**NAME : MARIYAM PATEL**

**SRN:202201178**

**ROLL NO:34**

**ASSIGNMENT-7**

Q.Simulate the following CPU scheduling algorithms:  
a. First come First serve b. Shortest Job First (Non-preemptive) c. Shortest Job First (Preemptive) d. Round Robin e. Priority (Non-preemptive) f. Priority (Non-preemptive)

(l)

CODE-

import matplotlib.pyplot as plt

import numpy as np

from collections import deque

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time, priority=0):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.remaining\_time = burst\_time

self.priority = priority

self.start\_time = 0

self.completion\_time = 0

self.waiting\_time = 0

self.turnaround\_time = 0

class CPUScheduler:

def \_\_init\_\_(self):

self.processes = []

self.gantt\_chart = []

def add\_process(self, pid, arrival\_time, burst\_time, priority=0):

self.processes.append(Process(pid, arrival\_time, burst\_time, priority))

def calculate\_times(self):

avg\_waiting\_time = sum(p.waiting\_time for p in self.processes) / len(self.processes)

avg\_turnaround\_time = sum(p.turnaround\_time for p in self.processes) / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for i, (pid, start, end) in enumerate(self.gantt\_chart):

ax.barh(0, end - start, left=start, height=0.3,

align='center', color=f'C{pid}', alpha=0.8)

ax.text((start + end)/2, 0, f'P{pid}',

ha='center', va='center')

ax.set\_ylim(-0.5, 0.5)

ax.set\_xlabel('Time')

ax.set\_yticks([])

ax.set\_title(title)

plt.grid(True)

plt.show()

def fcfs(self):

self.gantt\_chart = []

processes = sorted(self.processes, key=lambda x: (x.arrival\_time, x.pid))

current\_time = 0

for process in processes:

if current\_time < process.arrival\_time:

current\_time = process.arrival\_time

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

return self.calculate\_times()

def sjf\_non\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.burst\_time, x.arrival\_time))

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

remaining\_processes.remove(process)

return self.calculate\_times()

def sjf\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

current\_process = None

for p in remaining\_processes:

p.remaining\_time = p.burst\_time

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.remaining\_time, x.arrival\_time))

if current\_process != process.pid:

if current\_process is not None:

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = process.pid

start\_time = current\_time

process.remaining\_time -= 1

current\_time += 1

if process.remaining\_time == 0:

process.completion\_time = current\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

remaining\_processes.remove(process)

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = None

return self.calculate\_times()

def round\_robin(self, quantum):

self.gantt\_chart = []

remaining\_processes = deque(sorted(self.processes, key=lambda x: x.arrival\_time))

current\_time = 0

for p in remaining\_processes:

p.remaining\_time = p.burst\_time

while remaining\_processes:

if not remaining\_processes:

break

process = remaining\_processes.popleft()

if process.arrival\_time > current\_time:

current\_time = process.arrival\_time

execution\_time = min(quantum, process.remaining\_time)

self.gantt\_chart.append((process.pid, current\_time,

current\_time + execution\_time))

current\_time += execution\_time

process.remaining\_time -= execution\_time

if process.remaining\_time > 0:

# Find the position to insert the process back

insert\_pos = 0

for i, p in enumerate(remaining\_processes):

if p.arrival\_time > current\_time:

insert\_pos = i

break

insert\_pos = i + 1

remaining\_processes.insert(insert\_pos, process)

else:

process.completion\_time = current\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

return self.calculate\_times()

def priority\_non\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.priority, x.arrival\_time))

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

remaining\_processes.remove(process)

return self.calculate\_times()

def priority\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

current\_process = None

for p in remaining\_processes:

p.remaining\_time = p.burst\_time

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.priority, x.arrival\_time))

if current\_process != process.pid:

if current\_process is not None:

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = process.pid

start\_time = current\_time

process.remaining\_time -= 1

current\_time += 1

if process.remaining\_time == 0:

process.completion\_time = current\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

remaining\_processes.remove(process)

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = None

return self.calculate\_times()

# Example usage and testing

if \_\_name\_\_ == "\_\_main\_\_":

# Test data

processes\_data = [

(1, 0, 6, 2), # (pid, arrival\_time, burst\_time, priority)

(2, 1, 4, 4),

(3, 2, 2, 1),

(4, 3, 3, 3)

