

AI ASSISTED CODING LAB ASSIGNMENT – 12.1

NAME : S.NADHIYA

ROLLNO : 2403A510C6

BATCH : 05

DEPT : CSE

Task – 1

Use AI to generate a Python program that implements the Merge Sort algorithm

Prompt:

Generate a Python program that implements Merge Sort.

Write a function `merge_sort(arr)` that sorts a list in ascending order.

Include time and space complexity details inside the function docstring.

Add test cases to verify the function works correctly and print the results make sur that user should give input

Code generated:

```
searchtree.py hashtable.py graph.py * priority.py remove.py emailduplicates.py titles.py slugify_tdd.py sports
merge_sort.py > _merge
1 def merge_sort(arr):
2     """
3     Sort a list in ascending order using the Merge Sort algorithm.
4
5     Merge Sort is a divide-and-conquer algorithm that recursively splits the list
6     into halves, sorts each half, and then merges the sorted halves back together.
7
8     Parameters
9     -----
10    arr : list
11        The input list of comparable elements to sort. The input is not modified;
12        a new sorted list is returned.
13
14    Returns
15    -----
16    list
17        A new list containing the elements of 'arr' in ascending order.
18
19    Complexity
20    -----
21    - Time:  $O(n \log n)$  in the best, average, and worst cases, where  $n$  is  $\text{len}(arr)$ .
22    - Space:  $O(n)$  auxiliary space for the temporary arrays used during merging.
23
24    Characteristics
25    -----
26    - Stable: Yes (equal elements preserve their relative order).
27    - Not in-place: Produces and returns a new sorted list.
28    """
29    # Base case: empty list or single element is already sorted
30    if len(arr) <= 1:
31        return arr[:] # return a shallow copy to avoid mutating caller's list
32
33    mid = len(arr) // 2
34    left_sorted = merge_sort(arr[:mid])
35    right_sorted = merge_sort(arr[mid:])
36
37    return _merge(left_sorted, right_sorted)
38
39
40 def _merge(left, right):
41     """Merge two sorted lists into one sorted list (ascending)."""
42     i = 0
43     j = 0
44     merged = []
45     # Ctrl+L to chat, Ctrl+K to generate
46     while i < len(left) and j < len(right):
47         # Use <= to maintain stability: take from left when equal
```

```

merge_sort.py > @ _merge
40 def _merge(left, right):
46     while i < len(left) and j < len(right):
47         # Use <= to maintain stability: take from left when equal
48         if left[i] <= right[j]:
49             merged.append(left[i])
50             i += 1
51         else:
52             merged.append(right[j])
53             j += 1
54
55     # Append remaining elements (one of these will be empty)
56     if i < len(left):
57         merged.extend(left[i:])
58     if j < len(right):
59         merged.extend(right[j:])
60
61     return merged
62
63
64 if __name__ == "__main__":
65     # Interactive input from the user
66     try:
67         raw = input("Enter numbers to sort (separated by spaces or commas): ").strip()
68         if raw:
69             import re
70             tokens = [t for t in re.split(r"[\s,]+", raw) if t]
71             numbers = []
72             for t in tokens:
73                 try:
74                     # Prefer int where possible to avoid 1.0 vs 1 differences
75                     n = int(t)
76                 except ValueError:
77                     try:
78                         n = float(t)
79                     except ValueError:
80                         raise ValueError(f"Invalid number: '{t}'")
81                 numbers.append(n)
82
83             sorted_numbers = merge_sort(numbers)
84             print(f"Input: {numbers}")
85             print(f"Sorted: {sorted_numbers}")
86         else:
87             print("No input provided. Skipping interactive sort.")
88     except Exception as exc:
89         print(f"Error: {exc}")
90
91     # Test cases: (input, expected_output)

