



**UGANDA CHRISTIAN
UNIVERSITY**

A Centre of Excellence in the Heart of Africa

**FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY
DEPARTMENT OF COMPUTING AND TECHNOLOGY
ADVENT 2024 SEMESTER DAA COURSEWORK TECHNICAL
DESIGN DOCUMENT**

PROGRAM: BSCS

COURSE: Design and Analysis of Algorithms

COURSE LECTURER: Habare John

PROJECT TITLE:

Personal Scheduling Assistant in Python

Submitted by:

NAME	REGISTRATION NUMBER
MOKILI PROMISE PIERRE	M23B23/032
BUWEMBO DAVID DENZEL	
LUFENE MARK TRAVIS	S23B23/032

Table of Contents

I. Overview.....	1
II. System Architecture.....	2
A. 2.1 Components.....	2.1
B. 2.2 High-Level Workflow.....	2.2
III. Features and Functional Requirements.....	3
A. 3.1 Features.....	3.1
B. 3.2 Functional Requirements.....	3.2
IV. Algorithm Design.....	4
A. 4.1 Sorting Tasks by Deadline (Merge Sort).....	4.1
B. 4.2 Dynamic Programming for Task Scheduling.....	4.2
V. Technical Design.....	5
A. 5.1 Task Class.....	5.1
B. 5.2 Scheduler Class.....	5.2
C. 5.3 SchedulerApp Class.....	5.3
VI. Gantt Chart Visualization.....	6

Personal Scheduling Assistant

1. Overview

The Personal Scheduling Assistant is a Python-based application designed to help users organize tasks efficiently using dynamic programming techniques, sorting algorithms, and a visual representation of tasks using a Gantt chart. Built using **Tkinter** for the GUI and **Matplotlib** for task visualization, this project demonstrates the application of algorithm design principles.

2. System Architecture

2.1 Components

- **Task Class:** Represents individual tasks with attributes like name, category, priority, deadline, and duration.
- **Scheduler Class:** Handles task management operations such as adding, removing, and visualizing tasks.
- **SchedulerApp Class (GUI):** Implements the user interface for interacting with the Scheduler.
- **Matplotlib Integration:** Generates a Gantt chart for tasks.

2.2 High-Level Workflow

1. User enters task details via the GUI.
2. The **Task** object is created and stored in the **Scheduler** class.
3. Tasks are sorted and visualized based on deadlines using optimized sorting algorithms (e.g., merge sort).
4. A Gantt chart displays task schedules visually.

3. Features and Functional Requirements

3.1 Features

- Add, remove, and view tasks categorized as **Personal** or **Academic**.

- Assign priorities and deadlines to tasks.
- Generate Gantt charts to display scheduled tasks.
- Sort tasks dynamically based on priority and deadlines.

3.2 Functional Requirements

- Input validation for task attributes.
- Deadlines must follow the format YYYY-MM-DD HH:MM.
- Efficient scheduling using dynamic programming and optimized sorting.
- Handle conflicts where tasks overlap or deadlines are missed.

4. Algorithm Design

4.1 Sorting Tasks by Deadline (Merge Sort)

- **Purpose:** To ensure tasks are scheduled in chronological order.
- **Complexity:** $O(n \log n)$

Pseudocode:

```
function merge_sort(tasks):
```

```
    if length(tasks) ≤ 1:
```

```
        return tasks
```

```
    mid = length(tasks) // 2
```

```
    left = merge_sort(tasks[0:mid])
```

```
    right = merge_sort(tasks[mid:])
```

```
    return merge(left, right)
```

```
function merge(left, right):
```

```
    sorted_tasks = []
```

```
    while left and right:
```

```
        if left[0].deadline < right[0].deadline:
```

```
            sorted_tasks.append(left.pop(0))
```

```
        else:
```

```
            sorted_tasks.append(right.pop(0))
```

```
return sorted_tasks + left + right
```

4.2 Dynamic Programming for Task Scheduling

- **Purpose:** Optimize task scheduling to minimize deadline conflicts.
- **Approach:** Use dynamic programming to calculate the maximum number of tasks that can be scheduled within their deadlines.

Pseudocode:

```
function schedule_tasks(tasks):  
    sort tasks by deadline  
    dp = [0] * (n + 1)  
    for i from 1 to n:  
        for j from i - 1 to 0:  
            if tasks[i].deadline >= tasks[j].deadline + tasks[j].duration:  
                dp[i] = max(dp[i], dp[j] + 1)  
    return dp[n]
```

5. Technical Design

5.1 Task Class

Attributes:

- name: Task name.
- category: Task type (Personal/Academic).
- priority: Task importance (1-3).
- deadline: Task deadline as a datetime object.
- duration: Task duration in hours.

Pseudocode:

```
class Task:  
    attributes: name, category, priority, deadline, duration  
    function __init__(name, category, priority, deadline, duration):  
        initialize attributes
```

5.2 Scheduler Class

Methods:

- `add_task(task)`: Adds a task to the list.
- `remove_task(task_name)`: Removes a task by name.
- `display_gantt_chart()`: Visualizes tasks using Matplotlib.

Pseudocode:

class Scheduler:

attributes: tasks = []

function `add_task(task)`:

append task to tasks list

function `remove_task(task_name)`:

filter tasks list to exclude task_name

function `display_gantt_chart()`:

sort tasks by deadline using mergeSort

generate Gantt chart with task durations

5.3 SchedulerApp Class

Methods:

- `add_task()`: Collects inputs, validates them, and adds tasks.
- `remove_task()`: Removes selected task from the scheduler.
- `refresh_task_list()`: Updates the GUI task list display.
- `show_gantt_chart()`: Displays the Gantt chart.

Pseudocode:

class SchedulerApp:

attributes: root, scheduler, GUI elements

function `add_task()`:

collect inputs

validate inputs

```
create Task object  
call scheduler.add_task(task)  
refresh task list display
```

```
function remove_task():  
    get selected task  
    call scheduler.remove_task(task_name)  
    refresh task list display
```

```
function refresh_task_list():  
    clear GUI list  
    display updated tasks
```

```
function show_gantt_chart():  
    call scheduler.display_gantt_chart()
```

6. Gantt Chart Visualization

The `display_gantt_chart` method uses Matplotlib to create horizontal bar plots for tasks, showing their start and end times on a timeline. Tasks are labeled by name for clarity.