Year:-	Academic Year- 2025-26	Semester:-

Experiment # 4
PART B

Roll No: D084	Name: Somish Jain
Class: BTech CE-B	Batch: 2
Date of Experiment: 11 August 2025	Date of Submission: 11 August 2025

Study / Implementation details:

PREEMPTIVE

```
import pandas as pd
import matplotlib.pyplot as plt
import requests


def srtf_scheduler(process_list):
    """Implements Preemptive Shortest Remaining Time First Scheduling"""


    proc_data = {pid: {"arr_time": at, "burst_time": bt} for pid, at, bt
in process_list}
    remaining_time = {pid: bt for pid, at, bt in process_list}

    current_time = min(at for _, at, _ in process_list)
    gantt = []
    first_exec = {}
    completion_time = {}
    active_pid = None
    seg_start_time = None

    while any(remaining_time[pid] > 0 for pid in remaining_time):

        available = [pid for pid in proc_data if
proc_data[pid]["arr_time"] <= current_time and remaining_time[pid] > 0]

        if not available:
            current_time += 1
```

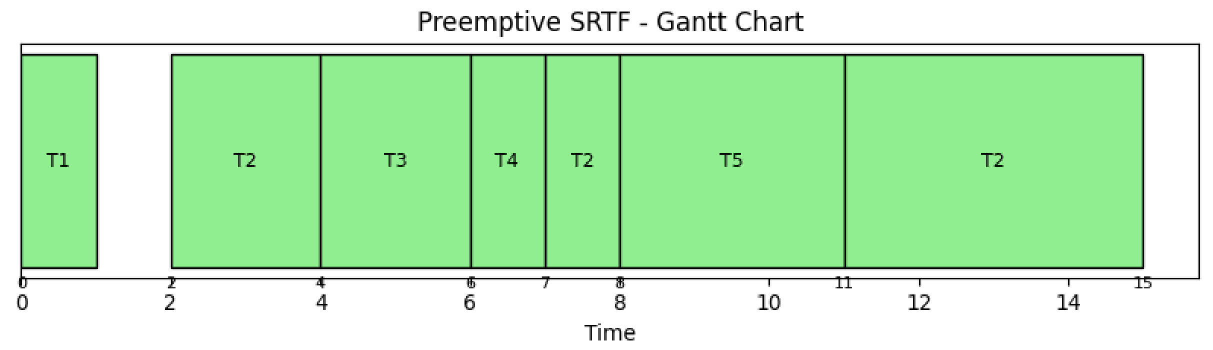
```
            continue

        selected = min(available, key=lambda p: remaining_time[p])

        if selected != active_pid:
            if active_pid is not None:
                gantt.append((active_pid, seg_start_time, current_time))
            seg_start_time = current_time
            active_pid = selected
            if selected not in first_exec:
                first_exec[selected] = current_time


        remaining_time[selected] -= 1
        current_time += 1

        if remaining_time[selected] == 0:
            gantt.append((selected, seg_start_time, current_time))
            completion_time[selected] = current_time
            active_pid = None
            seg_start_time = None


    rows = []
    for pid in sorted(proc_data):
        at = proc_data[pid]["arr_time"]
        bt = proc_data[pid]["burst_time"]
        st = first_exec[pid]
        ct = completion_time[pid]
        tat = ct - at
        wt = tat - bt
        rt = st - at
        rows.append({"PID": pid, "Arrival": at, "Burst": bt,
                     "Start": st, "Finish": ct,
                     "TAT": tat, "WT": wt, "RT": rt})

    df = pd.DataFrame(rows)
    metrics = {
        "Average WT": df["WT"].mean(),
        "Average TAT": df["TAT"].mean(),
        "Average RT": df["RT"].mean()
    }
    return gantt, df, metrics
```

```
def plot_gantt_chart(gantt_data):
    fig, ax = plt.subplots(figsize=(10, 2))
    for process, start, end in gantt_data:
        ax.barh(y=0, width=end - start, left=start, height=0.5,
                align='center', color="lightgreen", edgecolor="black")
        ax.text((start + end) / 2, 0, process, ha="center", va="center",
fontsize=9)
        ax.text(start, -0.3, str(start), ha="center", fontsize=8)
    ax.text(gantt_data[-1][2], -0.3, str(gantt_data[-1][2]), ha="center",
fontsize=8)
    ax.set_yticks([])
    ax.set_xlabel("Time")
    ax.set_title("Preemptive SRTF - Gantt Chart")
    plt.show()


def manual_input():
    print("Enter process info: PID ArrivalTime BurstTime")
    result = []
    for i in range(5):
        pid, at, bt = input(f"Process {i+1}: ").split()
        result.append((pid, float(at), float(bt)))
    return result


def fetch_from_api():
    try:
        res =
requests.get("https://jsonplaceholder.typicode.com/comments")
        res.raise_for_status()
    except Exception as err:
        print("API fetch failed:", err)
        exit(1)

    data = res.json()
    tasks = []
    for idx, item in enumerate(data[:5]):
        pid = f"T{idx+1}"
        arrival = idx * 2
        burst = (len(item["name"]) % 7) + 1
        tasks.append((pid, arrival, burst))
    return tasks


if __name__ == "__main__":
```

```python
    print("Choose input type:")
    print("1. Manual")
    print("2. API")
    option = input("Your choice: ").strip()

    if option == "1":
        procs = manual_input()
    elif option == "2":
        procs = fetch_from_api()
    else:
        print("Invalid choice")
        exit(0)

    gantt, table, averages = srtf_scheduler(procs)
    print("\nSchedule Table:\n", table)
    print("\nAverages:\n", averages)
    plot_gantt_chart(gantt)
```

