SCHOOLOFCOMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENTOFCOMPUTER SCIENCE ENGINEERING	
ProgramName:B. Tech		**AssignmentType: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
DateandDay of Assignment	Week6 - Monday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:12.1(Presentassignmentnumber)/**24**(Totalnumberofassignments)			

Q.No.	Question	*ExpectedTime to complete*
1	**Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms** **Lab Objectives:** • Apply AI-assisted programming to implement and optimize sorting and searching algorithms. • Compare different algorithms in terms of efficiency and use	Week6 - Monday

cases.

- Understand how AI tools can suggest optimized code and complexity improvements.

Task Description #1 (Sorting – Merge Sort Implementation)

- Task: Use AI to generate a Python program that implements the Merge Sort algorithm.
- Instructions:
  - Prompt AI to create a function merge_sort(arr) that sorts a list in ascending order.
  - Ask AI to include time complexity and space complexity in the function docstring.
  - Verify the generated code with test cases.
- Expected Output:
  - A functional Python script implementing Merge Sort with proper documentation.

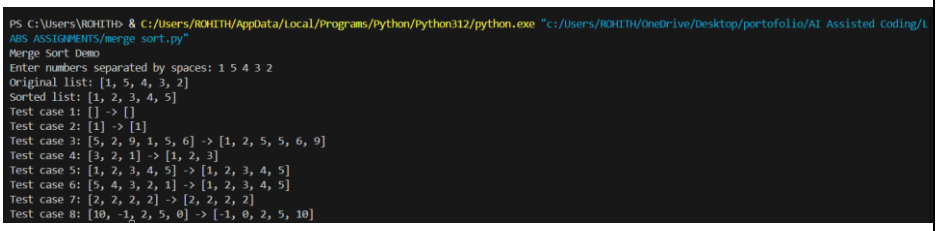
CODE:

```
defmerge_sort(arr):
    """
    Sorts a list in ascending order using the Merge Sort
algorithm.

    Time Complexity: O(n log n), where n is the number of
elements in the list.
    Space Complexity: O(n), due to the use of temporary
arrays during merging.

    Args:
        arr (list): The list of elements to be sorted.

    Returns:
        list: A new sorted list in ascending order.
    """
    iflen(arr) <=1:
        returnarr

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

    returnmerge(left, right)

defmerge(left, right):
```

```
    result= []
    i=j=0

    # Merge the two sorted lists
    whilei<len(left) andj<len(right):
        ifleft[i] <=right[j]:
            result.append(left[i])
            i+=1
        else:
            result.append(right[j])
            j+=1

    # Append remaining elements
    result.extend(left[i:])
    result.extend(right[j:])
    returnresult


defmain():
    """
    Main function to demonstrate merge sort.
    Prompts user for input, sorts the list, and displays the
result.
    """
    print("Merge Sort Demo")
    try:
        user_input=input("Enter numbers separated by spaces:
")
        arr= [int(x) forxinuser_input.strip().split()]
    exceptValueError:
        print("Invalid input. Please enter integers only.")
        return

    print("Original list:", arr)
    sorted_arr=merge_sort(arr)
    print("Sorted list:", sorted_arr)


if__name__=="__main__":
    main()

# Test cases
if__name__=="__main__":
    test_cases= [
        [],
```

```
        [1],
        [5, 2, 9, 1, 5, 6],
        [3, 2, 1],
        [1, 2, 3, 4, 5],
        [5, 4, 3, 2, 1],
        [2, 2, 2, 2],
        [10, -1, 2, 5, 0]
    ]

    fori, testinenumerate(test_cases):
        print(f"Test case {i+1}: {test} -
>{merge_sort(test)}")
```

OUTPUT:



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

- Task: Use AI to create a binary search function that finds a target element in a sorted list.
- Instructions:
  - Prompt AI to create a function binary_search(arr, target) returning the index of the target or -1 if not found.
  - Include docstrings explaining best, average, and worst-case complexities.
  - Test with various inputs.
- Expected Output:
  - Python code implementing binary search with AI-generated comments and docstrings.

