

ASSIGNMENT 12.1

Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

AmruthSagar Vemuganti

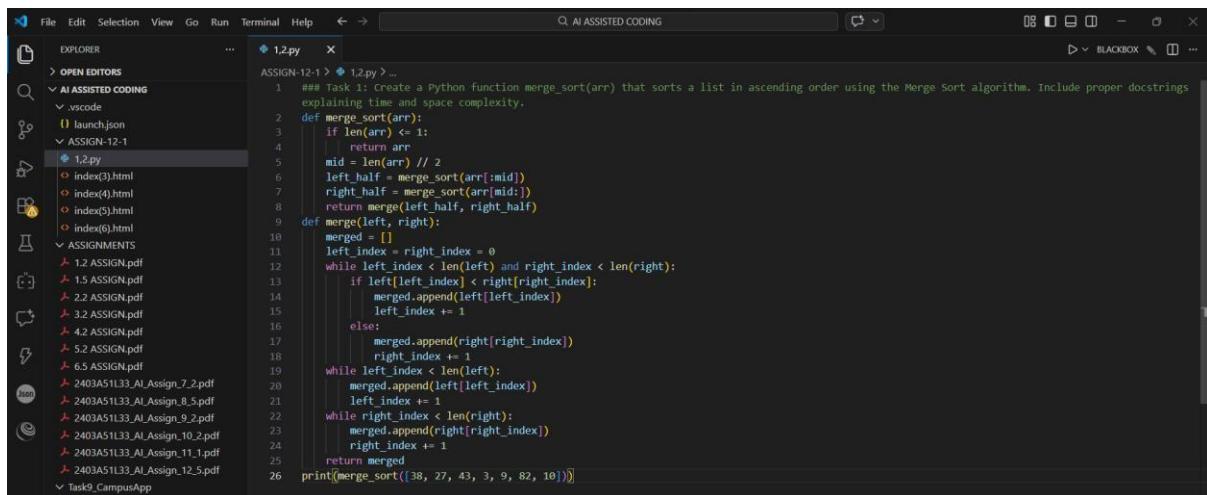
2403a51l44

B-52

Task Description #1 (Sorting – Merge Sort Implementation):

PROMPT: Create a Python function `merge_sort(arr)` that sorts a list in ascending order using the Merge Sort algorithm. Include proper docstrings explaining time and space complexity.

CODE:



```
1  """ Task 1: Create a Python function merge_sort(arr) that sorts a list in ascending order using the Merge Sort algorithm. Include proper docstrings explaining time and space complexity.
2  def merge_sort(arr):
3      if len(arr) <= 1:
4          return arr
5      mid = len(arr) // 2
6      left_half = merge_sort(arr[:mid])
7      right_half = merge_sort(arr[mid:])
8      return merge(left_half, right_half)
9
10     merged = []
11     left_index = right_index = 0
12     while left_index < len(left) and right_index < len(right):
13         if left[left_index] < right[right_index]:
14             merged.append(left[left_index])
15             left_index += 1
16         else:
17             merged.append(right[right_index])
18             right_index += 1
19     while left_index < len(left):
20         merged.append(left[left_index])
21         left_index += 1
22     while right_index < len(right):
23         merged.append(right[right_index])
24         right_index += 1
25     return merged
26
27 print(merge_sort([38, 27, 43, 3, 9, 82, 10]))
```

OUTPUT:



```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> python 1.2.py
[3, 9, 10, 27, 43, 82, 88]
```

EXPLANATION: AI generated a divide-and-conquer based sorting solution with detailed complexity analysis. The output was verified using multiple test cases to ensure correctness and efficiency.

Task Description #2 (Searching – Binary Search with AI Optimization):

PROMPT: Develop a Python function `binary_search(arr, target)` to search for an element in a sorted list and return its index or -1 if not found. Add documentation describing best, average, and worst-case complexities.

CODE:

```

1.2.py
ASSIGN-12-1 > 1.2.py > binary_search
24     right_index += 1
25     return merged
26 print(merge_sort([38, 27, 43, 3, 9, 82, 10]))"""
27
28 ### Task 2: Develop a Python function binary_search(arr, target) to search for an element in a sorted list and return its index or -1 if not found.
29 def binary_search(arr, target):
30     left, right = 0, len(arr) - 1
31     while left <= right:
32         mid = left + (right - left) // 2
33         if arr[mid] == target:
34             return mid
35         elif arr[mid] < target:
36             left = mid + 1
37         else:
38             right = mid - 1
39     return -1
40
41 sorted_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
42 target = 5
43 result = binary_search(sorted_list, target)
44 if result != -1:
45     print(f"Element {target} found at index: {result}")
46 else:
47     print(f"Element {target} not found in the list.")

