+1]:13 14                # Swap if elements are in wrong order14                arr[i], arr[i+1] = arr[i+1], arr[i]15 16                swapped = True17 18            # If no swapping happened, array is sorted19 20            if not swapped:21 22                break23 24        return arr25 26    data = [5, 3, 8, 6, 2]27 28    print("Sorted!", bubble_sort(data))
```

The terminal tab shows the output of running the script:

```
('name': 'Student03', 'roll': 63, 'cgpa': 9.87)
('name': 'Student301', 'roll': 30, 'cgpa': 9.97)
('name': 'Student333', 'roll': 33, 'cgpa': 9.97)
('name': 'Student091', 'roll': 91, 'cgpa': 9.98)
('name': 'Student092', 'roll': 92, 'cgpa': 9.95)
('name': 'Student093', 'roll': 69, 'cgpa': 9.94)
PS D:\AI Coding & C:\Users\ADALI\AppData\Local\Programs\Python\Python311\python.exe "d:/AI Coding/12.3 ai.py"
Sorted! [2, 3, 5, 6, 8]
PS D:\AI Coding
```

A status bar at the bottom right indicates: "Extension Biscet is active and has disabled 3 extensions. Check if you can still reproduce the problem and proceed by selecting from the dropdown menu." There is also a small icon for "Python" in the status bar.

Task 3: Quick Sort and Merge Sort Comparison

- Task: Implement Quick Sort and Merge Sort using recursion

```

1 #Complete recursive Quick Sort and Merge Sort Functions with docstrings and compare performance on random, sorted, and reverse-sorted lists.
2
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
40 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
41
42 ('name': 'Student63', 'roll': 63, 'cgpa': 9.97)
43 ('name': 'Student303', 'roll': 303, 'cgpa': 9.97)
44 ('name': 'Student133', 'roll': 133, 'cgpa': 9.97)
45 ('name': 'Student901', 'roll': 901, 'cgpa': 9.96)
46 ('name': 'Student958', 'roll': 958, 'cgpa': 9.95)
47 ('name': 'Student627', 'roll': 627, 'cgpa': 9.95)
48
49 PS D:\AI Coding & C:\Users\HANLI\AppData\Local\Programs\Python\Python311\python.exe "d:/AI Coding/12.3 ai.py"
50 Sorted: [2, 3, 5, 6, 8]
51 PS D:\AI Coding & C:\Users\HANLI\AppData\Local\Programs\Python\Python311\python.exe "d:/AI Coding/12.3 ai.py"
52 Sorted: [8, 13, 38, 49, 65, 83, 94, 105, 142, 150, 154, 172, 173, 193, 196, 197, 207, 223, 226, 246, 255, 262, 282, 299, 306, 316, 322, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 428, 439, 485, 493, 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, 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, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599]
50 Merge Sort: [8, 13, 38, 49, 65, 83, 94, 105, 142, 150, 154, 172, 173, 193, 196, 197, 207, 223, 226, 246, 255, 262, 282, 299, 306, 316, 322, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 428, 439, 485, 493, 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, 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, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599]
51 PS D:\AI Coding

```

```

1 #Complete recursive Quick Sort and Merge Sort Functions with docstrings and compare performance on random, sorted, and reverse-sorted lists.
2
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    mid = len(arr)/2
14    left = quick_sort(arr[:mid])
15    right = quick_sort(arr[mid:])
16    return merge(left, right)
17
18 def merge(left, right):
19     result = []
20     while left and right:
21         if left[0] <= right[0]:
22             result.append(left.pop(0))
23         else:
24             result.append(right.pop(0))
25     result.extend(left)
26     result.extend(right)
27
28
29 def quick_sort(data):
30     print("Quick Sort:", quick_sort(data))
31
32 def merge_sort(data):
33     print("Merge Sort:", merge_sort(data))
34
35
36 data = random.sample(range(1000), 100)
37 print("Quick Sort:", quick_sort(data))
38 print("Merge Sort:", merge_sort(data))
39
40 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
41
42 ('name': 'Student63', 'roll': 63, 'cgpa': 9.97)
43 ('name': 'Student303', 'roll': 303, 'cgpa': 9.97)
44 ('name': 'Student133', 'roll': 133, 'cgpa': 9.97)
45 ('name': 'Student901', 'roll': 901, 'cgpa': 9.96)
46 ('name': 'Student958', 'roll': 958, 'cgpa': 9.95)
47 ('name': 'Student627', 'roll': 627, 'cgpa': 9.95)
48
49 PS D:\AI Coding & C:\Users\HANLI\AppData\Local\Programs\Python\Python311\python.exe "d:/AI Coding/12.3 ai.py"
50 Sorted: [2, 3, 5, 6, 8]
51 PS D:\AI Coding & C:\Users\HANLI\AppData\Local\Programs\Python\Python311\python.exe "d:/AI Coding/12.3 ai.py"
52 Sorted: [8, 13, 38, 49, 65, 83, 94, 105, 142, 150, 154, 172, 173, 193, 196, 197, 207, 223, 226, 246, 255, 262, 282, 299, 306, 316, 322, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 428, 439, 485, 493, 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, 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, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599]
50 Merge Sort: [8, 13, 38, 49, 65, 83, 94, 105, 142, 150, 154, 172, 173, 193, 196, 197, 207, 223, 226, 246, 255, 262, 282, 299, 306, 316, 322, 346, 347, 353, 358, 365, 367, 370, 382, 393, 396, 406, 409, 419, 427, 428, 439, 485, 493, 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, 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, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599]
51 PS D:\AI Coding

```

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

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.

The screenshot shows a Jupyter Notebook interface with the following code in cell 12.3 ai.py:

```
# simulate stock data and implement Heap Sort for ranking by percentage change and use Hash Map for fast stock lookup.
import heapq

stocks = [
    {"symbol": "AMZN", "open": 100, "close": 110},
    {"symbol": "TSLA", "open": 120, "close": 130},
    {"symbol": "INFL", "open": 150, "close": 165}
]

# calculate percentage change
for s in stocks:
    s["change"] = ((s["close"] - s["open"]) / s["open"]) * 100

# Heap Sort (max heap using negative values)
heap = [(-s["change"], s) for s in stocks]
 heapq.heapify(heap)

sorted_stocks = [heapq.heappop(heap)[1] for _ in range(len(heap))]

# Hash Map for fast search
stock_dict = {s["symbol"]: s for s in stocks}

def search_stock(symbol):
    return stock_dict.get(symbol, "Stock Not Found")

print("Ranked Stocks:", sorted_stocks)
print("Search:", search_stock("TSLA"))
```

The output of the code is displayed in the terminal tab:

```
Ranked Stocks: [{'symbol': 'INFL', 'open': 150, 'close': 165, 'change': 10}, {'symbol': 'AMZN', 'open': 100, 'close': 110, 'change': 10}, {'symbol': 'TSLA', 'open': 120, 'close': 130, 'change': 10}]
Search: TSLA
```

A tooltip message in the bottom right corner states: "Extension Block is active and has disabled 3 extensions. Check if you can still reproduce the problem and proceed by selecting from these options." It includes buttons for "Can reproduce", "Stop Block", and "Don't reproduce".