Output:

```
Choose input type:
1. Manual
2. API
Your choice: 2

Schedule Table:
   PID  Arrival  Burst  Start  Finish  TAT  WT  RT
0  T1        0      1      0       1    1   0   0
1  T2        2      7      2      15   13   6   0
2  T3        4      2      4       6    2   0   0
3  T4        6      1      6       7    1   0   0
4  T5        8      3      8      11    3   0   0

Averages:
 {'Average WT': np.float64(1.2), 'Average TAT': np.float64(4.0), 'Average RT': np.float64(0.0)}
```



NON-PREEMPTIVE

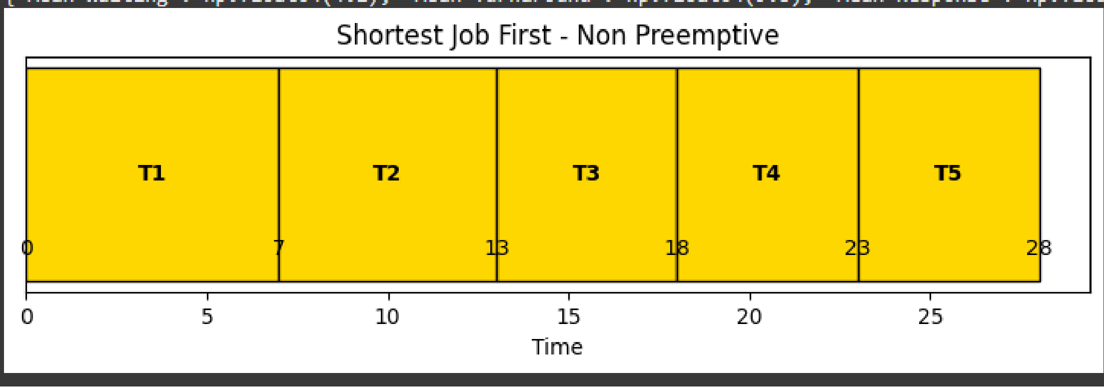
```
import pandas as pd
import matplotlib.pyplot as plt
import requests

def sjf_non_preemptive(processes):
    proc_data = {p: {"arr": a, "burst": b} for p, a, b in processes}
    done = set()
    timeline = []
    starts, finishes = {}, {}
    t = min(a for _, a, _ in processes)

    while len(done) < len(proc_data):
        ready = [p for p, v in proc_data.items() if v["arr"] <= t and p
not in done]
        if not ready:
            t += 1
            continue
        job = min(ready, key=lambda x: proc_data[x]["burst"])
        starts[job] = t
        t += proc_data[job]["burst"]
        finishes[job] = t
        timeline.append((job, starts[job], finishes[job]))
        done.add(job)

    records = []
    for p in sorted(proc_data.keys()):
        a = proc_data[p]["arr"]
        b = proc_data[p]["burst"]
        s = starts[p]
        c = finishes[p]
        tat = c - a
        wt = tat - b
        rt = s - a
        records.append({
            "PID": p,
            "Arrival": a,
            "Burst": b,
            "Start": s,
            "Completion": c,
            "Turnaround": tat,
            "Waiting": wt,
            "Response": rt
        })
```

```
    df = pd.DataFrame(records)
    metrics = {
        "Mean Waiting": df["Waiting"].mean(),
        "Mean Turnaround": df["Turnaround"].mean(),
        "Mean Response": df["Response"].mean()
    }
    return timeline, df, metrics

def plot_gantt(entries):
    fig, ax = plt.subplots(figsize=(9, 2))
    for pid, st, et in entries:
        ax.barh(0, et - st, left=st, color="gold", edgecolor="black")
        ax.text((st + et) / 2, 0, pid, ha="center", va="center",
weight="bold")
        ax.text(st, -0.3, str(st), ha="center")
    ax.text(entries[-1][2], -0.3, str(entries[-1][2]), ha="center")
    ax.set_yticks([])
    ax.set_xlabel("Time")
    ax.set_title("Shortest Job First - Non Preemptive")
    plt.show()

def manual_input():
    res = []
    for i in range(5):
        pid, a, b = input(f"Enter job {i+1} (PID Arrival Burst):
").split()
        res.append((pid, float(a), float(b)))
    return res

def api_input():
    try:
        r = requests.get("https://jsonplaceholder.typicode.com/users")
        r.raise_for_status()
    except:
        print("API unavailable")
        exit(1)
    data = r.json()
    jobs = []
    for i, item in enumerate(data[:5]):
        pid = f"T{i+1}"
        arr = i * 4
        burst = (len(item["name"]) % 5) + 4
        jobs.append((pid, arr, burst))
    return jobs
```

```python
if __name__ == "__main__":
    choice = input("1 - Manual | 2 - API: ").strip()
    if choice == "1":
        plist = manual_input()
    elif choice == "2":
        plist = api_input()
    else:
        exit(0)
    chart, table, avg = sjf_non_preemptive(plist)
    print(table)
    print(avg)
    plot_gantt(chart)
```

