1. Problem Statement:

Implement a basic system to manage a list of tasks. Each task has a priority (a number) and a due date (in the format YYYY-MM-DD). For example: (5, "2024-09-10"). Your system should be able to add tasks and display them sorted by priority.

2. Task Requirements:

2.1. Simple Data Structure:

i. Use a list to store tasks. Each task should be represented as a tuple (priority, due_date), e.g., (5, "2024-09-10").

```
# Task list initialized as an empty list
task_list = []
```

ii. Write a function to add a new task to the list.

```
# Function to add a new task to the list
def add_task(priority, due_date):
    task = (priority, due_date)
    task_list.append(task)       # Add the task to the task list
add_task(5, "2024-09-22")
add_task(3, "2024-10-21")
add_task(1, "2024-11-20")
```

iiii. Write a function to display all tasks.

```
# Function to display all tasks without sorting

def display_tasks():

    for task in task_list:

        print(f"Priority: {task[0]}, Due Date: {task[1]}")

# Display all tasks

display_tasks()
```

2.2. Sorting Tasks:

Implement a simple sorting algorithm to sort tasks by priority (highest priority first). You may use Bubble Sort for simplicity or select another sorting algorithm of your choice. Provide a brief explanation of how your sorting algorithm works.

i. Implement a simple sorting algorithm to sort tasks by priority

```
# Function to implement Bubble Sort

def bubble_sort_tasks():

    n = len(task_list)

    # Perform bubble sort on task_list based on priority

    for i in range(n):

        for j in range(0, n-i-1):
```

```
            # Compare priority and swap if needed
            if task_list[j][0] < task_list[j+1][0]:  # Sort by highest priority first
                task_list[j], task_list[j+1] = task_list[j+1], task_list[j]
# Function to display all tasks
def display_tasks():
    print("Tasks sorted by priority:")
    for task in task_list:
        print(f"Priority: {task[0]}, Due Date: {task[1]}")
# add task in list
add_task(5, "2024-09-22")
add_task(3, "2024-10-21")
add_task(1, "2024-11-20")
# Sort the tasks using Bubble Sort
bubble_sort_tasks()
# Display the sorted tasks
display_tasks()
```

ii. Bubble Sort works:

- The bubble_sort_tasks function sorts task_list in descending order of priority. It uses two loops to compare and swap tasks.

- Outer Loop runs from i = 0 to n-1. This loop controls the number of passes through the list.
- Inner Loop runs from j = 0 to n-i-2. This loop compares adjacent tasks and swaps them if the one with a lower priority is before the one with a higher priority.
- For each pair of adjacent tasks (task_list[j] and task_list[j+1]), the function checks if the priority of task_list[j] is less than the priority of task_list[j+1]. If so, it swaps them..
- The highest priority task moves to the top.
- The next highest priority task moves to its correct position, and so on.
- After all the passes, the task with the highest priority bubbles up to the beginning of the list, and the list is sorted in descending order.

2.3. Basic Operations:

i. Write a function to find the task with the highest priority

```
# Function to find the task with the highest priority
def find_highest_priority_task():
    if not task_list:
        print("No tasks available.")
        return None
    # Initialize the highest priority task
    highest_priority_task = task_list[0]
    # Iterate through the list to find the task with the highest priority
```

```
    for task in task_list:
        if task[0] > highest_priority_task[0]:  # Compare priorities
            highest_priority_task = task
    print(f"Highest priority task: Priority {highest_priority_task[0]}, Due Date
{highest_priority_task[1]}")
    return highest_priority_task
# Add task in list
add_task(5, "2024-09-22")
add_task(3, "2024-10-21")
add_task(1, "2024-11-20")
# Display the highest priority task
find_highest_priority_task()
```

ii. Write a function to remove a task by its priority

```
    # Function to remove a task by its priority
def remove_task_by_priority(priority):
    for task in task_list:
        if task[0] == priority:
            task_list.remove(task)
            print(f"Task with priority {priority} removed.")
```

```
        return
    print(f"No task found with priority {priority}.")
# add task in list
add_task(5, "2024-09-22")
add_task(3, "2024-10-21")
add_task(1, "2024-11-20")
# Remove task with priority
remove_task_by_priority(3)
# Display the updated task list
display_tasks()
```

2.4. Complexity Analysis:

i. Analyse the time complexity of the sorting algorithm you implement in (2).

The Bubble Sort function arranges tasks by priority in descending order. y. It consists of an inner loop that compares and swaps adjacent tasks up to n-i-1 times and an outer loop that runs n times.

The time complexity of Bubble Sort:

- If average and worst cases when compare every pair of tasks, the time complexity of Bubble Sort would be $O(n^2)$
- If an optimized version of Bubble Sort is used and the list has already been sorted, the best case time would be O(n).

ii. Write a short explanation (1-2 sentences) of how the time it takes to add, find, and remove tasks might change as the number of tasks increases.

Adding a task takes O(1) time due to simple append operations. Finding the highest priority task takes $O(n^2)$ time due to sorting, but is O(1) after sorting, while removing a task takes O(n) time due to searching through the list.

2.5. Report and Reflection:

i. Implementation the data structures and algorithms.

- Python List Data Structures: Stores tasks as tuples with priority and due date.
- Algorithms:
  - Adding a Task: Uses append() method to add new tasks.
  - Displaying Tasks: Uses simple loop to display tasks in order.
  - Sorting Tasks: Bubble Sort sort tasks by priority in descending order.
  - Finding Highest Priority Task: Accesses highest priority task first.
  - Removing a Task: Searches for and removes tasks.

ii. Challenges and Solutions

- Sorting Tasks: Bubble Sort is slow for large lists, but Quick Sort can be used for larger lists.
- Finding the Highest Priority Task: Sorting the list first is necessary, but the task is easily found as the first item.
- Removing a Task: Searching the list for a task can be time-consuming, but the built-in remove() method simplifies task removal.

iii. Lessons Learned

- Sorting algorithms impact performance; faster algorithms like Quick Sort better for larger datasets.
- Simple data structures may become inefficient for tasks like searching and removal.
- Advanced data structures like priority queues improve performance for priority-based tasks.