CODE:

```
defbinary_search(arr, target):
    """
    Performs binary search to locate the index of 'target' in
a sorted list 'arr'.

    Best Case Complexity: O(1)
        - Target is found at the middle index on the first
comparison.
```

```
    Average Case Complexity: O(log n)
        - Each iteration halves the search space.
    Worst Case Complexity: O(log n)
        - Target is not present or found after all possible
divisions.

    Args:
        arr (list): Sorted list of elements to search.
        target: Element to find.

    Returns:
        int: Index of target if found, else -1.
    """
    left, right=0, len(arr) -1

    whileleft<=right:
        mid= (left+right) //2  # AI optimization: Efficient
midpoint calculation
        ifarr[mid] ==target:
            returnmid  # Target found
        elifarr[mid] <target:
            left=mid+1  # Search right half
        else:
            right=mid-1  # Search left half

    return-1  # Target not found

if__name__=="__main__":
    print("Binary Search Demo")
    try:
        arr_input=input("Enter sorted numbers separated by
spaces: ")
        arr= [int(x) forxinarr_input.strip().split()]
        target=int(input("Enter the target value to search
for: "))
    exceptValueError:
        print("Invalid input. Please enter integers only.")
        exit(1)

    index=binary_search(arr, target)
    ifindex!=-1:
        print(f"Target{target} found at index {index}.")
    else:
        print(f"Target{target} not found in the list.")
```

OUTPUT:

Task Description #3 (Real-Time Application – Inventory Management 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.
- Task:
    - Use AI to suggest the most efficient search and sort algorithms for this use case.
    - Implement the recommended algorithms in Python.
    - Justify the choice based on dataset size, update frequency, and performance requirements.
- Expected Output:
    - A table mapping operation → recommended algorithm → justification.
    - Working Python functions for searching and sorting the inventory.

CODE:

```
classProduct:
    def__init__(self, product_id, name, price, quantity):
        self.product_id=product_id
        self.name=name
        self.price=price
        self.quantity=quantity

    def__repr__(self):
        return (f"Product(ID={self.product_id},
Name='{self.name}', "
                f"Price={self.price},
Quantity={self.quantity})")

classInventory:
    def__init__(self, products):
        # Hash maps for fast search
```

```
        self.products_by_id= {p.product_id: pforpinproducts}
        self.products_by_name= {p.name: pforpinproducts}
        self.products=products

    defsearch_by_id(self, product_id):
        """O(1) average time using hash map."""
        returnself.products_by_id.get(product_id, None)

    defsearch_by_name(self, name):
        """O(1) average time using hash map."""
        returnself.products_by_name.get(name, None)

    defsort_by_price(self, reverse=False):
        """O(n log n) using Timsort."""
        returnsorted(self.products, key=lambdap: p.price,
reverse=reverse)

    defsort_by_quantity(self, reverse=False):
        """O(n log n) using Timsort."""
        returnsorted(self.products, key=lambdap: p.quantity,
reverse=reverse)

# --- Example Usage ---
if__name__=="__main__":
    print("Inventory Management System Demo")
    products= []
    try:
        n=int(input("Enter number of products: "))
        foriinrange(n):
            print(f"Enter details for product {i+1}:")
            product_id=int(input("  Product ID: "))
            name=input("  Name: ")
            price=float(input("  Price: "))
            quantity=int(input("  Quantity: "))
            products.append(Product(product_id, name, price,
quantity))
    exceptValueError:
        print("Invalid input. Please enter correct data
types.")
        exit(1)

    inventory=Inventory(products)

    whileTrue:
        print("\nChoose an operation:")
```

```
        print("1. Search by Product ID")
        print("2. Search by Name")
        print("3. Sort by Price")
        print("4. Sort by Quantity")
        print("5. Exit")
        choice=input("Enter choice (1-5): ")

        ifchoice=="1":
            pid=int(input("Enter Product ID to search: "))
            result=inventory.search_by_id(pid)
            print("Result:", result)
        elifchoice=="2":
            name=input("Enter Product Name to search: ")
            result=inventory.search_by_name(name)
            print("Result:", result)
        elifchoice=="3":
            sorted_products=inventory.sort_by_price()
            print("Products sorted by price:")
            forpinsorted_products:
                print(p)
        elifchoice=="4":

sorted_products=inventory.sort_by_quantity(reverse=True)
            print("Products sorted by quantity
(descending):")
            forpinsorted_products:
                print(p)
        elifchoice=="5":
            print("Exiting.")
            break
        else:
            print("Invalid choice. Please try again.")
```