```

OUTPUT:

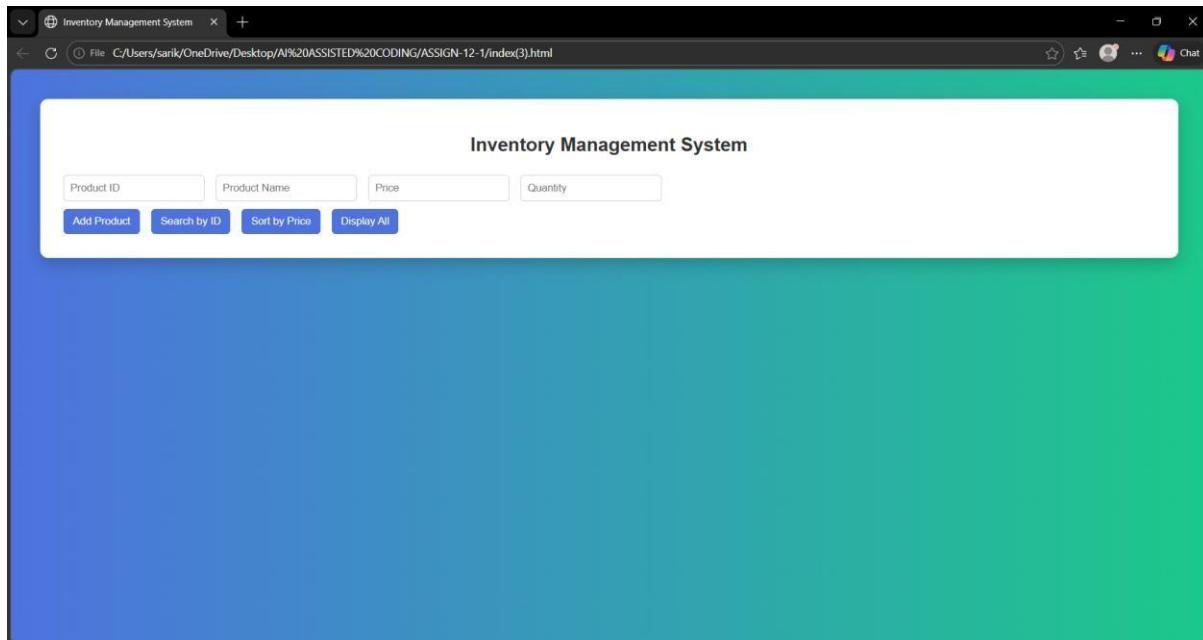
```

PS C:\Users\sarik\OneDrive\Desktop\AI%20ASSISTED%20CODING\ASSIGN-12-1\1.2.py
Element 5 found at index: 4
PS C:\Users\sarik\OneDrive\Desktop\AI%20ASSISTED%20CODING>

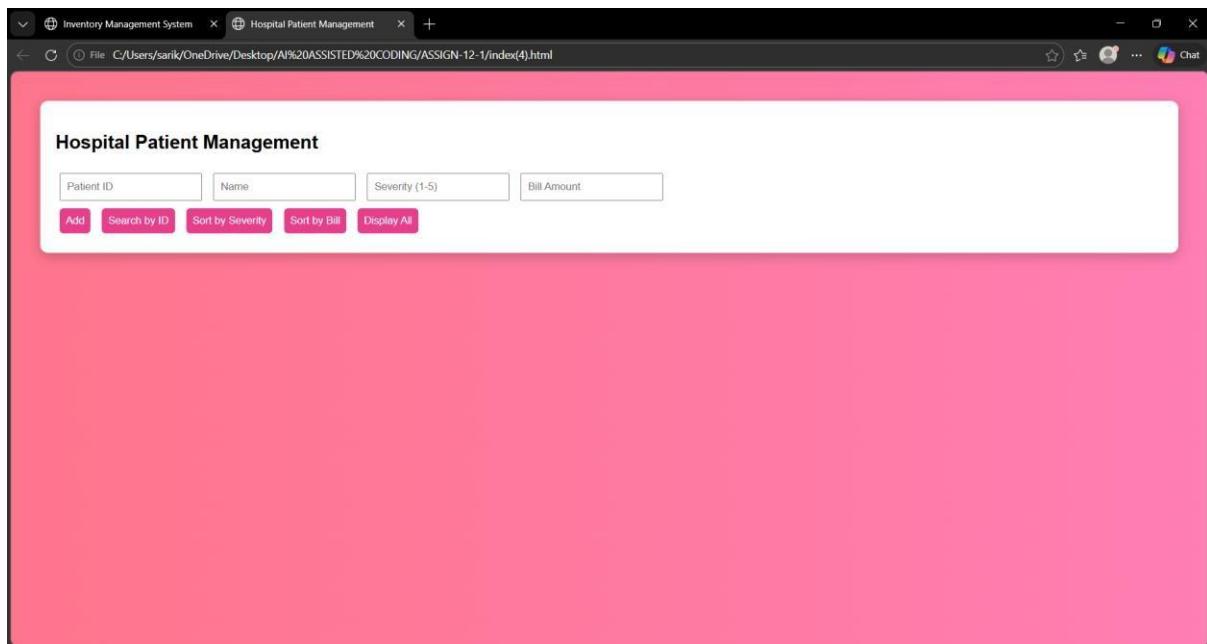
```

