

ASSIGNMENT – 12.3

NAME:CH.Charan Kumar

Roll.no:2303A54006

BATCH:47-A

Lab 12: Algorithms with AI Assistance Sorting, Searching, and Algorithm

Optimization Using AI Tools

Lab Objectives

The objectives of this laboratory exercise are to:

Week3 -

Wednesday

- Apply AI-assisted programming techniques to implement sorting and searching algorithms.
- Analyze and compare algorithm efficiency using time and space complexity.
- Understand how AI tools can suggest optimizations and alternative algorithmic approaches.
- Strengthen problem-solving skills through real-world, data-driven scenarios.

Learning Outcomes

After completing this lab, students will be able to:

- Implement and optimize classic algorithms using AI-assisted coding tools.
- Compare multiple algorithms for the same problem and justify their selection.
- Measure and analyze runtime performance using experimental data.
- Critically review and refine AI-generated algorithmic solutions.

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.

Tasks

1. Use GitHub Copilot to generate a program that stores student records (Name, Roll Number, CGPA).
2. Implement the following sorting algorithms using AI assistance:

o Quick Sort

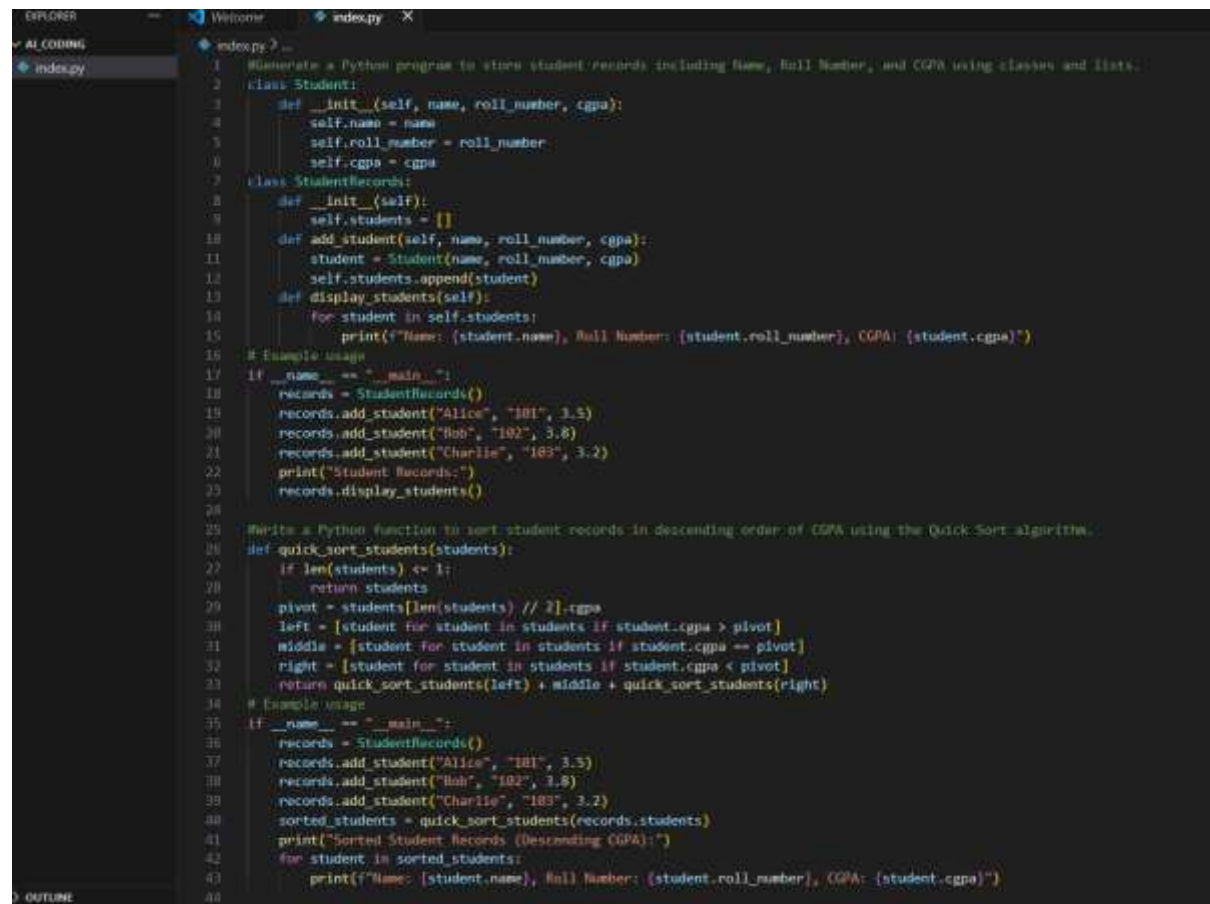
o Merge Sort

3. Measure and compare runtime performance for large datasets.

4. Write a function to display the top 10 students based on CGPA.

Expected Outcome

- Correctly sorted student records.
- Performance comparison between Quick Sort and Merge Sort.
- Clear output of top-performing students.



```
1 #Generate a Python program to store student records including Name, Roll Number, and CGPA using classes and lists.
2 class Student:
3     def __init__(self, name, roll_number, cgpa):
4         self.name = name
5         self.roll_number = roll_number
6         self.cgpa = cgpa
7 class StudentRecords:
8     def __init__(self):
9         self.students = []
10    def add_student(self, name, roll_number, cgpa):
11        student = Student(name, roll_number, cgpa)
12        self.students.append(student)
13    def display_students(self):
14        for student in self.students:
15            print(f'Name: {student.name}, Roll Number: {student.roll_number}, CGPA: {student.cgpa}')
16 # Example usage
17 if __name__ == "__main__":
18     records = StudentRecords()
19     records.add_student("Alice", "101", 3.5)
20     records.add_student("Bob", "102", 3.8)
21     records.add_student("Charlie", "103", 3.2)
22     print("Student Records:")
23     records.display_students()
24
25 #Write a Python function to sort student records in descending order of CGPA using the Quick Sort algorithm.
26 def quick_sort_students(students):
27     if len(students) <= 1:
28         return students
29     pivot = students[len(students) // 2].cgpa
30     left = [student for student in students if student.cgpa > pivot]
31     middle = [student for student in students if student.cgpa == pivot]