]

algorithms = [

("FCFS", lambda s: s.fcfs()),

("SJF (Non-preemptive)", lambda s: s.sjf\_non\_preemptive()),

("SJF (Preemptive)", lambda s: s.sjf\_preemptive()),

("Round Robin (Q=2)", lambda s: s.round\_robin(2)),

("Priority (Non-preemptive)", lambda s: s.priority\_non\_preemptive()),

("Priority (Preemptive)", lambda s: s.priority\_preemptive())

]

for algo\_name, algo\_func in algorithms:

scheduler = CPUScheduler()

# Add processes

for pid, arrival\_time, burst\_time, priority in processes\_data:

scheduler.add\_process(pid, arrival\_time, burst\_time, priority)

# Run algorithm

avg\_waiting\_time, avg\_turnaround\_time = algo\_func(scheduler)

# Print results

print(f"\n{algo\_name} Results:")

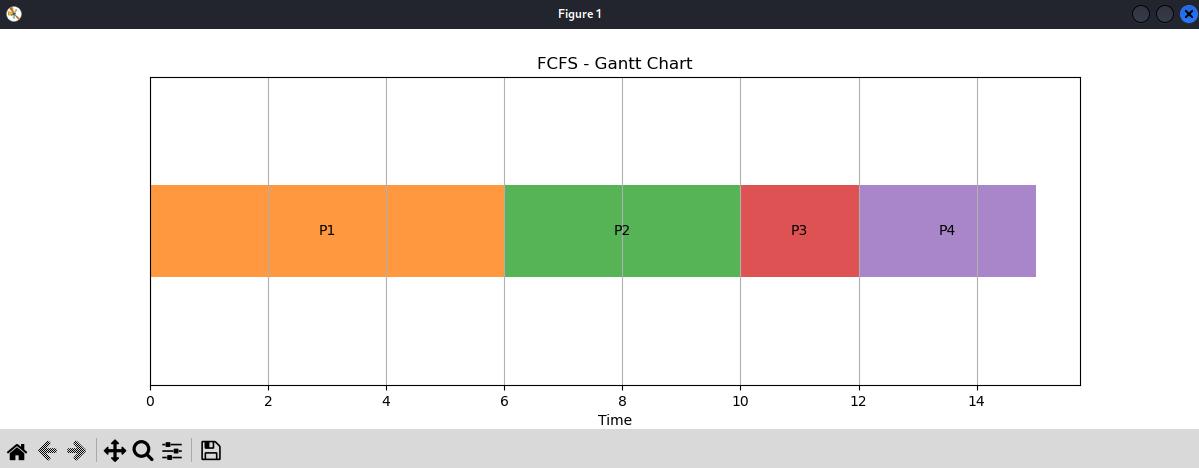
print(f"Average Waiting Time: {avg\_waiting\_time:.2f}")

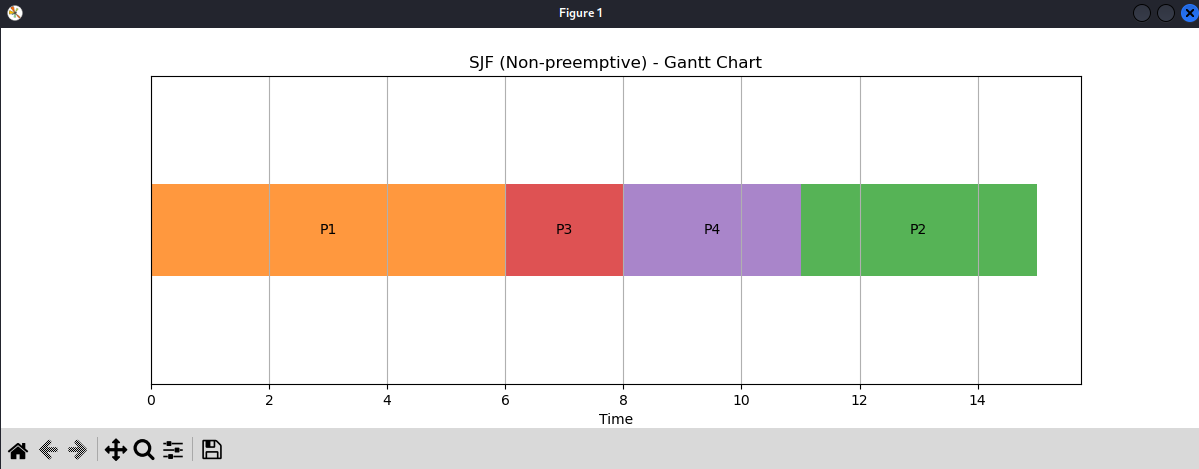
print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

