

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:

Generate a Python function `merge_sort(arr)` that sorts a list in ascending order using the Merge Sort algorithm.
Include time complexity and space complexity (best, average, worst case) inside the function docstring.
Provide sample test cases without output.

Python Code:

```
def merge_sort(arr): ""  
    MergeSortAlgorithm  
    m TimeComplexity:  
        BestCase:O(nlogn)  
        AverageCase:O(nlogn)  
        WorstCase:O(nlogn)  
    SpaceComplexity:O(n  
    ) ""  
    if len(arr)<=1:  
        return arr  
  
    mid=len(arr)//2  
    left=merge_sort(arr[:mid])  
    right=merge_sort(arr[mid:])  
  
    return merge(left,right)  
def merge(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]

```

main.py
1 - def merge_sort(arr):
2
3 -     if len(arr) <= 1:
4 -         return arr
5
6 -     mid = len(arr) // 2
7 -     left = merge_sort(arr[:mid])
8 -     right = merge_sort(arr[mid:])
9
10 -    return merge(left, right)
11
12 - def merge(left, right):
13 -     result = []
14 -     i = j = 0
15
16 -     while i < len(left) and j < len(right):
17 -         if left[i] < right[j]:
18 -             result.append(left[i])
19 -             i += 1
20 -         else:
21 -             result.append(right[j])
22 -             j += 1
23
24 -     result.extend(left[i:])
25 -     result.extend(right[j:])
26 -     return result

```

Original Array: [38, 27, 43, 3, 9, 82, 10]
Sorted Array: [3, 9, 10, 27, 38, 43, 82]
--- Code Execution Successful ---

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

```

main.py
1 def binary_search(arr, target):
2     left, right = 0, len(arr) - 1
3
4     while left <= right:
5         mid = (left + right) // 2
6
7         if arr[mid] == target:
8             return mid
9         elif arr[mid] < target:
10            left = mid + 1
11        else:
12            right = mid - 1
13
14    return -1
15
16 arr = [3, 9, 10, 27, 38, 43, 82]
17
18 print("Array:", arr)
19 print("Search 27 → Index:", binary_search(arr, 27))
20 print("Search 100 → Index:", binary_search(arr, 100))

```

Output

```

Array: [3, 9, 10, 27, 38, 43, 82]
Search 27 → Index: 3
Search 100 → Index: -1
== Code Execution Successful ==

```

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

```

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

Sorted by Price:
[{"id": 103, "name": "Pencil", "price": 5, "quantity": 300}, {"id": 101, "name": "Pen", "price": 10, "quantity": 200}, {"id": 102, "name": "Notebook", "price": 50, "quantity": 150}]

Search Product ID 102:
{"id": 102, "name": "Notebook", "price": 50, "quantity": 150}

== Code Execution 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

The screenshot shows a Python code editor interface with a dark theme. On the left, there is a sidebar with various icons for file types like Python, SQL, HTML, CSS, JS, TS, and Go. The main area has tabs for 'main.py' and 'Output'. The code in 'main.py' is as follows:

```
patients = [
    {"id": 1, "name": "Ravi", "severity": 5, "bill": 20000},
    {"id": 2, "name": "Sita", "severity": 8, "bill": 50000},
    {"id": 3, "name": "Arjun", "severity": 3, "bill": 15000}
]

def search_patient(patients, target_id):
    patients = sorted(patients, key=lambda x: x["id"])
    left, right = 0, len(patients) - 1

    while left <= right:
        mid = (left + right) // 2
        if patients[mid]["id"] == target_id:
            return patients[mid]
        elif patients[mid]["id"] < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return None

print("Patients Sorted by Severity (High to Low):")
print(sorted(patients, key=lambda x: x["severity"], reverse=True))

print("\nSearch Patient ID 2:")
print(search_patient(patients, 2))

print("\nSearch Patient ID 2:")
print(search_patient(patients, 2))
```

The 'Output' tab shows the execution results:

```
Patients Sorted by Severity (High to Low):
[{"id": 2, "name": "Sita", "severity": 8, "bill": 50000}, {"id": 1, "name": "Ravi", "severity": 5, "bill": 20000}, {"id": 3, "name": "Arjun", "severity": 3, "bill": 15000}

Search Patient ID 2:
{"id": 2, "name": "Sita", "severity": 8, "bill": 50000}

== Code Execution Successful ==
```

