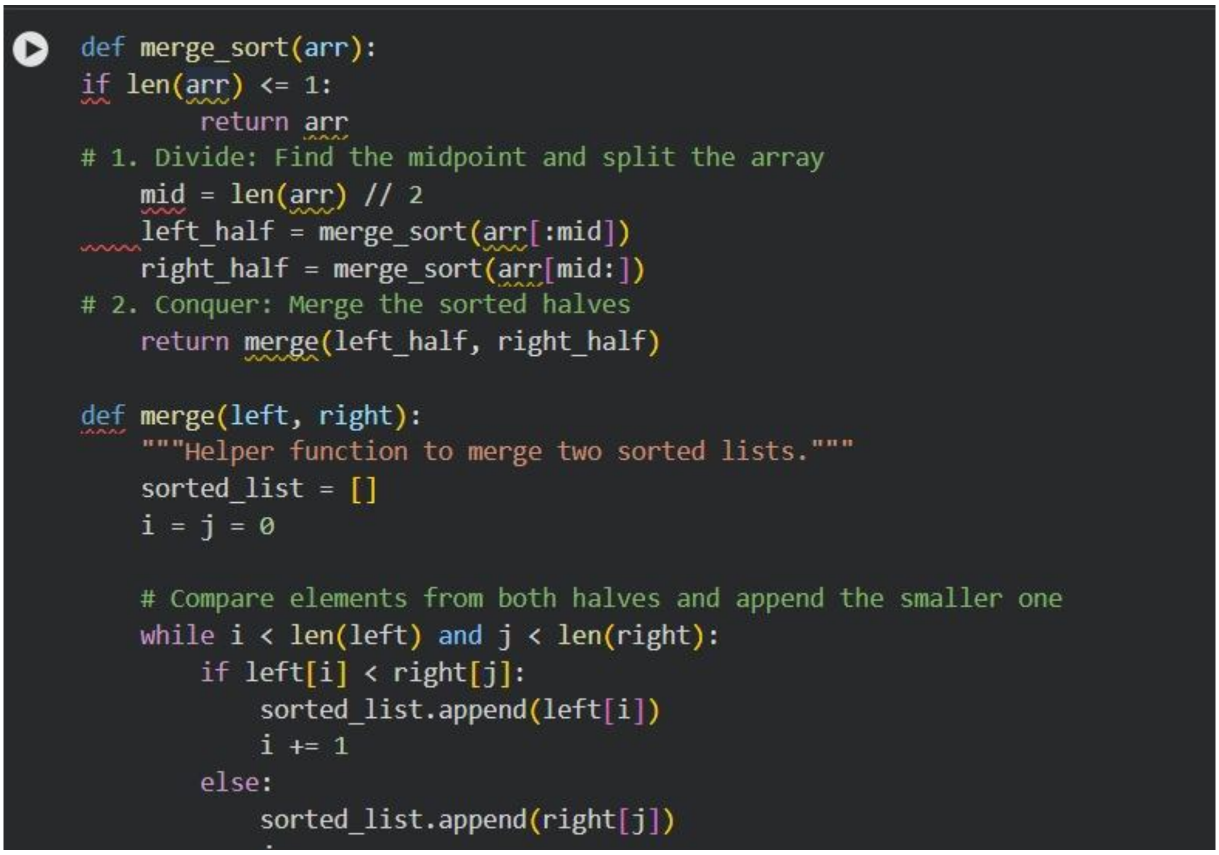
Assingment-12.1

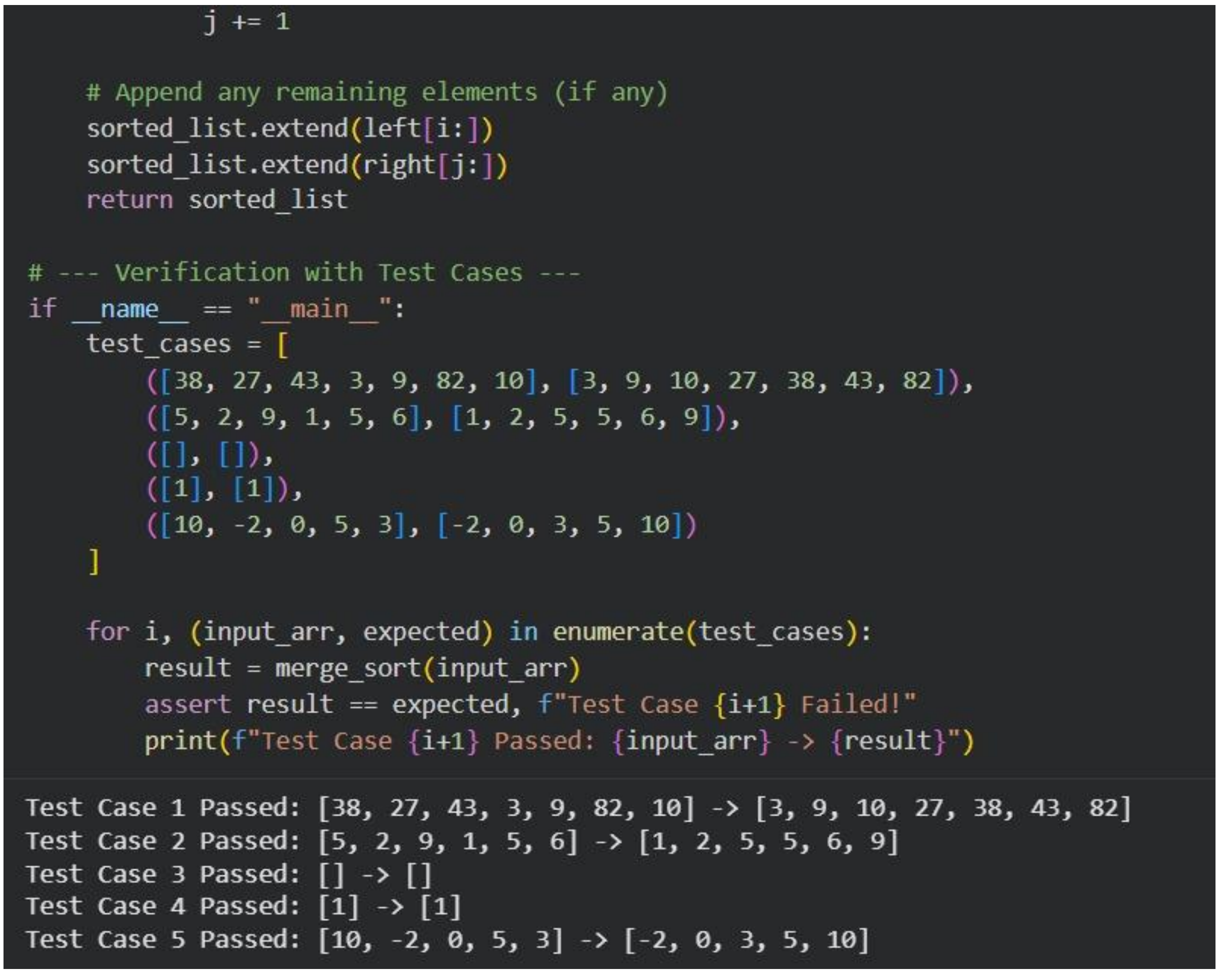
Name: Y.Poojitha
Hall Ticket no: 2303A51499
Batch:08