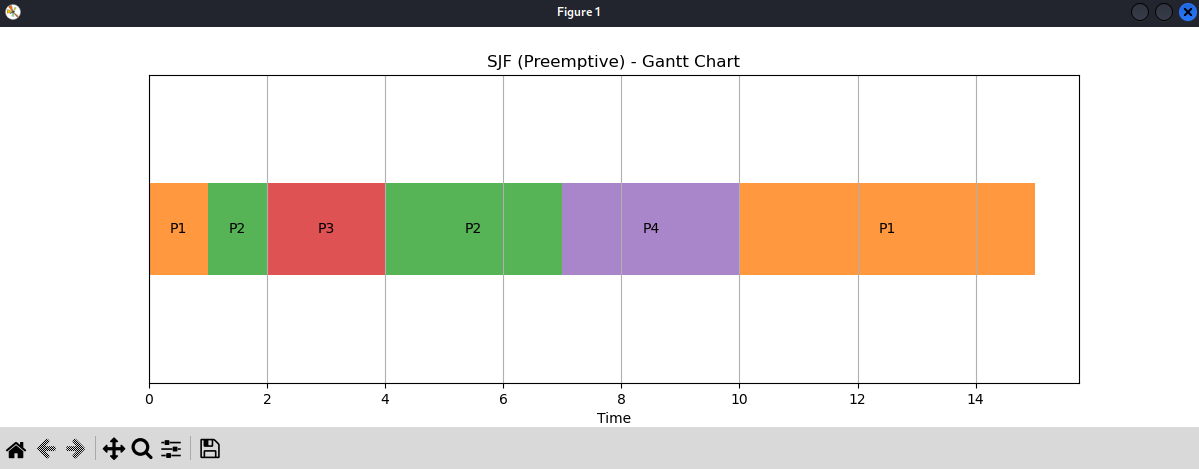
# Plot Gantt chart

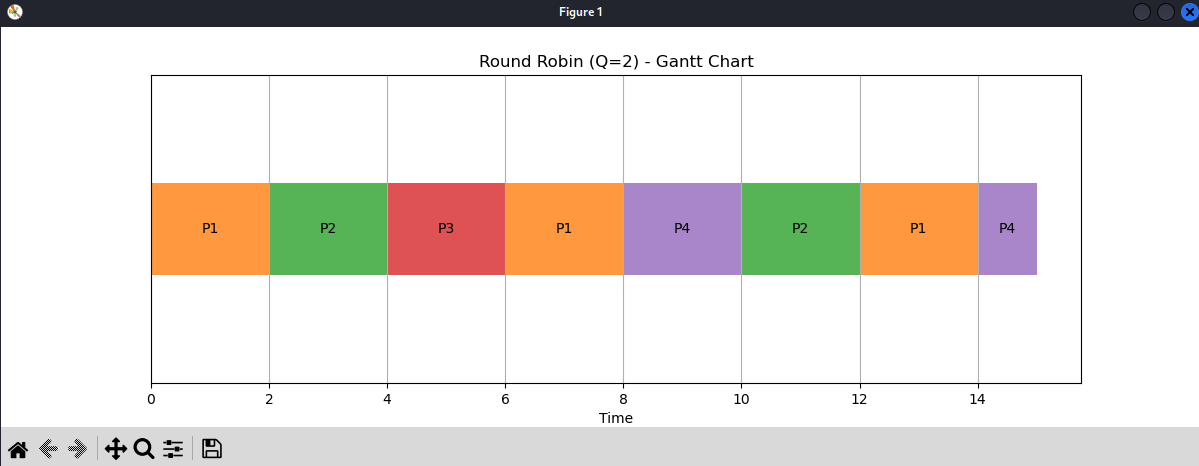
scheduler.plot\_gantt\_chart(f"{algo\_name} - Gantt Chart")