32     right = [student for student in students if student.cgpa < pivot]
33     return quick_sort_students(left) + middle + quick_sort_students(right)
34 # Example usage
35 if __name__ == "__main__":
36     records = StudentRecords()
37     records.add_student("Alice", "101", 3.5)
38     records.add_student("Bob", "102", 3.8)
39     records.add_student("Charlie", "103", 3.2)
40     sorted_students = quick_sort_students(records.students)
41     print("Sorted Student Records (Descending CGPA):")
42     for student in sorted_students:
43         print(f'Name: {student.name}, Roll Number: {student.roll_number}, CGPA: {student.cgpa}')
44
```

```

import time
# Performance comparison
if __name__ == "__main__":
    records = StudentRecords()
    records.add_student("Alice", "101", 3.5)
    records.add_student("Bob", "102", 3.8)
    records.add_student("Charlie", "103", 3.2)
    # Timing Quick Sort
    start_time = time.time()
    sorted_students_quick = quick_sort_students(records.students)
    end_time = time.time()
    print(f"Quick Sort Time: {end_time - start_time:.6f} seconds")
    # Timing Merge Sort
    start_time = time.time()
    sorted_students_merge = merge_sort_students(records.students)
    end_time = time.time()
    print(f"Merge Sort Time: {end_time - start_time:.6f} seconds")

#Generate Python code to measure and compare the execution time of Quick Sort and Merge Sort for large datasets.
import random
def generate_random_students(num_students):
    students = []
    for i in range(num_students):
        name = f"Student_{i}"
        roll_number = f"{100 + i}"
        cgpa = round(random.uniform(2.0, 4.0), 2)
        students.append(Student(name, roll_number, cgpa))
    return students
if __name__ == "__main__":
    num_students = 1000
    random_students = generate_random_students(num_students)
    # Timing Quick Sort
    start_time = time.time()
    sorted_students_quick = quick_sort_students(random_students.copy())
    end_time = time.time()
    print(f"Quick Sort Time for {num_students} students: {end_time - start_time:.6f} seconds")
    # Timing Merge Sort
    start_time = time.time()
    sorted_students_merge = merge_sort_students(random_students.copy())
    end_time = time.time()
    print(f"Merge Sort Time for {num_students} students: {end_time - start_time:.6f} seconds")

#Write a Python function to display the top 10 students based on CGPA from a sorted list.
def display_top_students(students, top_n=10):
    print(f"Top {top_n} Students based on CGPA:")
    for student in students[:top_n]:
        print(f"Name: {student.name}, Roll Number: {student.roll_number}, CGPA: {student.cgpa}")
# Example usage
if __name__ == "__main__":
    records = StudentRecords()
    records.add_student("Alice", "101", 3.5)
    records.add_student("Bob", "102", 3.8)
    records.add_student("Charlie", "103", 3.2)
    sorted_students = quick_sort_students(records.students)
    display_top_students(sorted_students)

