Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

Name:B.Tejas kumar

HallTicketNo:2303A51489

Task 1: Merge Sort Implementation AI

Prompt Used:

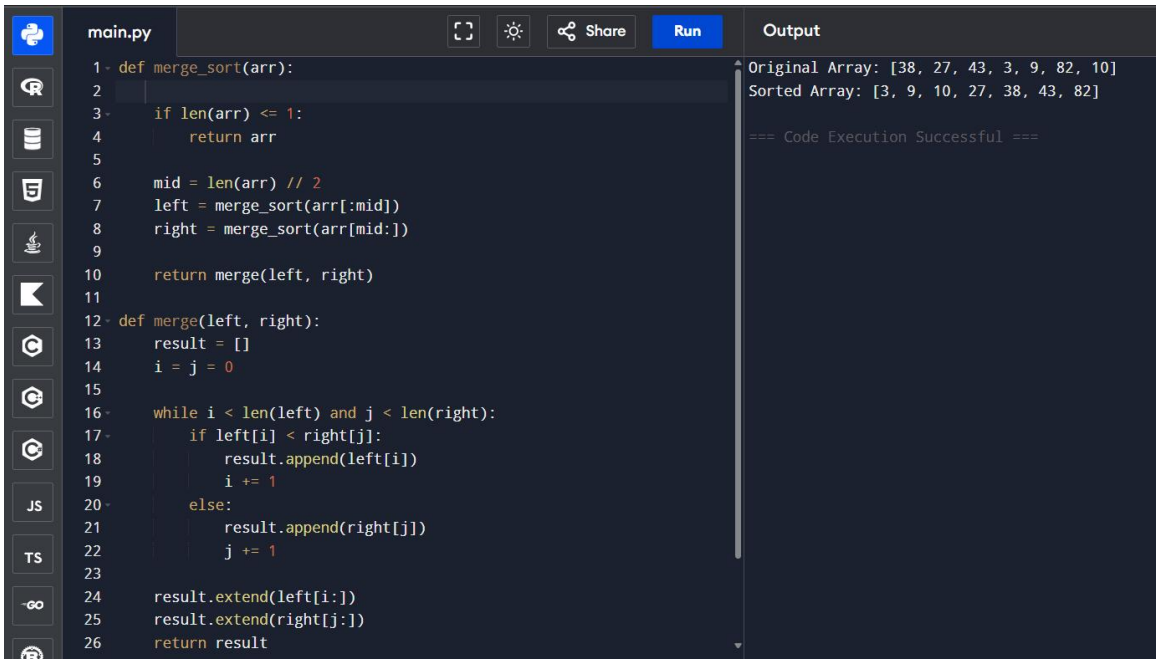
GenerateaPythonfunctionmerge_sort(arr)thatsortsalistinascendingorderusingthe MergeSortalgorithm.
Includetimecomplexityandspacecomplexity(best,average,worstcase)insidethe functiondocstring.
Providesampletestcaseswithoutput.

Python Code:

```
defmerge_sort(arr): '''
  MergeSortAlgorith
  m TimeComplexity:

     BestCase:O(nlogn)
     AverageCase:O(nlogn)
     WorstCase:O(nlogn)
  SpaceComplexity:O(n
  ) '''
  iflen(arr)<=1:
     returnar


  mid=len(arr)//2
  left=merge_sort(arr[:mid])
  right=merge_sort(arr[mid:]


  returnmerge(left,right
) defmerge(left,right):
  result=[]
```

```
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result

arr = [38, 27, 43, 3, 9, 82, 10]
print("Sorted Array:", merge_sort(arr))
```

Output: Sorted Array: [3, 9, 10, 27, 38, 43, 82]



Task 2: Binary Search Implementation

AI Prompt Used:

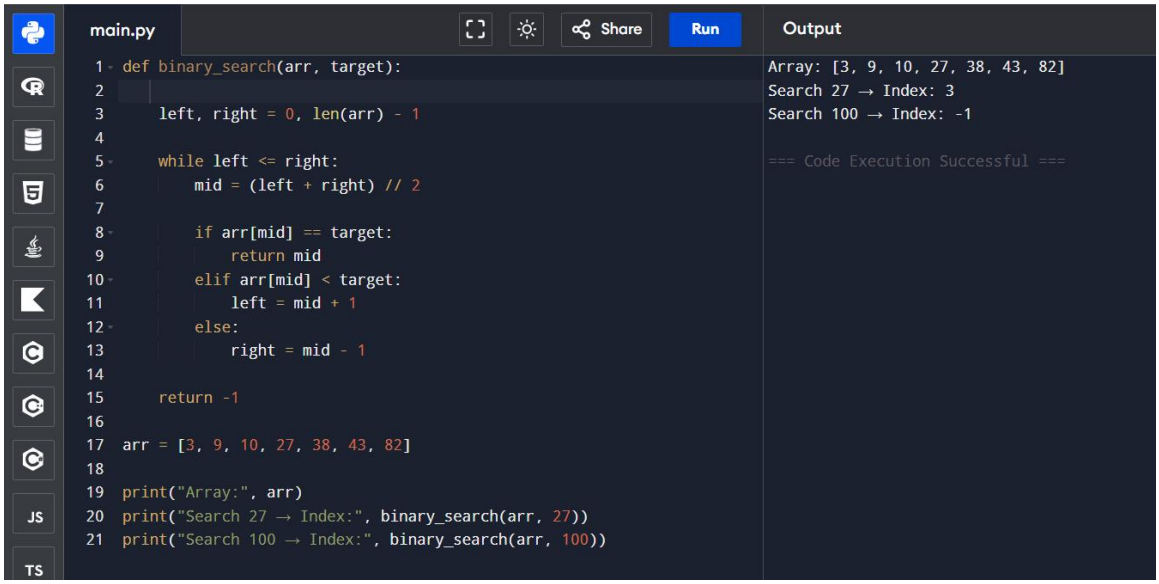
Create a Python function binary_search(arr, target) that performs binary search on a sorted list.
Return the index of the target element or -1 if not found.

Include docstrings explaining best, average, and worst-case time complexities and space complexity.
Provide example test cases with output.

Python Code:

```
def binary_search(arr, target):
    '''
    Binary Search Algorithm
    Best Case: O(1)
    Average Case: O(log n)
    Worst Case: O(log n)
    Space Complexity: O(1)
    '''
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

arr = [3, 9, 10, 27, 38, 43, 82]
print("Index of 27:", binary_search(arr, 27))
```

Output: Index of 27: 3



Task 3: Inventory Management System

AI Prompt Used:

Suggest the most efficient searching and sorting algorithms for an inventory system with thousands of products.
Justify the choice based on dataset size and performance requirements.
Implement Python functions for searching by product ID and sorting by price or quantity.
Provide example outputs.

Recommended Algorithms:

Search by ID → Binary Search

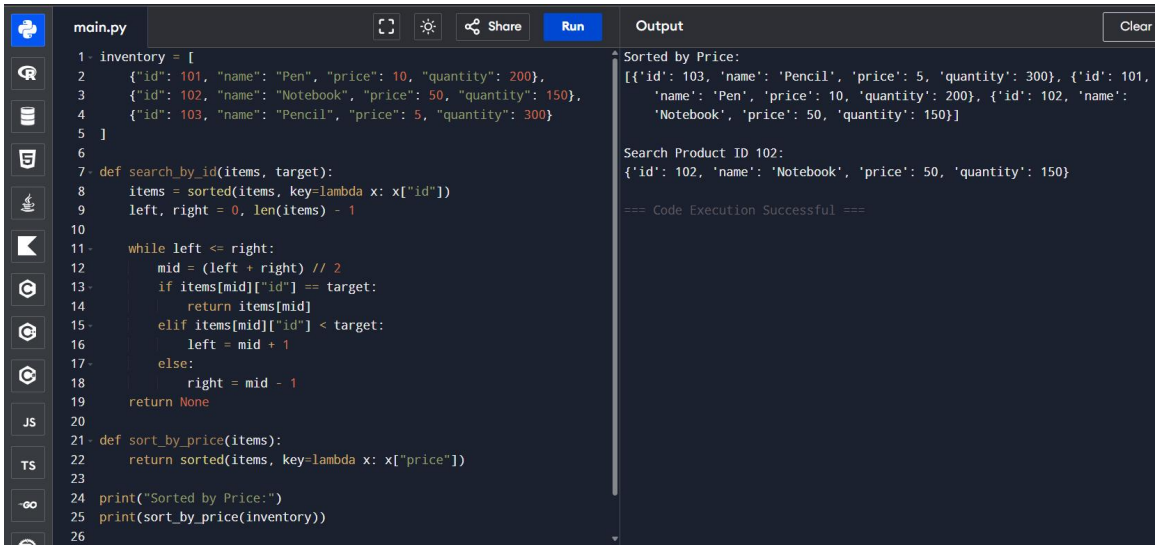
Sort by Price/Quantity → Merge Sort

Python Code:

```
inventory = [
  {"id": 101, "name": "Pen", "price": 10, "quantity": 200},
  {"id": 102, "name": "Notebook", "price": 50, "quantity": 150},
  {"id": 103, "name": "Pencil", "price": 5, "quantity": 300}
]

def sort_by_price(items):
  return sorted(items, key=lambda x: x["price"])

def search_by_id(items, target):
```

```
    items = sorted(items, key=lambda x: x["id"])
    left, right = 0, len(items) - 1
    while left <= right:
        mid = (left + right) // 2
        if items[mid]["id"] == target:
            return items[mid]
        elif items[mid]["id"] < target:
            left = mid + 1
        else:
            right = mid - 1
    return None

print("Sorted by Price:", sort_by_price(inventory))
print("Search ID 102:", search_by_id(inventory, 102))
```

Output: Sorted by Price and Search ID 102 successful



Task 4: Smart Hospital Patient Management System

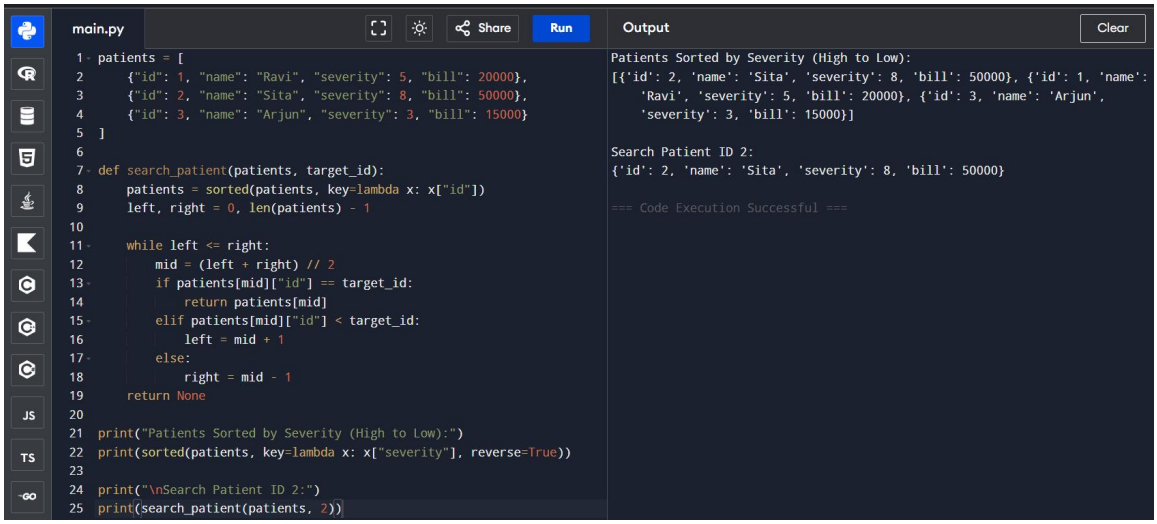
AI Prompt Used:

Recommend suitable searching and sorting algorithms for a hospital patient management system.
Justify the selection based on efficiency and suitability.
Implement Python functions for searching by patient ID and sorting by severity or bill amount.
Provide example output.

