GROUP 4

OKIDI NORBERT	S23B23/051
KIISA ANGELA GRACE	S23B23/027
KISUZE GARETH NEVILLE	S23B23/029

CSC2107 PROJECT

Technical Design Document

Project:

Python-Based Personal Scheduling Assistant

Objective:

To create a personal scheduling assistant that helps users manage and optimize daily activities using efficient algorithms for sorting, searching, and scheduling.

System Overview

The system manages tasks categorized into personal and academic types. It uses dynamic programming to optimize scheduling, binary search for quick retrieval, and sorting algorithms for task organization. The system provides reminders and visualizations using Gantt charts.

System Components

1. Task Management Module

Manages the addition, deletion, and updating of tasks. Tasks are categorized as either personal or academic.

2. Sorting Module

Uses efficient algorithms (merge sort) to organize tasks based on priority, deadlines, or type.

3. Search Module

Implements binary search for efficient task retrieval by specific attributes like deadline or priority.

4. Scheduling Optimization Module

Uses dynamic programming to maximize the number of non-conflicting tasks within a timeframe.

5. Visualization Module

Generates Gantt charts using matplotlib to visually represent task schedules.

6. Reminder Module

Schedules reminders for tasks approaching their deadlines using a time-based scheduler.

Data Structures

Task Class

A blueprint for representing tasks.

```
class Task:
  - name (str): Name of the task
  - task_type (str): Type of the task ('personal' or 'academic')
  - deadline (int): Deadline of the task (in hours)
  - start_time (int): Start time of the task (in hours)
  - duration (int): Duration of the task (in hours)
  - priority (int): Priority of the task (lower value = higher priority)
```

Algorithms

1. Sorting Tasks Using Merge Sort

Organizes tasks based on priority, deadlines, or other attributes.

Pseudo-code:

```
function merge_sort(tasks, key):
    if length(tasks) <= 1:
        return tasks
    mid = length(tasks) // 2
    left = merge_sort(tasks[:mid], key)
    right = merge_sort(tasks[mid:], key)
    return merge(left, right, key)

function merge(left, right, key):
    sorted_list = []
    while left and right:
        if left[0][key] <= right[0][key]:
            append left[0] to sorted_list and remove from left
        else:
            append right[0] to sorted_list and remove from right
    append remaining left or right to sorted_list
    return sorted_list
```

---

2. Searching Tasks Using Binary Search

Finds tasks with a specific deadline or priority.

Pseudo-code:

```
function binary_search(tasks, target):
    deadlines = [task.deadline for task in tasks]
    low, high = 0, length(tasks) - 1
    while low <= high:
        mid = (low + high) // 2
        if deadlines[mid] == target:
            return tasks[mid]
        else if deadlines[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return None
```

---

3. Scheduling Using Dynamic Programming

Maximizes task duration without conflicts.

Pseudo-code:

```
function maximize_tasks(tasks):
    sort tasks by deadline using merge_sort
    dp = [0] * length(tasks)
    dp[0] = tasks[0].duration

    for i = 1 to length(tasks) - 1:
        include_task = tasks[i].duration
        last_non_conflicting = -1

        for j = i-1 to 0:
            if tasks[j].deadline <= tasks[i].start_time:
                last_non_conflicting = j
                break

        if last_non_conflicting != -1:
            include_task += dp[last_non_conflicting]

        dp[i] = max(include_task, dp[i-1])

    return dp[length(tasks) - 1]
```

---

4. Visualization Using Gantt Chart

Plots task schedules using horizontal bars.

Pseudo-code:

```
function plot_gantt(tasks):
    initialize plot
    for each task in tasks:
        if task.type == 'personal':
            plot horizontal bar with color skyblue
        else:
            plot horizontal bar with color orange
    show plot
```

---

5. Reminder Scheduling

Schedules reminders for upcoming tasks.

Pseudo-code:

```
function schedule_reminder(task):
    delay = max(0, task.deadline - current_time)
    scheduler.enter(delay, priority=1, function=notify_user, arguments=(task.name,))
```

Execution Flow

1Add Tasks: Users input tasks with details (name, type, start time, duration, deadline, priority).

2. Sort Tasks: Sort tasks based on desired attributes (e.g., priority, deadline).

3. Search Tasks: Quickly retrieve tasks by deadline or priority.

4. Optimize Schedule: Use dynamic programming to resolve overlaps and maximize completed tasks.

5. Visualize Schedule: Display tasks as a Gantt chart for easy understanding.

6. Set Reminders: Notify users of upcoming deadlines.

Tools and Libraries

1. Python Libraries: `matplotlib`, `sched`, `time`

2. IDE: VSCode

Conclusion

This design ensures efficient task management, scheduling, and visualization while leveraging dynamic programming and algorithmic principles.