```

```

PS C:\AI_CODING> & C:/Users/chara/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/AI_CODING/index.py
Student Records:
Name: Alice, Roll Number: 101, CGPA: 3.5
Name: Bob, Roll Number: 102, CGPA: 3.8
Name: Charlie, Roll Number: 103, CGPA: 3.2
Sorted Student Records (Descending CGPA):
Name: Bob, Roll Number: 102, CGPA: 3.8
Name: Alice, Roll Number: 101, CGPA: 3.5
Name: Charlie, Roll Number: 103, CGPA: 3.2
Sorted Student Records (Descending CGPA) using Merge Sort:
Name: Bob, Roll Number: 102, CGPA: 3.8
Name: Alice, Roll Number: 101, CGPA: 3.5
Name: Alice, Roll Number: 101, CGPA: 3.5
Name: Charlie, Roll Number: 103, CGPA: 3.2
Quick Sort Time: 0.000036 seconds
Merge Sort Time: 0.000016 seconds
Quick Sort Time for 1000 students: 0.003273 seconds
Merge Sort Time for 1000 students: 0.004223 seconds
Merge Sort Time: 0.000016 seconds
Quick Sort Time for 1000 students: 0.003273 seconds
Merge Sort Time for 1000 students: 0.004223 seconds
Top 10 Students based on CGPA:
Name: Bob, Roll Number: 102, CGPA: 3.8
Name: Alice, Roll Number: 101, CGPA: 3.5
Top 10 Students based on CGPA:
Name: Bob, Roll Number: 102, CGPA: 3.8
Name: Alice, Roll Number: 101, CGPA: 3.5
Name: Charlie, Roll Number: 103, CGPA: 3.2

```

Task 2: 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.
- Expected Output:
 - A Bubble Sort implementation with AI-generated explanatory comments and complexity analysis.

```
128 #Write a Python implementation of Bubble Sort and include detailed inline comments explaining swapping, passes, and loop termination.
129 def bubble_sort_students(students):
130     n = len(students)
131     # Outer loop to traverse through all elements
132     for i in range(n):
133         # Flag to check if any swapping occurs
134         swapped = False
135         # Inner loop to compare adjacent elements
136         for j in range(0, n-i-1):
137             # Compare CGPA of adjacent students
138             if students[j].cgpa < students[j+1].cgpa:
139                 # If the current student's CGPA is less than the next student's CGPA, swap them
140                 students[j], students[j+1] = students[j+1], students[j]
141                 swapped = True # Set flag to True if a swap occurred
142             # If no swapping occurred, the list is already sorted, so we can break out of the loop
143             if not swapped:
144                 break
145     return students
146
147 # Example usage
148 if __name__ == "__main__":
149     records = StudentRecords()
150     records.add_student("Alice", "101", 3.5)
151     records.add_student("Bob", "102", 3.8)
152     records.add_student("Charlie", "103", 3.2)
153     sorted_students_bubble = bubble_sort_students(records.students)
154     print("Sorted Student Records (Descending CGPA) using Bubble Sort:")
155     for student in sorted_students_bubble:
156         print(f"Name: {student.name}, Roll Number: {student.roll_number}, CGPA: {student.cgpa}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\AI_CODEING> & C:/Users/chara/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/AI_CODEING/Index.py
Sorted Student Records (Descending CGPA) using Bubble Sort:
Name: Bob, Roll Number: 102, CGPA: 3.8
Name: Alice, Roll Number: 101, CGPA: 3.5
Name: Charlie, Roll Number: 103, CGPA: 3.2
PS C:\AI_CODEING>
```