```

```

merge_sort.py > @ _merge
89     print(f"Error: {exc}")
90
91     # Test cases: (input, expected_output)
92     test_cases = [
93         ([], []),
94         ([1], [1]),
95         ([2, 1], [1, 2]),
96         ([5, 3, 8, 4, 2], [2, 3, 4, 5, 8]),
97         ([3, 3, 2, 1, 2], [1, 2, 2, 3, 3]),
98         ([-1, -3, 0, 2, -2], [-3, -2, -1, 0, 2]),
99         ([1, 2, 3, 4, 5], [1, 2, 3, 4, 5]),
100        ([5, 4, 3, 2, 1], [1, 2, 3, 4, 5]),
101        (["b", "a", "d", "c"], ["a", "b", "c", "d"]),
102    ]
103
104    total = len(test_cases)
105    passed = 0
106
107    for idx, (given, expected) in enumerate(test_cases, start=1):
108        result = merge_sort(given)
109        ok = result == expected
110        if ok:
111            passed += 1
112        print(f"Test {idx}: input={given} -> result={result} | expected={expected} | {'PASS' if ok else 'FAIL'}")
113
114    print(f"\nPassed {passed}/{total} tests.")
115

```

Output:

```
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/merge_sort.py
Enter numbers to sort (separated by spaces or commas): 1,2,9,8,7,4,2
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/merge_sort.py
Enter numbers to sort (separated by spaces or commas): 1,2,9,8,7,4,2
Input: [1, 2, 9, 8, 7, 4, 2]
Sorted: [1, 2, 2, 4, 7, 8, 9]
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/merge_sort.py
Enter numbers to sort (separated by spaces or commas): 1,2,9,8,7,4,2
Enter numbers to sort (separated by spaces or commas): 1,2,9,8,7,4,2
Input: [1, 2, 9, 8, 7, 4, 2]
Sorted: [1, 2, 2, 4, 7, 8, 9]
Sorted: [1, 2, 2, 4, 7, 8, 9]
Test 1: input=[] -> result=[] | expected=[] | PASS
Test 2: input=[1] -> result=[1] | expected=[1] | PASS
Test 3: input=[2, 1] -> result=[1, 2] | expected=[1, 2] | PASS
Test 4: input=[5, 3, 8, 4, 2] -> result=[2, 3, 4, 5, 8] | expected=[2, 3, 4, 5, 8] | PASS
Test 5: input=[3, 3, 2, 1, 2] -> result=[1, 2, 2, 3, 3] | expected=[1, 2, 2, 3, 3] | PASS
Test 6: input=[-1, -3, 0, 2, -2] -> result=[-3, -2, -1, 0, 2] | expected=[-3, -2, -1, 0, 2] | PASS
Test 7: input=[1, 2, 3, 4, 5] -> result=[1, 2, 3, 4, 5] | expected=[1, 2, 3, 4, 5] | PASS
Test 6: input=[-1, -3, 0, 2, -2] -> result=[-3, -2, -1, 0, 2] | expected=[-3, -2, -1, 0, 2] | PASS
Test 7: input=[1, 2, 3, 4, 5] -> result=[1, 2, 3, 4, 5] | expected=[1, 2, 3, 4, 5] | PASS
Test 7: input=[1, 2, 3, 4, 5] -> result=[1, 2, 3, 4, 5] | expected=[1, 2, 3, 4, 5] | PASS
Test 8: input=[5, 4, 3, 2, 1] -> result=[1, 2, 3, 4, 5] | expected=[1, 2, 3, 4, 5] | PASS
Test 9: input=['b', 'a', 'd', 'c'] -> result=['a', 'b', 'c', 'd'] | expected=['a', 'b', 'c', 'd'] | PASS

Passed 9/9 tests.
PS C:\Users\nadhi\OneDrive\Desktop\AI> 
```

Observation:

The program correctly implements **Merge Sort** with detailed documentation, interactive input handling, and comprehensive test cases.

It maintains **stability**, **avoids mutating the input list**, and **passes all tests**, confirming its correctness.

Task – 02:

Use AI to create a binary search function that finds a target element in a sorted list

Prompt:

Generate a Python code that implements binary search on a sorted list.

Write a function `binary_search(arr, target)` that returns the index of the target or -1 if not found.

Add a detailed docstring with best, average, and worst-case time complexities and space complexity.