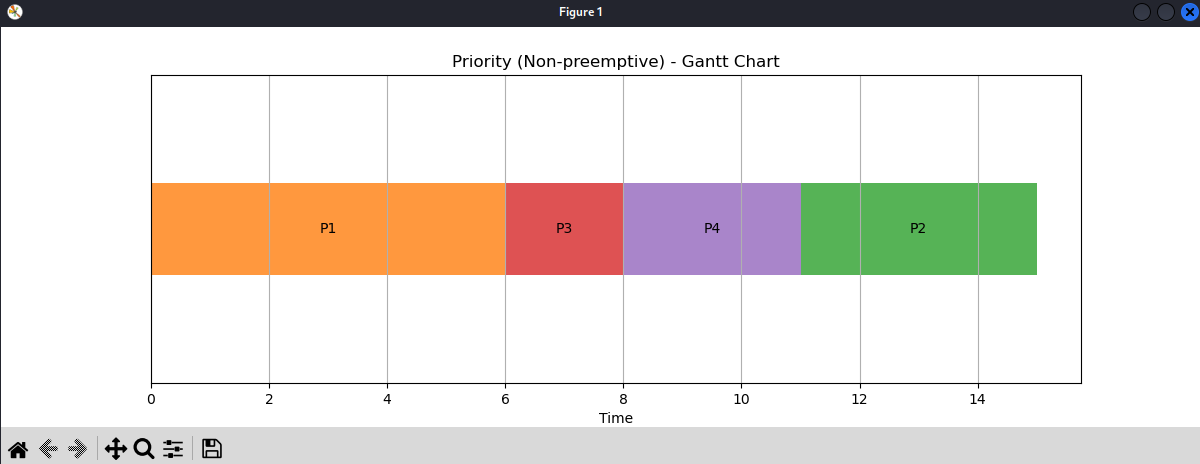
**OUTPUT-**

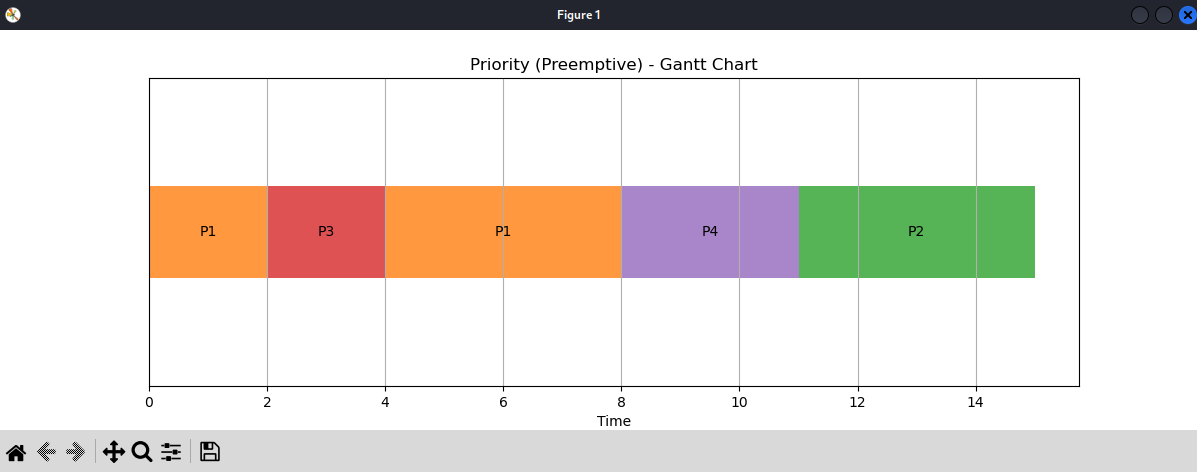
****

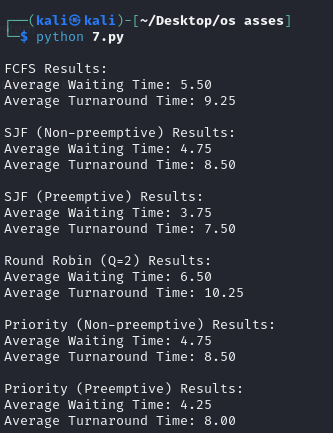
****

****

****

****

****

****

1. **First come First serve**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.start\_time = 0