Lab: Algorithms with AI Assistance-Sorting,Searching,optimizing Algorithms

Task 1: Sorting-Merge sort Implementation



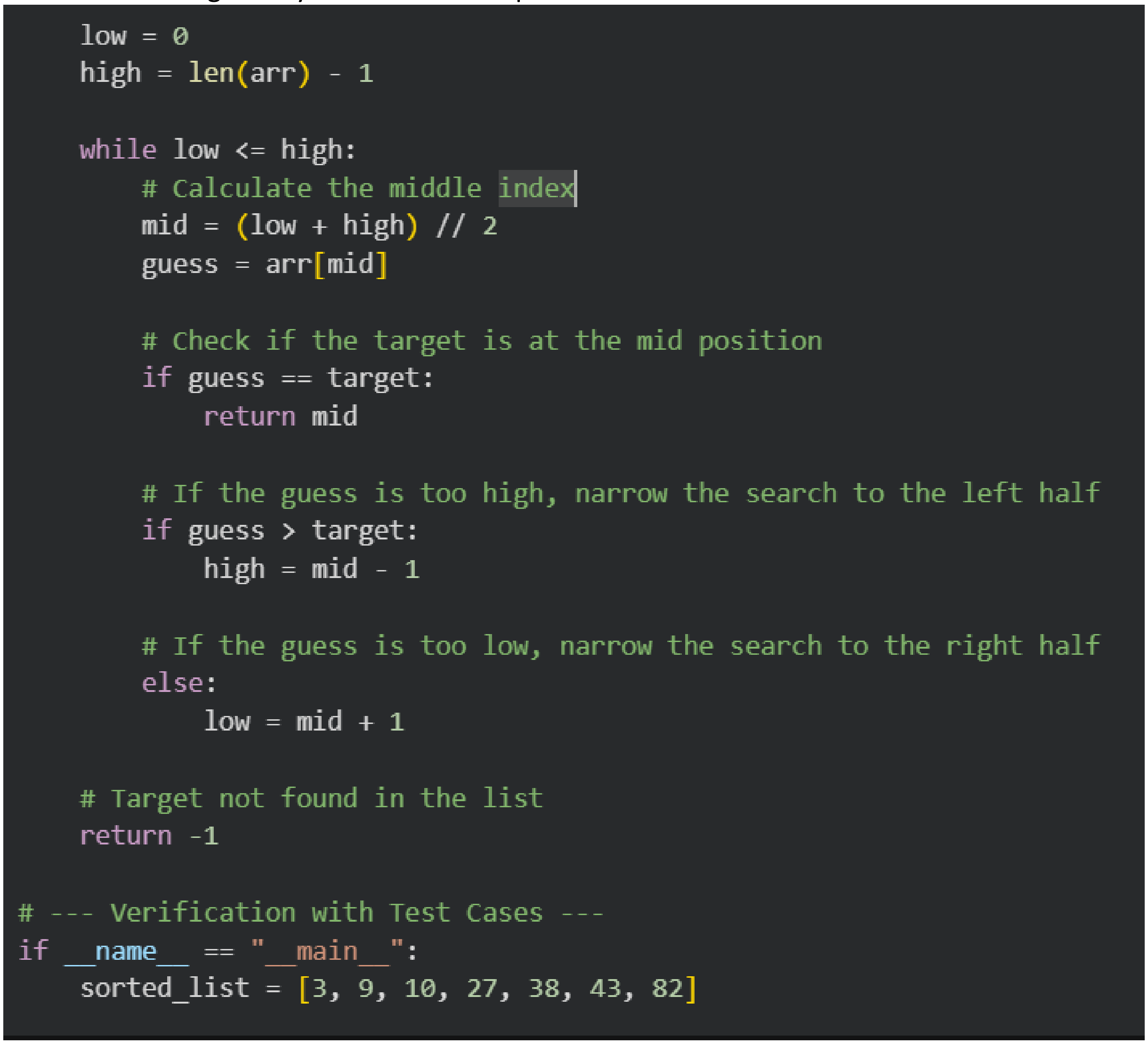
```
def merge_sort(arr):
if len(arr) <= 1:
        return arr
# 1. Divide: Find the midpoint and split the array
    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])
# 2. Conquer: Merge the sorted halves
    return merge(left_half, right_half)

def merge(left, right):
    """Helper function to merge two sorted lists."""
    sorted_list = []
    i = j = 0

    # Compare elements from both halves and append the smaller one
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_list.append(left[i])
            i += 1
        else:
            sorted_list.append(right[j])
```