Task 3: Quick Sort and Merge Sort Comparison

- Task: Implement Quick Sort and Merge Sort using recursion.
- Instructions:
- Provide AI with partially completed functions for recursion.
- Ask AI to complete the missing logic and add docstrings.
- Compare both algorithms on random, sorted, and reverse-sorted lists.
- Expected Output:
- Working Quick Sort and Merge Sort implementations.
- AI-generated explanation of average, best, and worst-case complexities.

AI CODING

index.py

index.py > quick_sort

```
155     print(f"Name: {student.name}, Roll Number: {student.roll_number}, CGPA: {student.cgpa}")
156 #Generate Python code to implement recursive Quick Sort and Merge Sort with proper docstrings and comments, then
157 import random
158 import time
159
160 def quick_sort(arr):
161     """Recursive Quick Sort"""
162     if len(arr) <= 1:
163         return arr
164     pivot = arr[len(arr) // 2]
165     left = [x for x in arr if x < pivot]
166     middle = [x for x in arr if x == pivot]
167     right = [x for x in arr if x > pivot]
168     return quick_sort(left) + middle + quick_sort(right)
169
170 def merge_sort(arr):
171     """Recursive Merge Sort"""
172     if len(arr) <= 1:
173         return arr
174
175     mid = len(arr) // 2
176     left = merge_sort(arr[:mid])
177     right = merge_sort(arr[mid:])
178     return merge(left, right)
179
180 def merge(left, right):
181     result = []
182     i = j = 0
183     while i < len(left) and j < len(right):
184         if left[i] < right[j]:
185             result.append(left[i])
186             i += 1
187         else:
188             result.append(right[j])
189             j += 1
190     result.extend(left[i:])
191     result.extend(right[j:])
192     return result
193
194 def test_case(arr, name):
195     print(f"\n{name} List Test")
196     a1 = arr.copy()
197     a2 = arr.copy()
198
199     start = time.time()
200     quick_sort(a1)
201     print("Quick Sort Time:", time.time() - start)
202
203     start = time.time()
204     merge_sort(a2)
205     print("Merge Sort Time:", time.time() - start)
206
207 size = 5000
208 random_list = random.sample(range(10000), size)
209 sorted_list = sorted(random_list)
210 reverse_list = sorted_list[::-1]
211
212 test_case(random_list, "Random")
213 test_case(sorted_list, "Sorted")
214 test_case(reverse_list, "Reverse Sorted")
215
```

```

PS C:\AI_CODING> & C:/Users/chara/AppData/Local/Microsoft
Name: Charlie, Roll Number: 103, CGPA: 3.2

Random List Test
Quick Sort Time: 0.006010293960571289

Random List Test
Quick Sort Time: 0.006010293960571289
Random List Test
Quick Sort Time: 0.006010293960571289
Quick Sort Time: 0.006010293960571289
Merge Sort Time: 0.006922245025634766

Sorted List Test
Quick Sort Time: 0.003777027130126953
Merge Sort Time: 0.00395655632019043

Sorted List Test
Quick Sort Time: 0.003777027130126953
Merge Sort Time: 0.00395655632019043

Quick Sort Time: 0.003777027130126953
Merge Sort Time: 0.00395655632019043

Merge Sort Time: 0.00395655632019043

Reverse Sorted List Test
Reverse Sorted List Test
Quick Sort Time: 0.003756999969482422
Quick Sort Time: 0.003756999969482422
Merge Sort Time: 0.005083560943603516
PS C:\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 need to:

1. Quickly search for a product by ID or name.
2. Sort products by price or quantity for stock analysis.

Task:

- Use AI to suggest the most efficient search and sort algorithms for this use case.
- Implement the recommended algorithms in Python.
- Justify the choice based on dataset size, update frequency, and performance requirements.

Expected Output:

- A table mapping operation → recommended algorithm → justification.
- Working Python functions for searching and sorting the inventory.