self.completion\_time = 0

self.waiting\_time = 0

self.turnaround\_time = 0

class CPUSchedulerFCFS:

def \_\_init\_\_(self):

self.processes = []

self.gantt\_chart = []

def add\_process(self, pid, arrival\_time, burst\_time):

self.processes.append(Process(pid, arrival\_time, burst\_time))

def fcfs(self):

self.gantt\_chart = []

processes = sorted(self.processes, key=lambda x: (x.arrival\_time, x.pid))

current\_time = 0

for process in processes:

if current\_time < process.arrival\_time:

current\_time = process.arrival\_time

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

return self.calculate\_times()

def calculate\_times(self):

avg\_waiting\_time = sum(p.waiting\_time for p in self.processes) / len(self.processes)

avg\_turnaround\_time = sum(p.turnaround\_time for p in self.processes) / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for i, (pid, start, end) in enumerate(self.gantt\_chart):

ax.barh(0, end - start, left=start, height=0.3, color=f'C{pid}', alpha=0.8)

ax.text((start + end) / 2, 0, f'P{pid}', ha='center', va='center')

ax.set\_ylim(-0.5, 0.5)

ax.set\_xlabel('Time')

ax.set\_yticks([])

ax.set\_title(title)

plt.grid(True)

plt.show()

# Example usage

scheduler = CPUSchedulerFCFS()

scheduler.add\_process(1, 0, 6)

scheduler.add\_process(2, 1, 4)

scheduler.add\_process(3, 2, 2)

scheduler.add\_process(4, 3, 3)

avg\_waiting\_time, avg\_turnaround\_time = scheduler.fcfs()

print(f"Average Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("FCFS - Gantt Chart")

IN C:

#include<stdio.h>

int main() {

int i, j, temp;

int bt[10] = {0}, at[10] = {0}, tat[10] = {0}, wt[10] = {0}, ct[10] = {0};

int n;

float totalTAT = 0, totalWT = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

int p[n];

for (i = 0; i < n; i++) {

p[i] = i + 1;

}

printf("Enter arrival time and burst time for each process\n\n");

for (i = 0; i < n; i++) {

printf("Arrival time of process[%d]: ", i + 1);

scanf("%d", &at[i]);

printf("Burst time of process[%d]: ", i + 1);

scanf("%d", &bt[i]);

printf("\n");

}

for (i = 0; i < n - 1; i++) {

for (j = i + 1; j < n; j++) {

if (at[i] > at[j]) {

temp = at[i];

at[i] = at[j];

at[j] = temp;

temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

int currentTime = 0;

for (i = 0; i < n; i++) {

if (currentTime < at[i]) {

currentTime = at[i];

}

ct[i] = currentTime + bt[i];

currentTime = ct[i];

tat[i] = ct[i] - at[i];

wt[i] = tat[i] - bt[i];

totalTAT += tat[i];

totalWT += wt[i];

}

printf("\nGantt Chart:\n");

printf(" |");

for (i = 0; i < n; i++) {

printf(" P%d |", p[i]);

}

printf("\n");

printf("0");

for (i = 0; i < n; i++) {

printf(" %d", ct[i]);

}

printf("\n\n");

printf("Solution: \n\n");

printf("P#\t AT\t BT\t CT\t TAT\t WT\n");

for (i = 0; i < n; i++) {

printf("P%d\t %d\t %d\t %d\t %d\t %d\n", p[i], at[i], bt[i], ct[i], tat[i], wt[i]);

}

printf("\nAverage Turnaround Time = %.2f\n", totalTAT / n);

printf("Average Waiting Time = %.2f\n\n", totalWT / n);

return 0;

}

1. **Shortest Job First (Non-preemptive)**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

class SJFScheduler:

def \_\_init\_\_(self, processes):

self.processes = processes

self.gantt\_chart = []

def schedule(self):

# Sort processes by arrival time, then by burst time

self.processes.sort(key=lambda p: (p.arrival\_time, p.burst\_time))

time = 0

completed\_processes = 0

while completed\_processes < len(self.processes):