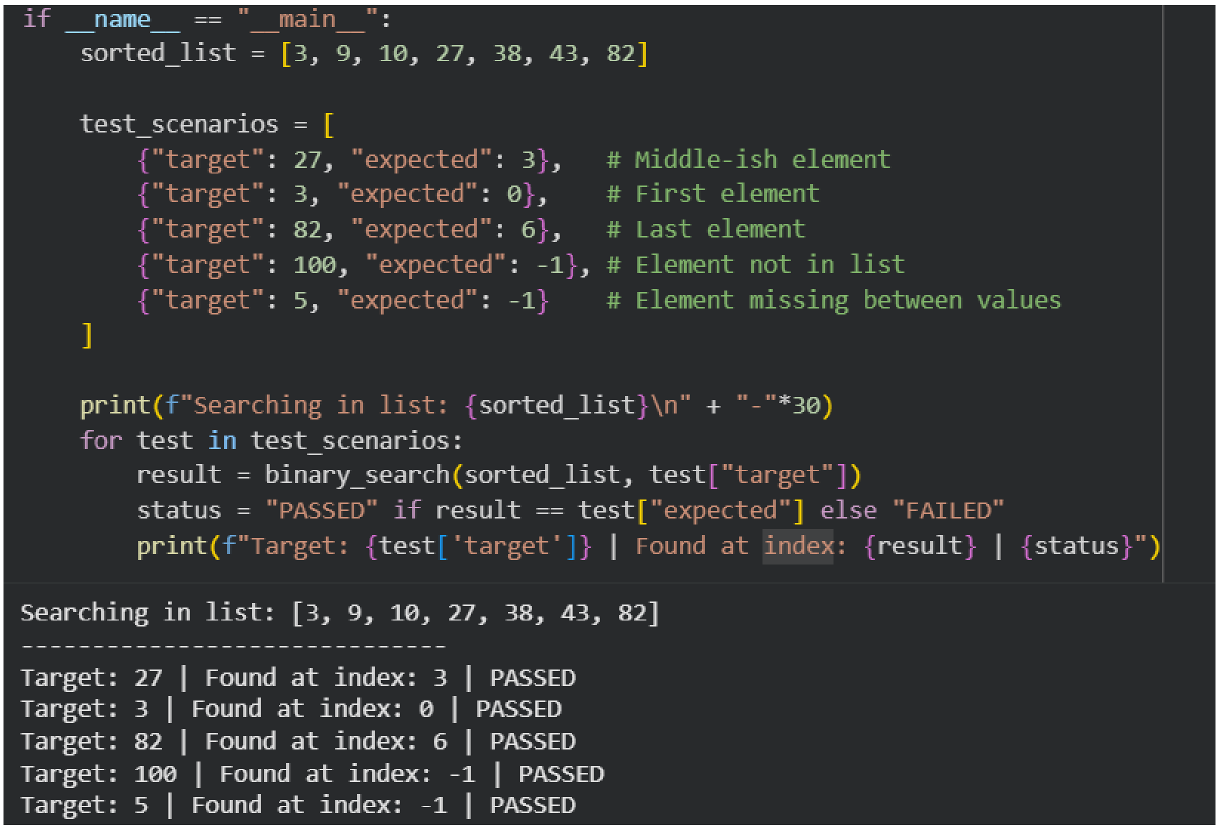
```python
            j += 1

    # Append any remaining elements (if any)
    sorted_list.extend(left[i:])
    sorted_list.extend(right[j:])
    return sorted_list

# --- Verification with Test Cases ---
if __name__ == "__main__":
    test_cases = [
        ([38, 27, 43, 3, 9, 82, 10], [3, 9, 10, 27, 38, 43, 82]),
        ([5, 2, 9, 1, 5, 6], [1, 2, 5, 5, 6, 9]),
        ([], []),
        ([1], [1]),
        ([10, -2, 0, 5, 3], [-2, 0, 3, 5, 10])
    ]

    for i, (input_arr, expected) in enumerate(test_cases):
        result = merge_sort(input_arr)
        assert result == expected, f"Test Case {i+1} Failed!"
        print(f"Test Case {i+1} Passed: {input_arr} -> {result}")
```

```
Test Case 1 Passed: [38, 27, 43, 3, 9, 82, 10] -> [3, 9, 10, 27, 38, 43, 82]
Test Case 2 Passed: [5, 2, 9, 1, 5, 6] -> [1, 2, 5, 5, 6, 9]
Test Case 3 Passed: [] -> []
Test Case 4 Passed: [1] -> [1]
Test Case 5 Passed: [10, -2, 0, 5, 3] -> [-2, 0, 3, 5, 10]
```

