

SR UNIVERSITY

AI ASSIST CODING

Lab-21.1: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

ROLL NO:2503A51L17

NAME:Simra Tahseen

Batchno.:24B2CAICSB19

Lab Objectives:

Apply AI-assisted programming to implement and optimize sorting and searching algorithms.

- Compare different algorithms in terms of efficiency and use cases. •
- Understand how AI tools can suggest optimized code and complexity improvements.

Task 1: (Sorting – Merge Sort Implementation)

Prompt : Write a Python program to implement merge sort with proper docstrings. Include time/space complexity analysis and test cases."

Code:

```
def merge_sort(arr):
```

```
    """
```

```
        Perform Merge Sort on a list of numbers.
```

Parameters:

arr (list): List of numbers to be sorted.

Returns:

list: Sorted list in ascending order.

Time Complexity:

Best Case: $O(n \log n)$

Average Case: $O(n \log n)$

Worst Case: $O(n \log n)$

Space Complexity:

$O(n)$ - requires extra space for merging.

```
"""    if len(arr) <= 1:
return arr    mid = len(arr) // 2
left_half = merge_sort(arr[:mid])
right_half =
merge_sort(arr[mid:]) return
merge(left_half, right_half) def
merge(left, right):
    """Helper function to merge two sorted lists into one sorted list."""
    result = []
    i = j = 0    # Compare elements from left and
right    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
# Add remaining elements (if any)
    result.extend(left[i:])    result.extend(right[j:])    return result
print("Merge Sort Tests:") print(merge_sort([38, 27, 43, 3, 9, 82, 10])) #
Expected: [3, 9, 10, 27, 38, 43, 82] print(merge_sort([5, 2, 4, 6, 1, 3]))
# Expected: [1, 2, 3, 4, 5, 6] print(merge_sort([1]))    #
Expected: [1]

print(merge_sort([]))
```

Code Generated:

```
mean.py > merge.py > ...
1 def merge_sort(arr):
2     """
3     Perform Merge Sort on a list of numbers.
4
5     Parameters:
6     arr (list): List of numbers to be sorted.
7
8     Returns:
9     list: Sorted list in ascending order.
10
11     Time Complexity:
12     Best Case: O(n log n)
13     Average Case: O(n log n)
14     Worst Case: O(n log n)
15
16     Space Complexity:
17     O(n) - requires extra space for merging.
18     """
19     if len(arr) <= 1:
20         return arr
21
22     mid = len(arr) // 2
23     left_half = merge_sort(arr[:mid])
24     right_half = merge_sort(arr[mid:])
25
26     return merge(left_half, right_half)
27
28
29 def merge(left, right):
30     """Helper function to merge two sorted lists into one sorted list."""
31     result = []
32     i = j = 0
33
34     # Compare elements from left and right
35     while i < len(left) and j < len(right):
36         if left[i] < right[j]:
37             result.append(left[i])
38             i += 1
39         else:
40             result.append(right[j])
41             j += 1
42
43     # Add remaining elements (if any)
44     result.extend(left[i:])
45     result.extend(right[j:])
46     return result
47
48
49 # Test cases
50 print("Merge Sort Tests:")
51 print(merge_sort([38, 27, 43, 3, 9, 82, 10])) # Expected: [3, 9, 10, 27, 38, 43, 82]
52 print(merge_sort([5, 2, 4, 6, 1, 3])) # Expected: [1, 2, 3, 4, 5, 6]
53 print(merge_sort([1])) # Expected: [1]
54 print(merge_sort([])) # Expected: []
```

Expected Output:

```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING\mean.py\merge.py"
Merge Sort Tests:
[3, 9, 10, 27, 38, 43, 82]
[1, 2, 3, 4, 5, 6]
[1]
[]
```

Observations:

- Merge Sort divides the array into halves recursively until single elements remain.
- The merging step combines two sorted halves into a single sorted list.
- Default order is ascending; comparison logic can be inverted for descending.
- Time complexity is consistently $O(n \log n)$ in best, average, and worst cases.
- Space complexity is $O(n)$ due to extra lists used in merging.

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

Prompt : Write a Python program to implement binary search with proper docstrings. Include time/space complexity analysis and test cases.