```
index.py > ...
215
216 #Generate Python code for an inventory management system using a Product class (product ID, name, price, qty)
217 class Product:
218     def __init__(self, pid, name, price, qty):
219         self.pid = pid
220         self.name = name
221         self.price = price
222         self.qty = qty
223
224     def __repr__(self):
225         return f"{self.pid} {self.name} ₹{self.price} Qty:{self.qty}"
226
227 inventory = [
228     Product(101, "Laptop", 60000, 5),
229     Product(102, "Phone", 20000, 20),
230     Product(103, "Keyboard", 800, 50),
231     Product(104, "Mouse", 500, 80),
232 ]
233
234 def search_by_id(pid):
235     for p in inventory:
236         if p.pid == pid:
237             return p
238     return None
239
240 def search_by_name(name):
241     for p in inventory:
242         if p.name.lower() == name.lower():
243             return p
244     return None
245
246 def sort_by_price():
247     return sorted(inventory, key=lambda x: x.price)
248
249 def sort_by_quantity():
250     return sorted(inventory, key=lambda x: x.qty)
251
252 print("\nSearch ID 102:", search_by_id(102))
253 print("Search Name 'Mouse':", search_by_name("Mouse"))
254 print("Sort by Price:", sort_by_price())
255 print("Sort by Quantity:", sort_by_quantity())
256
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Search ID 102: 102 Phone ₹20000 Qty:20
Search ID 102: 102 Phone ₹20000 Qty:20
Search Name 'Mouse': 104 Mouse ₹500 Qty:80
Sort by Price: [104 Mouse ₹500 Qty:80, 103 Keyboard ₹800 Qty:50, 102 Phone ₹20000 Qty:20, 101 Laptop ₹60000 Qty:5]
Sort by Quantity: [101 Laptop ₹60000 Qty:5, 102 Phone ₹20000 Qty:20, 103 Keyboard ₹800 Qty:50, 104 Mouse ₹500 Qty:80]
PS C:\AI_CODING>
```

Task 5: Real-Time Stock Data Sorting & Searching

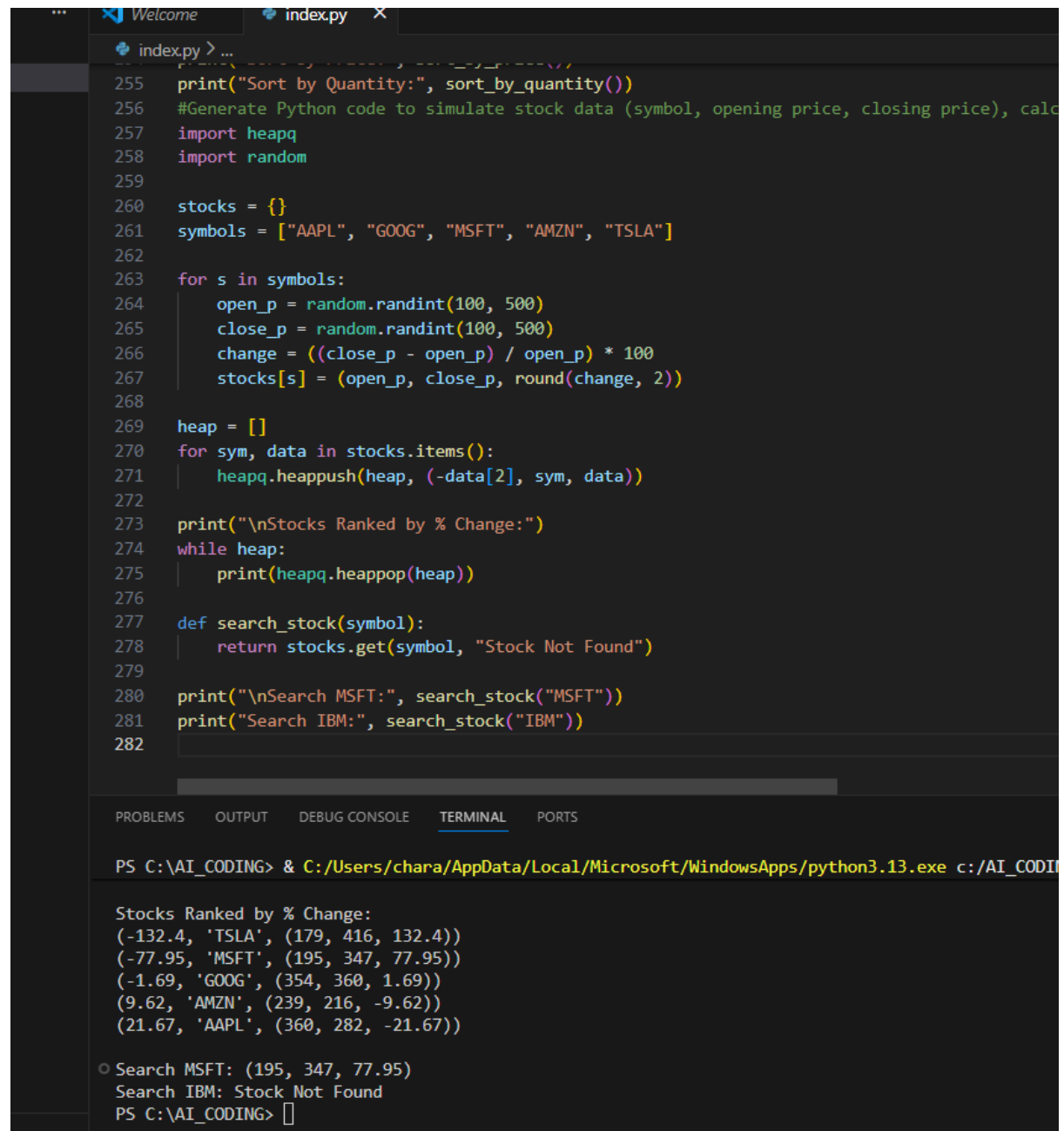
Scenario:

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.

- Use GitHub Copilot to fetch or simulate stock price data (Stock

Symbol, Opening Price, Closing Price).

- Implement sorting algorithms to rank stocks by percentage change.
- Implement a search function that retrieves stock data instantly when a stock symbol is entered.
- Optimize sorting with Heap Sort and searching with Hash Maps.
- Compare performance with standard library functions (sorted(), dict lookups) and analyze trade-offs