Task 2: Searching-Binary search with AI Optimization



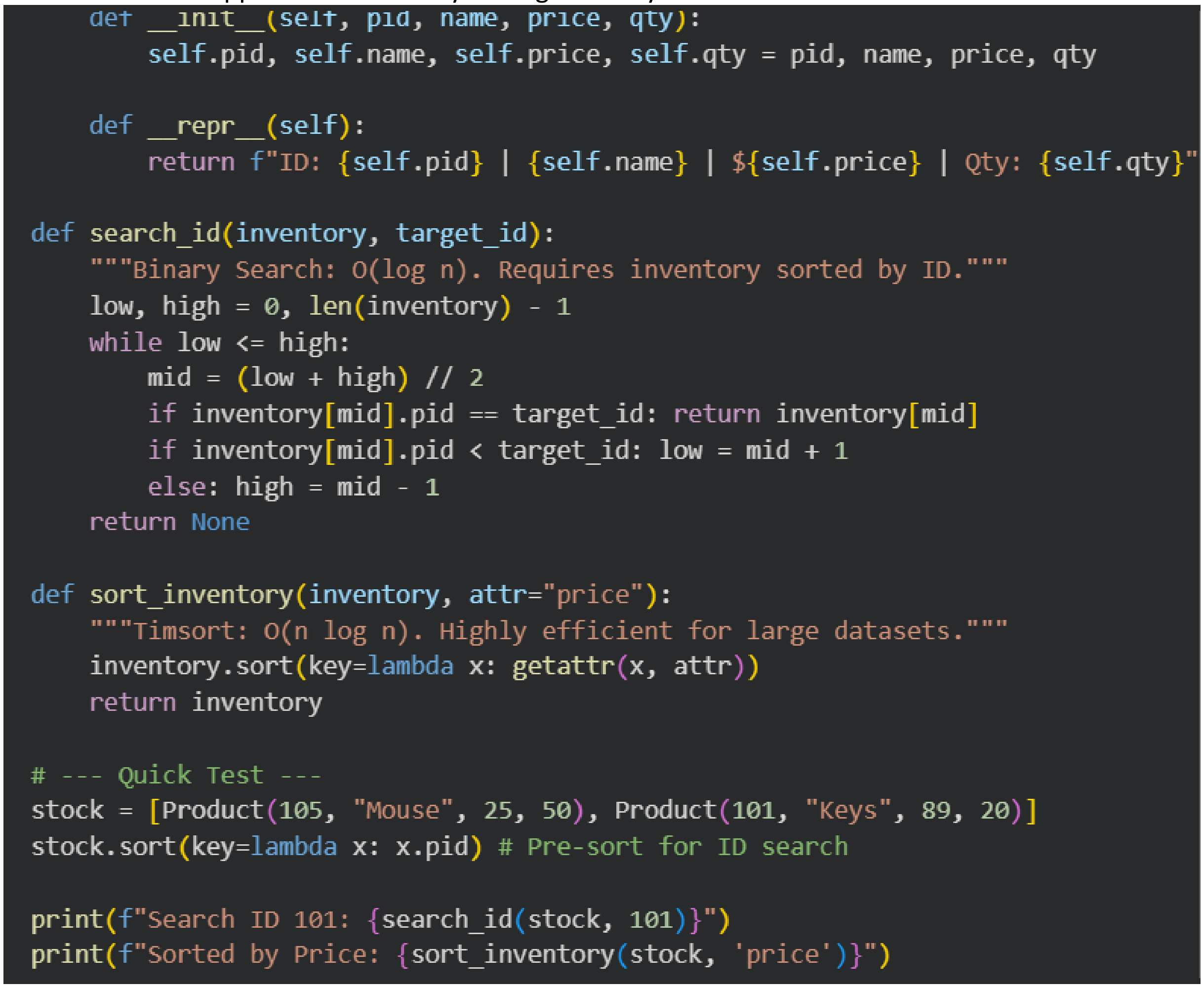
```
    low = 0
    high = len(arr) - 1

    while low <= high:
        # Calculate the middle index
        mid = (low + high) // 2
        guess = arr[mid]

        # Check if the target is at the mid position
        if guess == target:
            return mid

        # If the guess is too high, narrow the search to the left half
        if guess > target:
            high = mid - 1

        # If the guess is too low, narrow the search to the right half
        else:
            low = mid + 1

    # Target not found in the list
    return -1

# --- Verification with Test Cases ---
if __name__ == "__main__":
    sorted_list = [3, 9, 10, 27, 38, 43, 82]
```



```
if __name__ == "__main__":
    sorted_list = [3, 9, 10, 27, 38, 43, 82]

    test_scenarios = [
        {"target": 27, "expected": 3},    # Middle-ish element
        {"target": 3, "expected": 0},     # First element
        {"target": 82, "expected": 6},    # Last element
        {"target": 100, "expected": -1}, # Element not in list
        {"target": 5, "expected": -1}     # Element missing between values
    ]

    print(f"Searching in list: {sorted_list}\n" + "-"*30)
    for test in test_scenarios:
        result = binary_search(sorted_list, test["target"])
        status = "PASSED" if result == test["expected"] else "FAILED"
        print(f"Target: {test['target']} | Found at index: {result} | {status}")
```

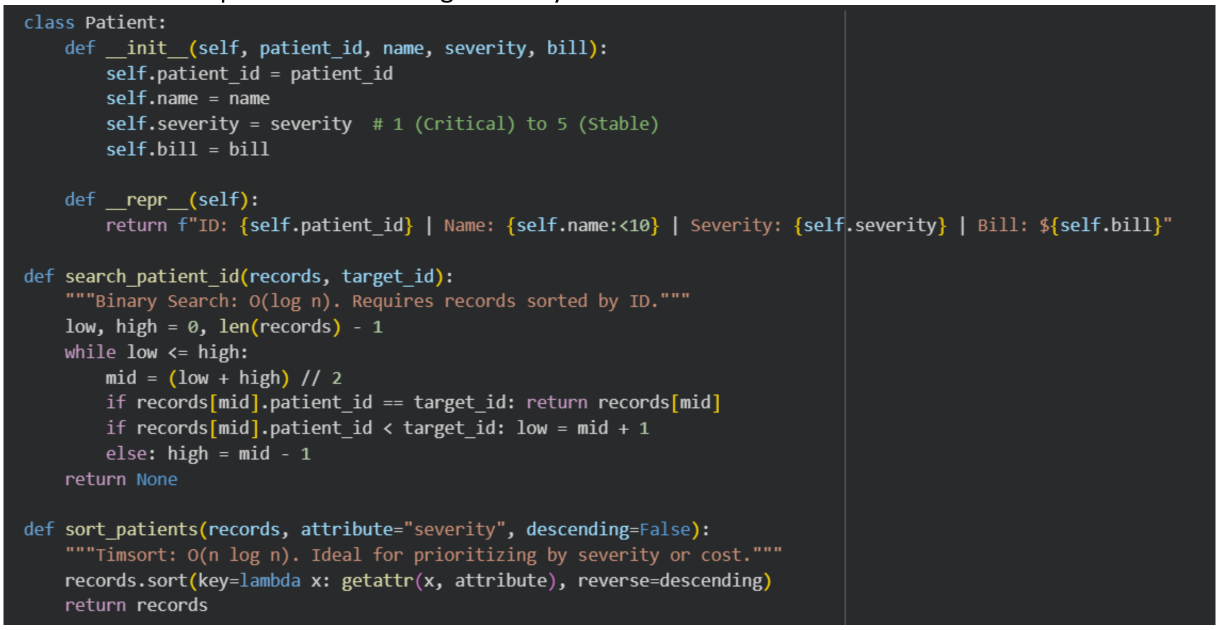
```
Searching in list: [3, 9, 10, 27, 38, 43, 82]
------------------------------
Target: 27 | Found at index: 3 | PASSED
Target: 3 | Found at index: 0 | PASSED
Target: 82 | Found at index: 6 | PASSED
Target: 100 | Found at index: -1 | PASSED
Target: 5 | Found at index: -1 | PASSED
```