Include user input handling and multiple test cases to verify correctness

Code generated:

```
table.py  graph.py  priority.py  remove.py  emailduplicates.py  titles.py  slugify_tdd.py  sports.py  binary.py
binary_search.py > _run_tests
1  from typing import List
2
3
4  def binary_search(arr: List[int], target: int) -> int:
5      """
6      Perform binary search on a sorted list to find the index of a target value.
7
8      The input list must be sorted in non-decreasing (ascending) order. If the
9      target appears multiple times, any valid index for the target may be
10     returned. If the target does not exist in the list, the function returns -1.
11
12     Parameters
13     -----
14     arr : List[int]
15         Sorted list of integers (ascending) to search within.
16     target : int
17         The integer value to search for.
18
19     Returns
20     -----
21     int
22         The index of the target in the list if found; otherwise -1.
23
24     Time Complexity
25     -----
26     - Best case: O(1) – when the middle element matches the target on the first check.
27     - Average case: O(log n) – the search interval halves each iteration.
28     - Worst case: O(log n) – when the element is absent or found after log n steps.
29
30     Space Complexity
31     -----
32     - O(1) – iterative implementation uses constant extra space.
33     """
34     left = 0
35     right = len(arr) - 1
36
37     while left <= right:
38         mid = left + (right - left) // 2
39         mid_value = arr[mid]
40
41         if mid_value == target:
```

```
table.py graph.py priority.py remove.py emailduplicates.py titles.py slugify_tdd.py sports.py binary.p
binary_search.py > _run_tests
4
40
41     if mid_value == target:
42         return mid
43     if mid_value < target:
44         left = mid + 1
45     else:
46         right = mid - 1
47
48     return -1
49
50
51 def _run_tests() -> None:
52     """Execute a series of basic tests to verify the implementation."""
53     tests = [
54         # (array, target, expected_index or None to indicate not found)
55         ([1, 10, -1],
56         ([1], 1, 0),
57         ([1], 2, -1),
58         ([1, 3, 5, 7, 9], 1, 0),
59         ([1, 3, 5, 7, 9], 9, 4),
60         ([1, 3, 5, 7, 9], 5, 2),
61         ([1, 2, 2, 2, 3], 2, 1), # any index 1..3 is acceptable; we validate value
62         (list(range(0, 1000, 2)), 500, 250),
63         (list(range(0, 1000, 2)), 501, -1),
64     ]
65
66     total = len(tests)
67     passed = 0
68
69     for idx, (arr, target, expected) in enumerate(tests, start=1):
70         result = binary_search(arr, target)
71         ok: bool
72         if expected == -1:
73             ok = result == -1
74         else:
75             ok = 0 <= result < len(arr) and arr[result] == target
76
77         if ok:
78             passed += 1
79     else:
80         print(f"Passed {passed} out of {total} tests")
```

```

binary_search.py > _run_tests
85 def _parse_numbers(input_str: str) -> List[int]:
86     cleaned = input_str.replace(",", " ")
87     tokens = [tok for tok in cleaned.split() if tok]
88     numbers: List[int] = []
89     for tok in tokens:
90         try:
91             numbers.append(int(tok))
92         except ValueError:
93             raise ValueError(f"Invalid integer token: '{tok}'") from None
94     return numbers
95
96
97
98 def _is_sorted_non_decreasing(values: List[int]) -> bool:
99     return all(values[i] <= values[i + 1] for i in range(len(values) - 1))
100
101
102 if __name__ == "__main__":
103     # Run built-in tests first
104     _run_tests()
105
106     print("\nBinary Search Interactive Demo\n" + "-" * 34)
107     try:
108         arr_input = input(
109             "Enter a sorted list of integers (space/comma separated): "
110         ).strip()
111         arr = _parse_numbers(arr_input)
112
113         if not _is_sorted_non_decreasing(arr):
114             print("Input list is not sorted in ascending order. Sorting it for you...")
115             arr.sort()
116
117         target_input = input("Enter the target integer to search for: ").strip()
118         target = int(target_input)
119
120         index = binary_search(arr, target)
121         if index != -1:
122             print(f"Target {target} found at index {index}.")
123         else:
124             print(f"Target {target} not found. Returned index: -1")
125     except Exception as exc:
126         print(f"Error: {exc}")

```

Output:

```

Passed 9/9 tests.
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/binary_search.py
/nadhi/OneDrive/Desktop/AI/binary_search.py
Tests passed: 9/9

pdf Binary Search Interactive Demo
pdf Tests passed: 9/9

4... Tests passed: 9/9
.... Tests passed: 9/9
.... Binary Search Interactive Demo
-----
Enter a sorted list of integers (space/comma separated): 1,2,3,4,5
Enter the target integer to search for: 2
Target 2 found at index 1.