Code :

```
def binary_search(arr, target):
```

```
    """
```

```
    Perform Binary Search on a sorted list.
```

```
    Parameters:
```

```
        arr (list): Sorted list of numbers.
```

```
        target (int): The number to search for.
```

```
    Returns:
```

```
        int: Index of the target if found, else -1.
```

```
    Time Complexity:
```

```
        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
```

code generated:

```
mean.py >  binar.py >  binary_search
```

```
1 def binary_search(arr, target):
2     """
3     Perform Binary Search on a sorted list.
4     Parameters:
5         arr (list): Sorted list of numbers.
6         target (int): The number to search for.
7     Returns:
8         int: Index of the target if found, else -1.
9     Time Complexity:
10         Best Case: O(1)
11         Average Case: O(log n)
12         Worst Case: O(log n)
13     Space Complexity:
14         O(1)
15     """
16     left, right = 0, len(arr) - 1
17
18     while left <= right:
19         mid = (left + right) // 2
20         if arr[mid] == target:
21             return mid
22         elif arr[mid] < target:
23             left = mid + 1
24         else:
25             right = mid - 1
26
27     return -1
28
29
30 # Function calls (test cases)
31 arr = [1, 3, 5, 7, 9, 11, 13]
32
33 print(binary_search(arr, 7)) # Expected output: 3
34 print(binary_search(arr, 1)) # Expected output: 0
35 print(binary_search(arr, 13)) # Expected output: 6
36 print(binary_search(arr, 2)) # Expected output: -1
```

Expected Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe" -c "import sys; sys.path.append('C:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING'); import mean; mean.mean([3, 0, 6, -1])"
3
0
6
-1

```

Observations:

- Binary Search is efficient as it halves the search range each step.
- It only works on sorted data.
- Time Complexity → Best: $O(1)$, Worst/Average: $O(\log n)$.
- Space Complexity → $O(1)$ (iterative).
- Much faster than linear search for large datasets.
- Limitation: Requires re-sorting if data changes often.

Task 3: (Real-Time Application – InventoryManagement 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.

Prompt :

1. implement the recommended algorithms in Python.
2. Justify the choice based on dataset size, update frequency, and performance requirements.

Code:

```
def build_id_index(products):
    """Build a hash table (dictionary) for fast product ID lookups."""
    return {product['id']: product for product in products}

def search_by_id(products_index, product_id):
    """Efficient search by product ID using hash table."""
    return products_index.get(product_id)

def search_by_name(products, name, partial=False):
    """
    Search by product name.
    For a partial match, return products containing the substring.
    For exact, match the full name (case-insensitive).
    """
    if partial:
        return [p for p in products if name.lower() in p['name'].lower()]
    else:
        return [p for p in products if p['name'].lower() == name.lower()]

def quick_sort(products, key):
    """Sort products by given key with Quick Sort."""
    if len(products) <= 1:
        return products
    pivot = products[0]
    left = [x for x in products[1:] if x[key] < pivot[key]]
    right = [x for x in products[1:] if x[key] >= pivot[key]]
    return quick_sort(left, key) + [pivot] + quick_sort(right, key)

# --- Interactive User Input ---

products = []
n = int(input("Enter number of products: "))
for _ in range(n):
```

```

prod_id = int(input("Enter Product ID: "))
name = input("Enter Product Name: ")
price = float(input("Enter Product Price: "))
quantity = int(input("Enter Product Quantity: "))
products.append({'id': prod_id, 'name': name, 'price': price, 'quantity': quantity})

products_index = build_id_index(products)

# --- Menu Loop ---
while True:
    print("""
Menu:
1. Search by ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
""")
    choice = input("Enter choice: ")
    if choice == '1':
        query_id = int(input("Enter Product ID to search: "))
        result = search_by_id(products_index, query_id)
        print("Result:", result if result else "Not found.")
    elif choice == '2':
        name = input("Enter Product Name to search: ")
        partial = input("Partial match? (y/n): ").lower() == 'y'
        result = search_by_name(products, name, partial)
        print("Results:", result if result else "None found.")
    elif choice == '3':
        sorted_list = quick_sort(products, 'price')
        print("Sorted by price:", sorted_list)
    elif choice == '4':
        sorted_list = quick_sort(products, 'quantity')
        print("Sorted by quantity:", sorted_list)
    elif choice == '5':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Please try again.")

```

Code Generated:

```
File Edit Selection View Go Run Terminal Help  AI CODING

main.py  inventory.py ~  database

> OPEN EDITORS
> AI CODING
> TIMELINE

1 def build_id_index(products):
2     """Build a hash table (dictionary) for fast product ID lookups."""
3     return {product['id']: product for product in products}
4
5 def search_by_id(products_index, product_id):
6     """Efficient search by product ID using hash table."""
7     return products_index.get(product_id)
8
9 def search_by_name(products, name, partial=False):
10    """
11    Search by product name.
12    For a partial match, return products containing the substring.
13    For exact, match the full name (case-insensitive).
14    """
15    if partial:
16        return [p for p in products if name.lower() in p['name'].lower()]
17    else:
18        return [p for p in products if p['name'].lower() == name.lower()]
19
20 def quick_sort(products, key):
21    """Sort products by given key with Quick Sort."""
22    if len(products) <= 1:
23        return products
24    pivot = products[0]
25    left = [x for x in products[1:] if x[key] < pivot[key]]
26    right = [x for x in products[1:] if x[key] >= pivot[key]]
27    return quick_sort(left, key) + [pivot] + quick_sort(right, key)
28
29 # --- Interactive User Input ---
30
31 products = []
32 n = int(input("Enter number of products: "))
33 for _ in range(n):
34     prod_id = int(input("Enter Product ID: "))
35     name = input("Enter Product Name: ")
36     price = float(input("Enter Product Price: "))
37     quantity = int(input("Enter Product Quantity: "))
38     products.append({'id': prod_id, 'name': name, 'price': price, 'quantity': quantity})
39
40 products_index = build_id_index(products)
41
42 # --- Menu Loop ---
43 while True:
44     print("""
45     Menu:
46     1. Search by ID
47     2. Search by Name
48     3. Sort by Price
49     4. Sort by Quantity
50     5. Exit
51     """)
52     choice = input("Enter choice: ")
53     if choice == '1':
54         query_id = int(input("Enter Product ID to search: "))
55         result = search_by_id(products_index, query_id)
56         print("Result:", result if result else "Not found.")
57     elif choice == '2':
58         name = input("Enter Product Name to search: ")
59         partial = input("Partial match? (y/n): ").lower() == 'y'
60         result = search_by_name(products, name, partial)
61         print("Results:", result if result else "None found.")
62     elif choice == '3':
63         sorted_list = quick_sort(products, 'price')
64         print("Sorted by price:", sorted_list)
65     elif choice == '4':
66         sorted_list = quick_sort(products, 'quantity')
67         print("Sorted by quantity:", sorted_list)
68     elif choice == '5':
69         print("Exiting program.")
70         break
71     else:
72         print("Invalid choice. Please try again.")
73
```


Expected Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "C:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/mean.py/inventory.py"
Enter number of products: 3
Enter Product ID: 181
Enter Product Name: Apple
Enter Product Price: 50
Enter Product Quantity: 100
Enter Product ID: 182
Enter Product Name: Grapes
Enter Product Price: 80
Enter Product Quantity: 150
Enter Product ID: 183
Enter Product Name: Banana
Enter Product Price: 20
Enter Product Quantity: 120

Menu:
1. Search by ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit

Enter choice: 1
Enter Product ID to search: 182
Result: [{'id': 182, 'name': 'Grapes', 'price': 80.0, 'quantity': 150}]

Menu:
1. Search by ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit

Enter choice: 2
Enter Product Name to search: Apple
Partial match? (y/n): n
Results: [{'id': 181, 'name': 'Apple', 'price': 50.0, 'quantity': 100}]

Menu:
1. Search by ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit

Enter choice: 3
Sorted by price: [{'id': 183, 'name': 'Banana', 'price': 20.0, 'quantity': 120}, {'id': 181, 'name': 'Apple', 'price': 50.0, 'quantity': 100}, {'id': 182, 'name': 'Grapes', 'price': 80.0, 'quantity': 150}]

Menu:
1. Search by ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit

Enter choice: 4
Sorted by quantity: [{'id': 181, 'name': 'Apple', 'price': 50.0, 'quantity': 100}, {'id': 183, 'name': 'Banana', 'price': 20.0, 'quantity': 120}, {'id': 182, 'name': 'Grapes', 'price': 80.0, 'quantity': 150}]

Menu:
1. Search by ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit

Enter choice: 5
Exiting program.
```

Observations:

- Hash tables provide lightning-fast search by product ID using constant-time lookups.
- Partial substring search enables flexible name queries, helping find products by any part of their name.
- Quick Sort delivers efficient sorting by price or quantity with average $O(n \log n)$ time complexity.
- Interactive menu-driven design allows real-time product input, searching, and sorting by store staff.