Task 3: Real time application-Inventory management System

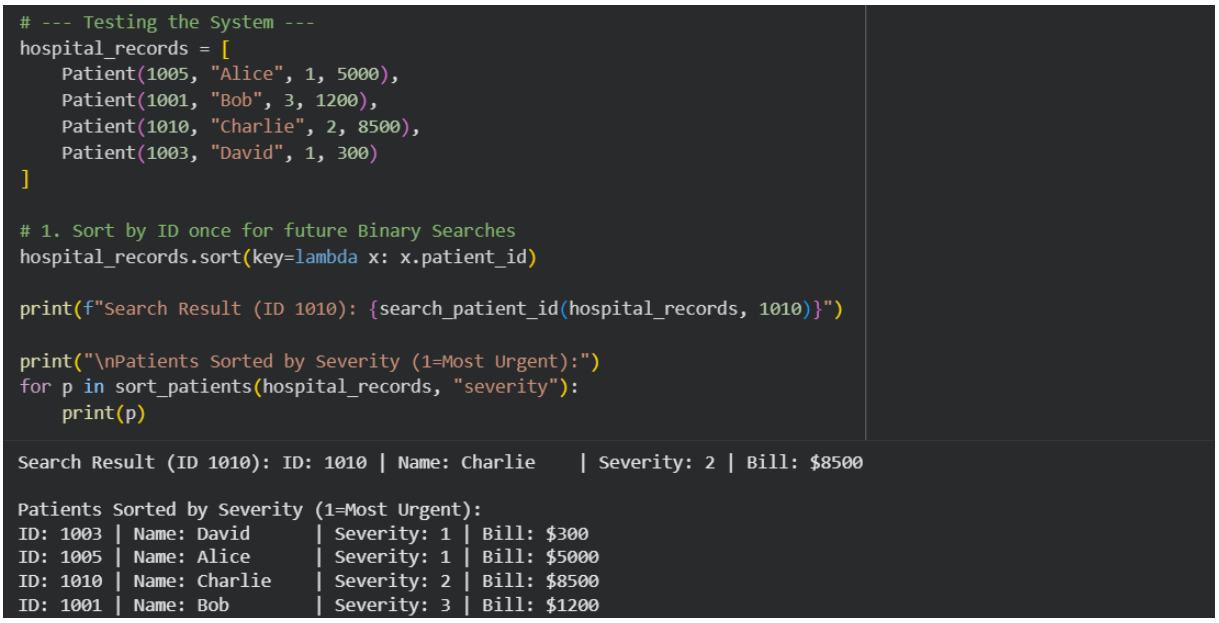


```
    def __init__(self, pid, name, price, qty):
        self.pid, self.name, self.price, self.qty = pid, name, price, qty

    def __repr__(self):
        return f"ID: {self.pid} | {self.name} | ${self.price} | Qty: {self.qty}"

def search_id(inventory, target_id):
    """Binary Search: O(log n). Requires inventory sorted by ID."""
    low, high = 0, len(inventory) - 1
    while low <= high:
        mid = (low + high) // 2
        if inventory[mid].pid == target_id: return inventory[mid]
        if inventory[mid].pid < target_id: low = mid + 1
        else: high = mid - 1
    return None

def sort_inventory(inventory, attr="price"):
    """Timsort: O(n log n). Highly efficient for large datasets."""
    inventory.sort(key=lambda x: getattr(x, attr))
    return inventory

# --- Quick Test ---
stock = [Product(105, "Mouse", 25, 50), Product(101, "Keys", 89, 20)]
stock.sort(key=lambda x: x.pid) # Pre-sort for ID search

print(f"Search ID 101: {search_id(stock, 101)}")
print(f"Sorted by Price: {sort_inventory(stock, 'price')}")
```