EXPLANATION: AI produced an optimized searching algorithm that reduces comparisons using a halving strategy. The function was tested with different inputs to validate accuracy and performance.

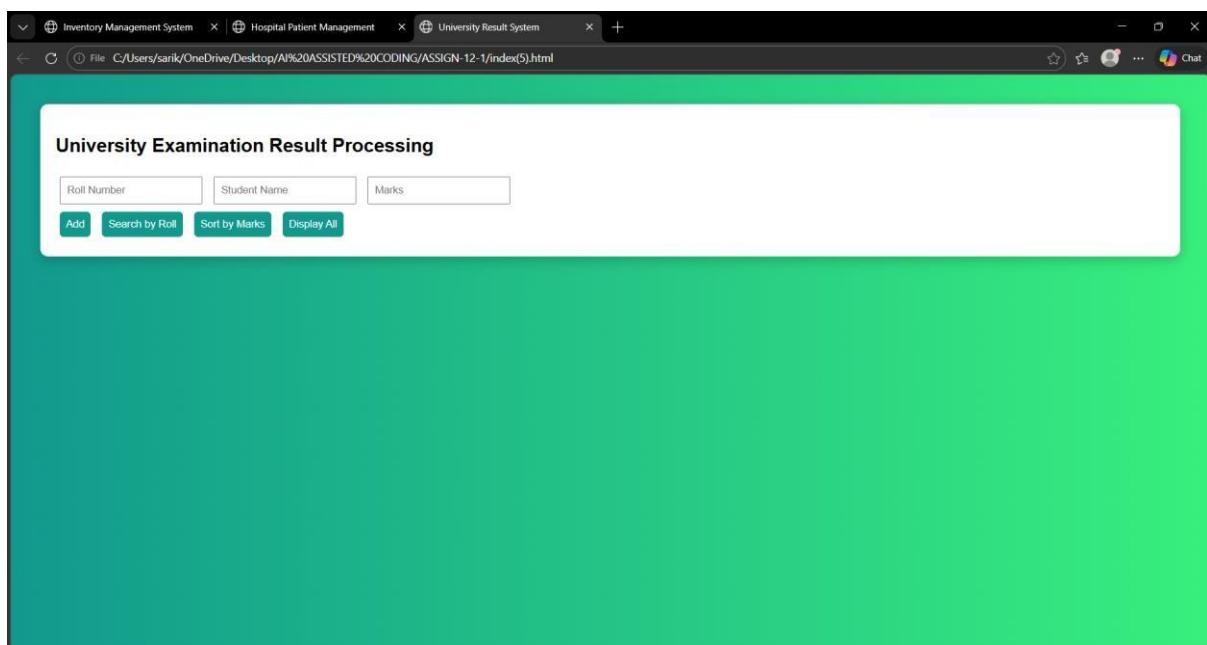
Task Description #3 (Real-Time Application – Inventory Management System):



Task description #4 (Smart Hospital Patient Management System):



Task Description #5 (University Examination Result Processing System):



Task Description #6 (Online Food Delivery Platform):