```
255 print("Sort by Quantity:", sort_by_quantity())
256 #Generate Python code to simulate stock data (symbol, opening price, closing price), calc
257 import heapq
258 import random
259
260 stocks = {}
261 symbols = ["AAPL", "GOOG", "MSFT", "AMZN", "TSLA"]
262
263 for s in symbols:
264     open_p = random.randint(100, 500)
265     close_p = random.randint(100, 500)
266     change = ((close_p - open_p) / open_p) * 100
267     stocks[s] = (open_p, close_p, round(change, 2))
268
269 heap = []
270 for sym, data in stocks.items():
271     heapq.heappush(heap, (-data[2], sym, data))
272
273 print("\nStocks Ranked by % Change:")
274 while heap:
275     print(heapq.heappop(heap))
276
277 def search_stock(symbol):
278     return stocks.get(symbol, "Stock Not Found")
279
280 print("\nSearch MSFT:", search_stock("MSFT"))
281 print("Search IBM:", search_stock("IBM"))
282
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\AI_CODING> & C:/Users/chara/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/AI_CODI

Stocks Ranked by % Change:
(-132.4, 'TSLA', (179, 416, 132.4))
(-77.95, 'MSFT', (195, 347, 77.95))
(-1.69, 'GOOG', (354, 360, 1.69))
(9.62, 'AMZN', (239, 216, -9.62))
(21.67, 'AAPL', (360, 282, -21.67))

○ Search MSFT: (195, 347, 77.95)
Search IBM: Stock Not Found
PS C:\AI_CODING>
```