Task 4 : Smart hospital Patient Management System



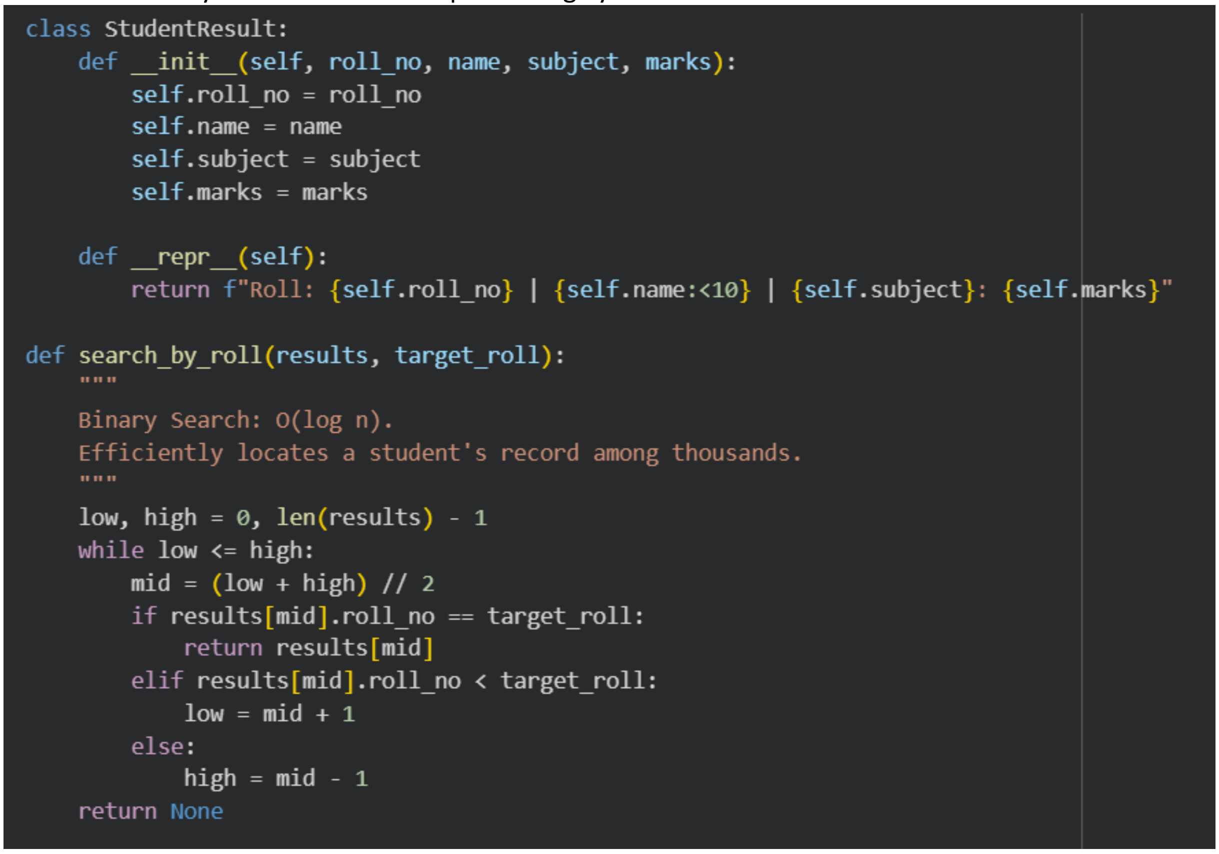
```python
class Patient:
    def __init__(self, patient_id, name, severity, bill):
        self.patient_id = patient_id
        self.name = name
        self.severity = severity  # 1 (Critical) to 5 (Stable)
        self.bill = bill

    def __repr__(self):
        return f"ID: {self.patient_id} | Name: {self.name:<10} | Severity: {self.severity} | Bill: ${self.bill}"

def search_patient_id(records, target_id):
    """Binary Search: O(log n). Requires records sorted by ID."""
    low, high = 0, len(records) - 1
    while low <= high:
        mid = (low + high) // 2
        if records[mid].patient_id == target_id: return records[mid]
        if records[mid].patient_id < target_id: low = mid + 1
        else: high = mid - 1
    return None

def sort_patients(records, attribute="severity", descending=False):
    """Timsort: O(n log n). Ideal for prioritizing by severity or cost."""
    records.sort(key=lambda x: getattr(x, attribute), reverse=descending)
    return records
```



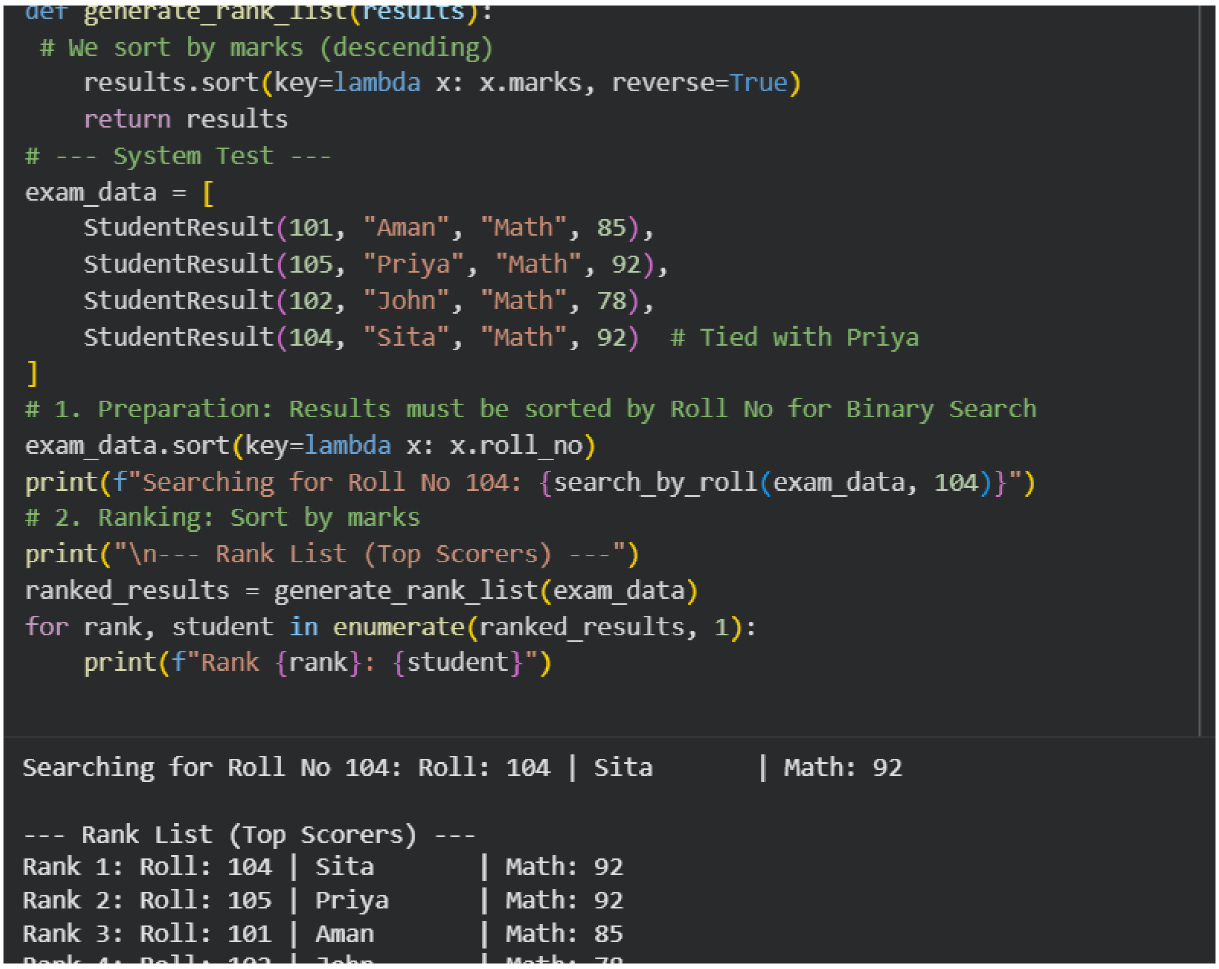
```python
# --- Testing the System ---
hospital_records = [
    Patient(1005, "Alice", 1, 5000),
    Patient(1001, "Bob", 3, 1200),
    Patient(1010, "Charlie", 2, 8500),
    Patient(1003, "David", 1, 300)
]

# 1. Sort by ID once for future Binary Searches
hospital_records.sort(key=lambda x: x.patient_id)

print(f"Search Result (ID 1010): {search_patient_id(hospital_records, 1010)}")

print("\nPatients Sorted by Severity (1=Most Urgent):")
for p in sort_patients(hospital_records, "severity"):
    print(p)
```

```
Search Result (ID 1010): ID: 1010 | Name: Charlie    | Severity: 2 | Bill: $8500

Patients Sorted by Severity (1=Most Urgent):
ID: 1003 | Name: David      | Severity: 1 | Bill: $300
ID: 1005 | Name: Alice      | Severity: 1 | Bill: $5000
ID: 1010 | Name: Charlie    | Severity: 2 | Bill: $8500
ID: 1001 | Name: Bob        | Severity: 3 | Bill: $1200
```