# Filter processes that have arrived and are not completed

available\_processes = [p for p in self.processes if p.arrival\_time <= time and p.completion\_time == 0]

if not available\_processes:

time += 1

continue

# Select the process with the shortest burst time

shortest\_job = min(available\_processes, key=lambda p: p.burst\_time)

start\_time = time

time += shortest\_job.burst\_time

shortest\_job.completion\_time = time

shortest\_job.turnaround\_time = shortest\_job.completion\_time - shortest\_job.arrival\_time

shortest\_job.waiting\_time = shortest\_job.turnaround\_time - shortest\_job.burst\_time

# Add to Gantt chart

self.gantt\_chart.append((shortest\_job.pid, start\_time, time))

completed\_processes += 1

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data

processes = [

Process(pid=1, arrival\_time=0, burst\_time=7),

Process(pid=2, arrival\_time=2, burst\_time=4),

Process(pid=3, arrival\_time=4, burst\_time=1),

Process(pid=4, arrival\_time=5, burst\_time=4)

]

scheduler = SJFScheduler(processes)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("SJF Non-preemptive - Gantt Chart")

IN C:

#include <stdio.h>

int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Burst Time:\n");

for (i = 0; i < n; i++) {

printf("P%d: ", i + 1);

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

for (i = 0; i < n; i++) {

index = i;

for (j = i + 1; j < n; j++) {

if (A[j][1] < A[index][1]) {

index = j;

}

}

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

for (i = 1; i < n; i++) {

A[i][2] = 0;

for (j = 0; j < i; j++) {

A[i][2] += A[j][1];

}

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

printf("\nP\tBT\tWT\tTAT\n");

for (i = 0; i < n; i++) {

A[i][3] = A[i][1] + A[i][2]; // TAT = BT + WT

total += A[i][3];

printf("P%d\t%d\t%d\t%d\n", A[i][0], A[i][1], A[i][2], A[i][3]);

}

avg\_tat = (float)total / n;

printf("\nGantt Chart:\n");

printf("|");

for (i = 0; i < n; i++) {

printf(" P%d |", A[i][0]);

}

printf("\n");

int completion\_time = 0;

printf("0");

for (i = 0; i < n; i++) {

completion\_time += A[i][1];

printf(" %d", completion\_time);

}

printf("\n\nAverage Waiting Time = %.2f", avg\_wt);

printf("\nAverage Turnaround Time = %.2f\n", avg\_tat);

return 0;

}

1. **Shortest Job First (Preemptive)**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.remaining\_time = burst\_time

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

self.start\_time = -1 # To track when the process started execution

class SJFPreemptiveScheduler:

def \_\_init\_\_(self, processes):

self.processes = processes

self.gantt\_chart = []

def schedule(self):

time = 0

completed\_processes = 0

processes = sorted(self.processes, key=lambda x: x.arrival\_time) # Sort by arrival time

ready\_queue = []

current\_process = None

last\_time = 0

while completed\_processes < len(processes):

# Add all processes that have arrived by the current time to the ready queue

for p in processes:

if p.arrival\_time <= time and p not in ready\_queue and p.remaining\_time > 0:

ready\_queue.append(p)

# If there's a process in the ready queue, choose the one with the shortest remaining time

if ready\_queue:

ready\_queue.sort(key=lambda p: p.remaining\_time)

current\_process = ready\_queue[0]

# If the process has not started, set its start time

if current\_process.start\_time == -1:

current\_process.start\_time = time

current\_process.remaining\_time -= 1

time += 1

# If the process has finished execution

if current\_process.remaining\_time == 0:

current\_process.completion\_time = time

current\_process.turnaround\_time = current\_process.completion\_time - current\_process.arrival\_time

current\_process.waiting\_time = current\_process.turnaround\_time - current\_process.burst\_time

completed\_processes += 1

ready\_queue.remove(current\_process)

self.gantt\_chart.append((current\_process.pid, last\_time, time))

else:

time += 1 # If no process is in the ready queue, increment time

return time

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data

processes = [

Process(pid=1, arrival\_time=0, burst\_time=7),

Process(pid=2, arrival\_time=2, burst\_time=4),

Process(pid=3, arrival\_time=4, burst\_time=1),

Process(pid=4, arrival\_time=5, burst\_time=4)

]

scheduler = SJFPreemptiveScheduler(processes)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("SJF Preemptive (SRTF) - Gantt Chart")

IN C:

#include <stdio.h>

#include <limits.h>

struct Process {

int pid;

int bt;

int art;