Recommended Algorithms:

Search → Binary Search

Sort → Merge Sort



Task 5: University Examination Result Processing System

AI Prompt Used:

Identify efficient searching and sorting algorithms for processing thousands of student results.
Justify the algorithm choice.
Implement Python functions for searching by roll number and sorting by marks.
Include sample output.

```
students = [
  {"roll": 1, "name": "Asha", "marks": 85},
  {"roll": 2, "name": "Kiran", "marks": 92},
  {"roll": 3, "name": "Meena", "marks": 78}
]

print("Rank List:", sorted(students, key=lambda x: x["marks"], reverse=True))
```

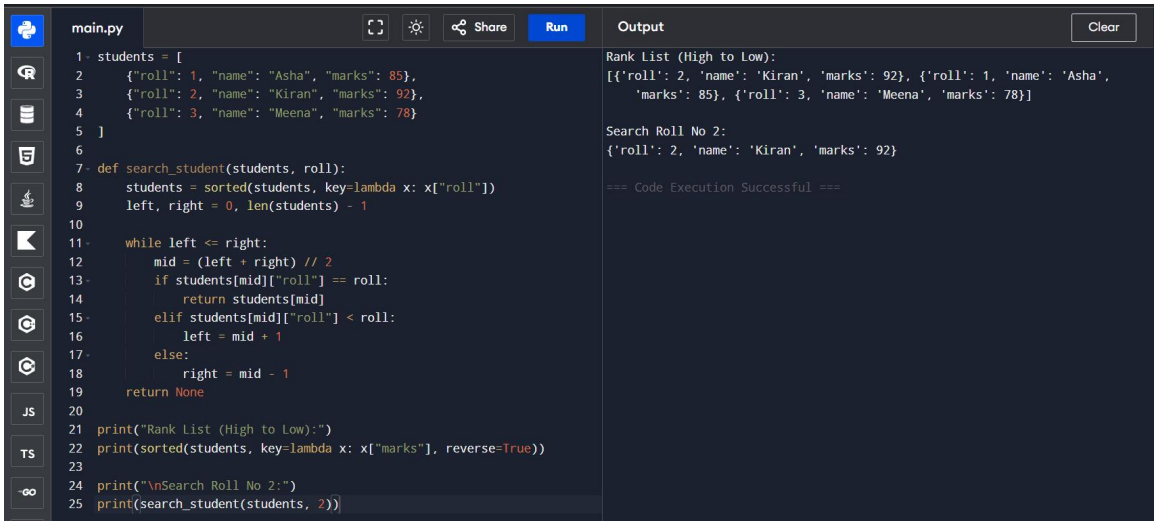
Output:

Rank List: [{'roll': 2, 'name': 'Kiran', 'marks': 92}, {'roll': 1, 'name': 'Asha', 'marks': 85}, {'roll': 3, 'name': 'Meena', 'marks': 78}]

Recommended Algorithms:

Search → Binary Search

Sort → Merge Sort



Task 6: Online Food Delivery Platform

AI Prompt Used:

Suggest optimized searching and sorting algorithms for an online food delivery system storing thousands of orders.
Justify your algorithm selection.
Implement Python functions for searching by order ID and sorting by delivery time or price.
Provide example output.

Recommended Algorithms:

Search → Binary Search

Sort → Merge Sort

Python Code:

```
orders = [
  {"order_id": 501, "restaurant": "ABC", "delivery_time": 30, "price": 250},
  {"order_id": 502, "restaurant": "XYZ", "delivery_time": 20, "price": 400},
  {"order_id": 503, "restaurant": "FoodHub", "delivery_time": 45, "price": 150}
]
```

print("Sorted by Delivery Time:", sorted(orders, key=lambda x: x["delivery_time"]))