Task 5: University Examination result processing System



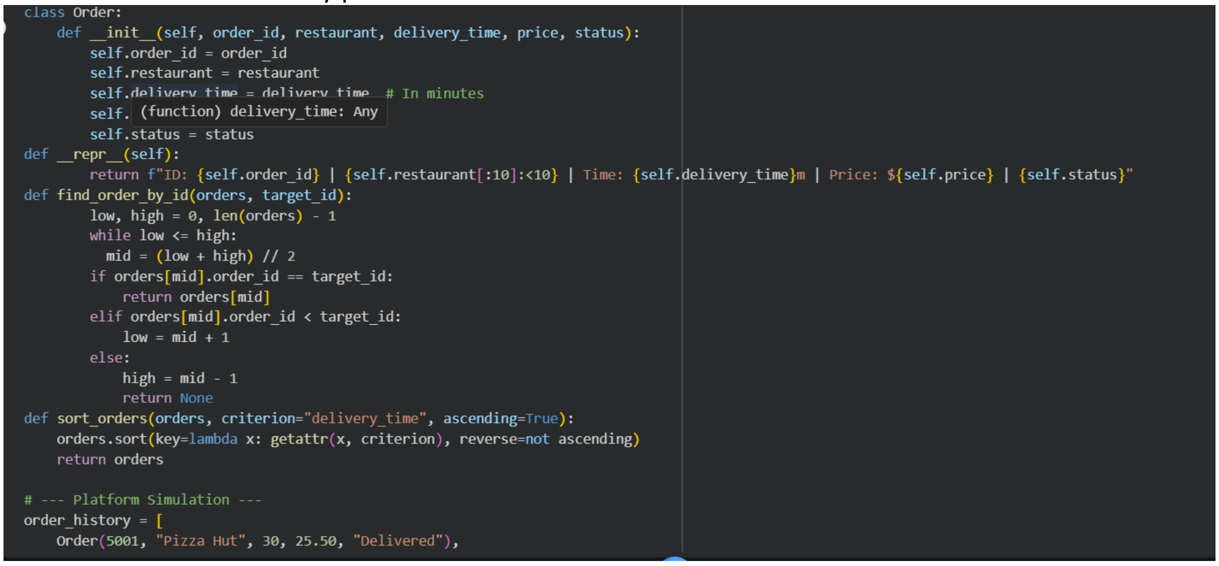
```python
class StudentResult:
    def __init__(self, roll_no, name, subject, marks):
        self.roll_no = roll_no
        self.name = name
        self.subject = subject
        self.marks = marks

    def __repr__(self):
        return f"Roll: {self.roll_no} | {self.name:<10} | {self.subject}: {self.marks}"

def search_by_roll(results, target_roll):
    """
    Binary Search: O(log n).
    Efficiently locates a student's record among thousands.
    """
    low, high = 0, len(results) - 1
    while low <= high:
        mid = (low + high) // 2
        if results[mid].roll_no == target_roll:
            return results[mid]
        elif results[mid].roll_no < target_roll:
            low = mid + 1
        else:
            high = mid - 1
    return None
```