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))
```

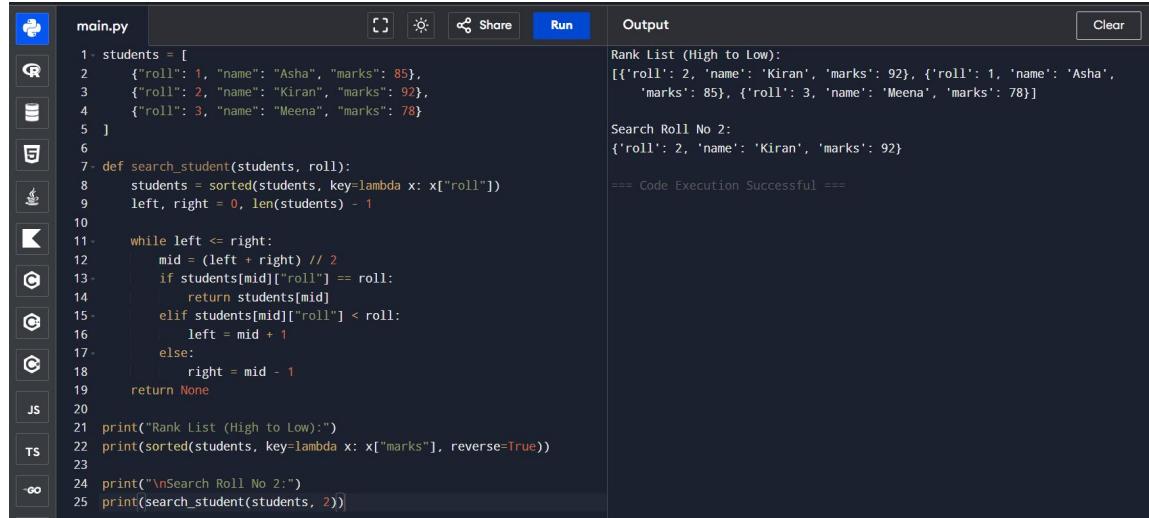
Output:

```
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



The screenshot shows a Python code editor with a dark theme. The file is named 'main.py'. The code implements a binary search algorithm to find a student by roll number and prints the rank list and search results.

```
main.py
1 students = [
2     {"roll": 1, "name": "Asha", "marks": 85},
3     {"roll": 2, "name": "Kiran", "marks": 92},
4     {"roll": 3, "name": "Meena", "marks": 78}
5 ]
6
7 def search_student(students, roll):
8     students = sorted(students, key=lambda x: x["roll"])
9     left, right = 0, len(students) - 1
10
11    while left <= right:
12        mid = (left + right) // 2
13        if students[mid]["roll"] == roll:
14            return students[mid]
15        elif students[mid]["roll"] < roll:
16            left = mid + 1
17        else:
18            right = mid - 1
19    return None
20
21 print("Rank List (High to Low):")
22 print(sorted(students, key=lambda x: x["marks"], reverse=True))
23
24 print("\nSearch Roll No 2:")
25 print(search_student(students, 2))
```

The output window shows the rank list and the search result for roll number 2.

```
Rank List (High to Low):
[{"roll": 2, "name": "Kiran", "marks": 92}, {"roll": 1, "name": "Asha", "marks": 85}, {"roll": 3, "name": "Meena", "marks": 78}]

Search Roll No 2:
{'roll': 2, 'name': 'Kiran', 'marks': 92}

== Code Execution Successful ==
```

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"]))
```

```
main.py
  "price": 250},
3   {"order_id": 502, "restaurant": "XYZ", "delivery_time": 20,
  "price": 400},
4   {"order_id": 503, "restaurant": "FoodHub", "delivery_time": 45,
  "price": 150}
5 ]
6
7 def search_order(orders, order_id):
8     orders = sorted(orders, key=lambda x: x["order_id"])
9     left, right = 0, len(orders) - 1
10
11    while left <= right:
12        mid = (left + right) // 2
13        if orders[mid]["order_id"] == order_id:
14            return orders[mid]
15        elif orders[mid]["order_id"] < order_id:
16            left = mid + 1
17        else:
18            right = mid - 1
19    return None
20
21 print("Orders Sorted by Delivery Time:")
22 print(sorted(orders, key=lambda x: x["delivery_time"]))
23
24 print("\nSearch Order ID 502:")
```

Output

```
Orders Sorted by Delivery Time:
[{"order_id": 502, "restaurant": "XYZ", "delivery_time": 20, "price": 400},
 {"order_id": 501, "restaurant": "ABC", "delivery_time": 30, "price": 250}, {"order_id": 503, "restaurant": "FoodHub", "delivery_time": 45, "price": 150}]

Search Order ID 502:
{"order_id": 502, "restaurant": "XYZ", "delivery_time": 20, "price": 400}

== Code Execution Successful ==
```