Output:-

```
1 - Manual | 2 - API: 2
  PID  Arrival  Burst  Start  Completion  Turnaround  Waiting  Response
0  T1        0      7      0           7           7        0         0
1  T2        4      6      7          13           9        3         3
2  T3        8      5     13          18          10        5         5
3  T4       12      5     18          23          11        6         6
4  T5       16      5     23          28          12        7         7
{'Mean Waiting': np.float64(4.2), 'Mean Turnaround': np.float64(9.8), 'Mean Response': np.float64(4.2)}
```



QA:

How does SJF differ from FCFS?

The Shortest Job First (SJF) algorithm prioritizes tasks by evaluating their execution duration, consistently selecting the process requiring minimal CPU time from available candidates. Conversely, First Come First Served (FCFS) maintains a rigid sequential approach, processing jobs in their exact arrival sequence regardless of computational demands. This core distinction enables SJF to achieve superior average waiting times and reduced turnaround periods compared to FCFS. The optimization occurs because shorter tasks complete quickly, reducing the overall queue delay for subsequent processes.

What is the major drawback of SJF scheduling?

The most significant weakness of SJF lies in its potential for indefinite postponement, where lengthy processes may never receive CPU allocation. This phenomenon occurs when continuous streams of shorter tasks arrive, perpetually pushing longer jobs further back in the scheduling priority. The algorithm also depends heavily on accurate burst time estimation, which presents practical challenges since predicting future execution durations is often impossible. These limitations make SJF theoretically optimal but practically challenging to implement effectively.

How can starvation be resolved in SJF?

Process starvation can be eliminated through priority aging mechanisms that incrementally boost the scheduling priority of waiting tasks over time. As processes remain in the ready queue longer, their effective priority scores increase, eventually allowing them to compete successfully against newly arrived shorter jobs. This gradual priority enhancement guarantees that all processes, regardless of their initial burst time, will eventually receive CPU allocation. The aging approach maintains SJF's efficiency benefits while ensuring system fairness and preventing indefinite delays.

Why is response time important in interactive systems?

Response time serves as a critical metric in interactive environments because it directly influences user satisfaction and system usability. Rapid system responses create an impression of smooth, real-time interaction, allowing users to maintain their workflow momentum and cognitive focus. Extended response delays lead to user frustration, decreased productivity, and perception of system inadequacy or malfunction. Interactive applications require immediate feedback to maintain the illusion of direct manipulation and responsive computing environments.

How does SJF perform on burst-time-heavy workloads?

When processing workloads consisting primarily of long-duration tasks, SJF demonstrates minimal performance advantages compared to simpler FCFS scheduling. The algorithm's optimization potential diminishes significantly when most processes have comparable lengthy execution times, leaving little room for strategic reordering. SJF excels in heterogeneous environments where substantial variation exists between short and long tasks, allowing the scheduler to maximize throughput by completing numerous brief jobs quickly. Homogeneous long-task workloads essentially neutralize SJF's primary advantage of selective processing based on execution duration.

How do you sort and select processes dynamically for SJF?

Dynamic process management in SJF requires maintaining an active ready queue that adapts continuously as new processes arrive and running tasks terminate. The scheduler employs efficient sorting mechanisms, often utilizing priority queues or heap-based data structures to maintain optimal ordering by burst time. Upon CPU availability, the system extracts the process with the minimum remaining execution time, ensuring consistent adherence to SJF principles. This dynamic approach allows real-time adaptation to changing process loads while maintaining algorithmic efficiency through optimized data structure utilization.