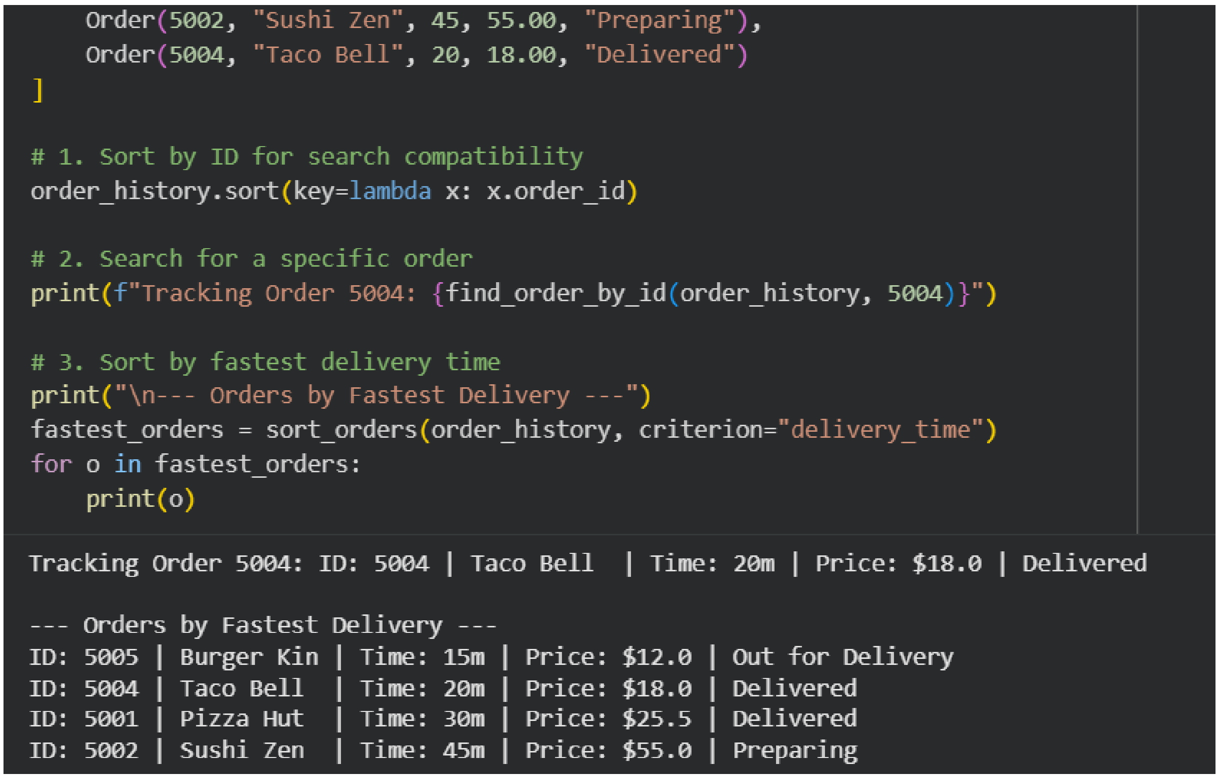
```python
def generate_rank_list(results):
 # We sort by marks (descending)
    results.sort(key=lambda x: x.marks, reverse=True)
    return results
# --- System Test ---
exam_data = [
    StudentResult(101, "Aman", "Math", 85),
    StudentResult(105, "Priya", "Math", 92),
    StudentResult(102, "John", "Math", 78),
    StudentResult(104, "Sita", "Math", 92)  # Tied with Priya
]
# 1. Preparation: Results must be sorted by Roll No for Binary Search
exam_data.sort(key=lambda x: x.roll_no)
print(f"Searching for Roll No 104: {search_by_roll(exam_data, 104)}")
# 2. Ranking: Sort by marks
print("\n--- Rank List (Top Scorers) ---")
ranked_results = generate_rank_list(exam_data)
for rank, student in enumerate(ranked_results, 1):
    print(f"Rank {rank}: {student}")
```

```
Searching for Roll No 104: Roll: 104 | Sita       | Math: 92

--- Rank List (Top Scorers) ---
Rank 1: Roll: 104 | Sita       | Math: 92
Rank 2: Roll: 105 | Priya      | Math: 92
Rank 3: Roll: 101 | Aman       | Math: 85
Rank 4: Roll: 102 | John       | Math: 78
```

Task 6: Online food delivery platform



```
class Order:
    def __init__(self, order_id, restaurant, delivery_time, price, status):
        self.order_id = order_id
        self.restaurant = restaurant
        self.delivery_time = delivery_time  # In minutes
        self. (function) delivery_time: Any
        self.status = status
def __repr__(self):
        return f"ID: {self.order_id} | {self.restaurant[:10]:<10} | Time: {self.delivery_time}m | Price: ${self.price} | {self.status}"
def find_order_by_id(orders, target_id):
        low, high = 0, len(orders) - 1
        while low <= high:
          mid = (low + high) // 2
        if orders[mid].order_id == target_id:
            return orders[mid]
        elif orders[mid].order_id < target_id:
            low = mid + 1
        else:
            high = mid - 1
            return None
def sort_orders(orders, criterion="delivery_time", ascending=True):
    orders.sort(key=lambda x: getattr(x, criterion), reverse=not ascending)
    return orders

# --- Platform Simulation ---
order_history = [
    Order(5001, "Pizza Hut", 30, 25.50, "Delivered"),
```



```
    Order(5002, "Sushi Zen", 45, 55.00, "Preparing"),
    Order(5004, "Taco Bell", 20, 18.00, "Delivered")
]

# 1. Sort by ID for search compatibility
order_history.sort(key=lambda x: x.order_id)

# 2. Search for a specific order
print(f"Tracking Order 5004: {find_order_by_id(order_history, 5004)}")

# 3. Sort by fastest delivery time
print("\n--- Orders by Fastest Delivery ---")
fastest_orders = sort_orders(order_history, criterion="delivery_time")
for o in fastest_orders:
    print(o)
```

```
Tracking Order 5004: ID: 5004 | Taco Bell  | Time: 20m | Price: $18.0 | Delivered

--- Orders by Fastest Delivery ---
ID: 5005 | Burger Kin | Time: 15m | Price: $12.0 | Out for Delivery
ID: 5004 | Taco Bell  | Time: 20m | Price: $18.0 | Delivered
ID: 5001 | Pizza Hut  | Time: 30m | Price: $25.5 | Delivered
ID: 5002 | Sushi Zen  | Time: 45m | Price: $55.0 | Preparing
```