OUTPUT:

1. A table mapping operation → recommended algorithm → justification.

Operation → Recommended Algorithm → Justification

Operation	Recommended Algorithm	Justification
Search by Product ID	Hash Map (Dictionary Lookup)	O(1) average time; ideal for large datasets and frequent lookups.
Search by Name	Hash Map (Dictionary Lookup)	O(1) average time; fast, scalable, and supports frequent updates.
Sort by Price/Quantity	Timsort (Python's sorted)	O(n log n) worst-case; stable, efficient for large lists, and built into Python.

- **Hash maps** (Python dictionaries) provide constant-time search and are optimal for large, frequently updated datasets.
- **Timsort** is Python's built-in sorting algorithm, optimized for real-world data and large lists.

2.Working Python functions for searching and sorting the inventory.

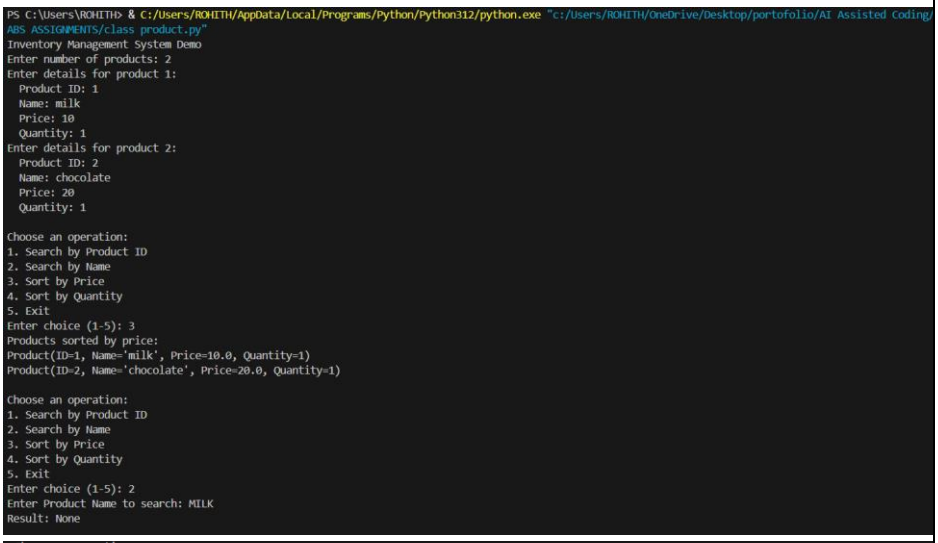
CODE:

```
classProduct:
    def__init__(self, product_id, name, price, quantity):
        self.product_id=product_id
        self.name=name
        self.price=price
        self.quantity=quantity

    def__repr__(self):
        return (f"Product(ID={self.product_id},
Name='{self.name}', "
                f"Price={self.price},
Quantity={self.quantity})")
```