```

Observation:

The program correctly implements an **iterative binary search** with detailed docstrings, input parsing, and auto-sorting for unsorted input.

It includes a robust **test suite** and interactive demo, ensuring both correctness and user-friendly behavior.

Task – 03:

Use AI to suggest the most efficient search and sort

algorithms for this use case.

Prompt:

Generate a python code to manage a retail store inventory with products having ID, name, price, and quantity.

Implement efficient search functions: dictionary lookup for ID and binary search for name.

Implement sorting functions using Python's built-in sort (Timsort) by price and quantity.

Include user input for products and interactive menu for searching and sorting operations.

Code generated:

```
Inventory_manager.py > Inventory
1  from dataclasses import dataclass
2  from typing import List, Dict, Optional
3  @dataclass
4  class Product:
5      product_id: str
6      name: str
7      price: float
8      quantity: int
9  class Inventory:
10     def __init__(self) -> None:
11         self.products_by_id: Dict[str, Product] = {}
12         self.products: List[Product] = []
13
14     # ----- Core CRUD -----
15     def add_product(self, product: Product) -> bool:
16         if product.product_id in self.products_by_id:
17             return False
18         self.products_by_id[product.product_id] = product
19         self.products.append(product)
20         return True
21
22     def list_products(self) -> List[Product]:
23         return list(self.products)
24
25     # ----- Efficient Lookups/Search -----
26     def get_by_id(self, product_id: str) -> Optional[Product]:
27         return self.products_by_id.get(product_id)
28
29     def binary_search_by_name(self, target_name: str) -> Optional[Product]:
30         # Ensure list is sorted by name for binary search
31         products_sorted = sorted(self.products, key=lambda p: p.name.lower())
32         left = 0
33         right = len(products_sorted) - 1
34         target = target_name.lower()
35         while left <= right:
36             mid = (left + right) // 2
37             mid_name = products_sorted[mid].name.lower()
38             if mid_name == target:
39                 return products_sorted[mid]
40             if mid_name < target:
41                 left = mid + 1
42             else:
43                 right = mid - 1
44         return None
45
46     # ----- Sorting -----
47     def sort_by_price(self, reverse: bool = False) -> List[Product]:
48         return sorted(self.products, key=lambda p: p.price, reverse=reverse)
49
50     def sort_by_quantity(self, reverse: bool = False) -> List[Product]:
51         return sorted(self.products, key=lambda p: p.quantity, reverse=reverse)
52
53     # ----- CLI -----
54     def prompt_product() -> Optional[Product]:
55         try:
56             product_id = input("Enter product ID: ").strip()
57             name = input("Enter product name: ").strip()
58             price = float(input("Enter price: ").strip())
59             quantity = int(input("Enter quantity: ").strip())
60             return Product(product_id=product_id, name=name, price=price, quantity=quantity)
61         except ValueError:
62             pass
63
64 Problems Output Debug Console Terminal Ports
```



```
◆ searchtree.py ◆ hashtable.py ◆ graph.py ◆ priority.py ◆ remove.py ◆ emailuplicates.py ◆ titles.py ◆ slugify_todo.py ◆ sports.py ◆ binary.py
◆ inventory_manager.py > Inventory
55 def prompt_product() -> Optional[Product]:
56     return Product(product_id=product_id, name=name, price=price, quantity=quantity)
57 except ValueError:
58     print("Invalid number for price or quantity. Please try again.")
59     return None
60
61
62 def print_products(products: List[Product]) -> None:
63     if not products:
64         print("No products to display.")
65         return
66     print("\nID\tName\tPrice\tQuantity")
67     print("-" * 40)
68     for p in products:
69         print(f"{p.product_id}\t{p.name}\t{p.price:.2f}\t{p.quantity}")
70     print()
71
72
73 def interactive_menu() -> None:
74     inventory = Inventory()
75     while True:
76         print("\nRetail Store Inventory Manager")
77         print("1. Add product")
78         print("2. Search by ID (dictionary lookup)")
79         print("3. Search by Name (binary search)")
80         print("4. Sort by Price (ascending)")
81         print("5. Sort by Price (descending)")
82         print("6. Sort by Quantity (ascending)")
83         print("7. Sort by Quantity (descending)")
84         print("8. List all products (insertion order)")
85         print("9. Exit")
86
87         choice = input("Choose an option (1-9): ").strip()
88
89         if choice == "1":
90             product = prompt_product()
91             if product is None:
92                 continue
93             if inventory.add_product(product):
94                 print("Product added.")
95             else:
96                 print("Product ID already exists. Use a unique ID.")
97
98         elif choice == "2":
99             pid = input("Enter product ID to search: ").strip()
100             result = inventory.get_by_id(pid)
101             if result:
102                 print_products([result])
103             else:
104                 print("Product not found.")
105
106         elif choice == "3":
107             name = input("Enter product name to search: ").strip()
108             result = inventory.binary_search_by_name(name)
109             if result:
110                 print_products([result])
111             else:
112                 print("Product not found.")
113
114         elif choice == "4":
115             print_products(inventory.sort_by_price(reverse=False))
116
117
118
119
120
121
```

```
inventory_manager.py > Inventory
78 def interactive_menu() -> None:
115     print_products([result])
116     else:
117         print("Product not found.")
118
119     elif choice == "4":
120         print_products(inventory.sort_by_price(reverse=False))
121
122     elif choice == "5":
123         print_products(inventory.sort_by_price(reverse=True))
124
125     elif choice == "6":
126         print_products(inventory.sort_by_quantity(reverse=False))
127
128     elif choice == "7":
129         print_products(inventory.sort_by_quantity(reverse=True))
130
131     elif choice == "8":
132         print_products(inventory.list_products())
133
134     elif choice == "9":
135         print("Goodbye!")
136         break
137
138     else:
139         print("Invalid choice. Please select a number between 1 and 9.")
140
141
142 if __name__ == "__main__":
143     interactive_menu()
144
145
146
```

Output:

```
Problems Output Debug Console Terminal Ports
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/inventory_manager.py

Retail Store Inventory Manager
1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit
Choose an option (1-9): 1
Enter product ID: 201
Enter product name: Apple
Enter price: 100
Enter quantity: 3
Product added.

Retail Store Inventory Manager
1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit
Choose an option (1-9): 1
Enter product ID: 202
Enter product name: Banana
Enter price: 50
Enter quantity: 2
Product added.

Retail Store Inventory Manager
1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)

Ctrl+K to generate a command
```

1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit

Choose an option (1-9): 3

Enter product name to search: Apple

ID	Name	Price	Quantity
201	Apple	100.00	3

Retail Store Inventory Manager

1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit

Choose an option (1-9): 4

ID	Name	Price	Quantity
202	Banana	50.00	2
201	Apple	100.00	3

Retail Store Inventory Manager

1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit

Ctrl+K to generate a command

```
Problems  Output  Debug Console  Terminal  Ports

5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit
Choose an option (1-9): 7

ID      Name      Price      Quantity
-----
201     Apple     100.00     3
202     Banana    50.00      2

Retail Store Inventory Manager
1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit
Choose an option (1-9): 8

ID      Name      Price      Quantity
-----
201     Apple     100.00     3
202     Banana    50.00      2

Retail Store Inventory Manager
1. Add product
2. Search by ID (dictionary lookup)
3. Search by Name (binary search)
4. Sort by Price (ascending)
5. Sort by Price (descending)
6. Sort by Quantity (ascending)
7. Sort by Quantity (descending)
8. List all products (insertion order)
9. Exit
Choose an option (1-9): 9
Goodbye!
PS C:\Users\nadhi\OneDrive\Desktop\AI>
```

Observation:

This program provides a **robust inventory management system** using a combination of dictionary lookups for fast ID searches and binary search for names.

It supports **adding products, listing, sorting by price/quantity, and interactive CLI operations**, ensuring user-friendly input handling.

The use of **dataclasses and efficient algorithms** ensures both code clarity and performance for large datasets