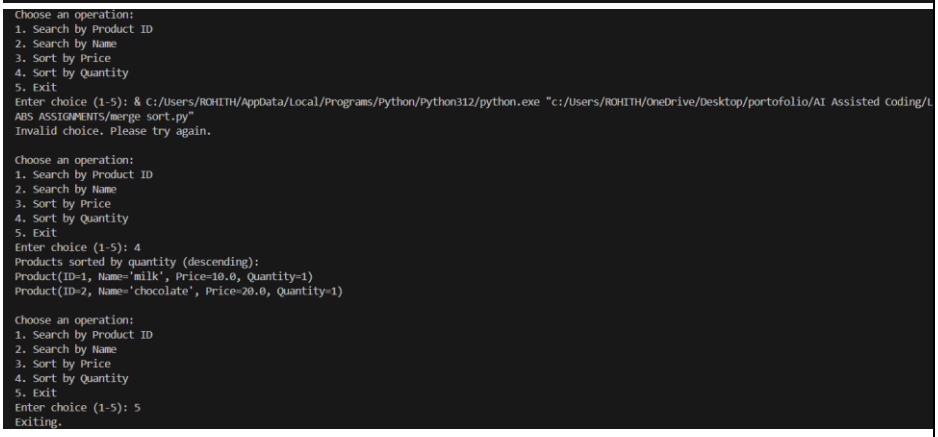
```
classInventory:
    def__init__(self, products):
        # Hash maps for fast search
        self.products_by_id= {p.product_id: pforpinproducts}
        self.products_by_name= {p.name: pforpinproducts}
        self.products=products

    defsearch_by_id(self, product_id):
        """O(1) average time using hash map."""
        returnself.products_by_id.get(product_id, None)

    defsearch_by_name(self, name):
        """O(1) average time using hash map."""
        returnself.products_by_name.get(name, None)

    defsort_by_price(self, reverse=False):
        """O(n log n) using Timsort."""
        returnsorted(self.products, key=lambdap: p.price,
reverse=reverse)

    defsort_by_quantity(self, reverse=False):
        """O(n log n) using Timsort."""
        returnsorted(self.products, key=lambdap: p.quantity,
reverse=reverse)

# --- Example Usage ---
if__name__=="__main__":
    print("Inventory Management System Demo")
    products= []
    try:
        n=int(input("Enter number of products: "))
        foriinrange(n):
            print(f"Enter details for product {i+1}:")
            product_id=int(input("  Product ID: "))
            name=input("  Name: ")
            price=float(input("  Price: "))
            quantity=int(input("  Quantity: "))
            products.append(Product(product_id, name, price,
quantity))
    exceptValueError:
        print("Invalid input. Please enter correct data
types.")
        exit(1)
```

```
    inventory=Inventory(products)

    whileTrue:
        print("\nChoose an operation:")
        print("1. Search by Product ID")
        print("2. Search by Name")
        print("3. Sort by Price")
        print("4. Sort by Quantity")
        print("5. Exit")
        choice=input("Enter choice (1-5): ")

        ifchoice=="1":
            pid=int(input("Enter Product ID to search: "))
            result=inventory.search_by_id(pid)
            print("Result:", result)
        elifchoice=="2":
            name=input("Enter Product Name to search: ")
            result=inventory.search_by_name(name)
            print("Result:", result)
        elifchoice=="3":
            sorted_products=inventory.sort_by_price()
            print("Products sorted by price:")
            forpinsorted_products:
                print(p)
        elifchoice=="4":

sorted_products=inventory.sort_by_quantity(reverse=True)
            print("Products sorted by quantity
(descending):")
            forpinsorted_products:
                print(p)
        elifchoice=="5":
            print("Exiting.")
            break
        else:
            print("Invalid choice. Please try again.")
```

OUTPUT:



```
PS C:\Users\ROHITH> & C:/Users/ROHITH/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ROHITH/OneDrive/Desktop/portofolio/AI Assisted Coding/
ABS ASSIGNMENTS/class product.py"
Inventory Management System Demo
Enter number of products: 2
Enter details for product 1:
  Product ID: 1
  Name: milk
  Price: 10
  Quantity: 1
Enter details for product 2:
  Product ID: 2
  Name: chocolate
  Price: 20
  Quantity: 1

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): 3
Products sorted by price:
Product(ID=1, Name='milk', Price=10.0, Quantity=1)
Product(ID=2, Name='chocolate', Price=20.0, Quantity=1)

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): 2
Enter Product Name to search: MILK
Result: None
```



```
Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): & C:/Users/ROHITH/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ROHITH/OneDrive/Desktop/portofolio/AI Assisted Coding/
ABS ASSIGNMENTS/merge sort.py"
Invalid choice. Please try again.

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): 4
Products sorted by quantity (descending):
Product(ID=1, Name='milk', Price=10.0, Quantity=1)
Product(ID=2, Name='chocolate', Price=20.0, Quantity=1)

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): 5
Exiting.
```

Let me know if you want this saved to a file or need further customization!

✅ Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.