};

void findWaitingTime(struct Process proc[], int n, int wt[]) {

int rt[n];

int i,j;

for ( i = 0; i < n; i++)

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

int check = 0;

while (complete != n) {

for ( j = 0; j < n; j++) {

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = 1;

}

}

if (check == 0) {

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

if (rt[shortest] == 0) {

complete++;

check = 0;

finish\_time = t + 1;

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

t++;

}

}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {

int i;

for ( i = 0; i < n; i++)

tat[i] = proc[i].bt + wt[i];

}

void findavgTime(struct Process proc[], int n) {

int i;

int wt[n], tat[n], total\_wt = 0,total\_tat = 0;

findWaitingTime(proc, n, wt);

findTurnAroundTime(proc, n, wt, tat);

printf(" P\t\t"

"BT\t\t"

"WT\t\t"

"TAT\t\t\n");

for ( i = 0; i < n; i++) {

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

printf(" %d\t\t"

"%d\t\t %d"

"\t\t %d\n", proc[i].pid,

proc[i].bt, wt[i], tat[i]);

}

printf("\nAverage waiting time = "

"%f", (float)total\_wt / (float)n);

printf("\nAverage turn around time = "

"%f", (float)total\_tat / (float)n);

}

int main() {

struct Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },

{ 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

1. **Round Robin**

import matplotlib.pyplot as plt

from collections import deque

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.remaining\_time = burst\_time

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

self.start\_time = -1 # To track when the process started execution

class RoundRobinScheduler:

def \_\_init\_\_(self, processes, time\_quantum):

self.processes = processes

self.time\_quantum = time\_quantum

self.gantt\_chart = []

def schedule(self):

time = 0

completed\_processes = 0

processes = sorted(self.processes, key=lambda x: x.arrival\_time) # Sort by arrival time

ready\_queue = deque()

last\_time = 0

while completed\_processes < len(processes):

# Add all processes that have arrived by the current time to the ready queue

for p in processes:

if p.arrival\_time <= time and p not in ready\_queue and p.remaining\_time > 0:

ready\_queue.append(p)

# If there's a process in the ready queue, select the next process to execute

if ready\_queue:

current\_process = ready\_queue.popleft()

# If the process has not started, set its start time

if current\_process.start\_time == -1:

current\_process.start\_time = time

# Calculate the time to execute the process

exec\_time = min(current\_process.remaining\_time, self.time\_quantum)

current\_process.remaining\_time -= exec\_time

time += exec\_time

# If the process has finished execution

if current\_process.remaining\_time == 0:

current\_process.completion\_time = time

current\_process.turnaround\_time = current\_process.completion\_time - current\_process.arrival\_time

current\_process.waiting\_time = current\_process.turnaround\_time - current\_process.burst\_time

completed\_processes += 1

self.gantt\_chart.append((current\_process.pid, last\_time, time))

else:

ready\_queue.append(current\_process)

last\_time = time

else:

time += 1 # If no process is in the ready queue, increment time

return time

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data

processes = [

Process(pid=1, arrival\_time=0, burst\_time=5),

Process(pid=2, arrival\_time=1, burst\_time=3),

Process(pid=3, arrival\_time=2, burst\_time=8),

Process(pid=4, arrival\_time=3, burst\_time=6)

]

time\_quantum = 4 # Set time quantum for Round Robin

scheduler = RoundRobinScheduler(processes, time\_quantum)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("Round Robin Scheduling - Gantt Chart")

IN C:

#include<stdio.h>

#include<conio.h>

void main()

{

int i, NOP, sum = 0, count = 0, y, quant, wt = 0, tat = 0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

int gantt[100], gantt\_time[100], gantt\_index = 0;

printf("Total number of processes in the system: ");

scanf("%d", &NOP);

y = NOP;

for(i = 0; i < NOP; i++)

{

printf("\nEnter the Arrival and Burst time of Process[%d]\n", i + 1);

printf("Arrival time: ");

scanf("%d", &at[i]);

printf("Burst time: ");

scanf("%d", &bt[i]);

temp[i] = bt[i];

}

printf("Enter the Time Quantum for the processes: ");

scanf("%d", &quant);

printf("\nProcess No\tBurst Time\tTurnaround Time\tWaiting Time");

for(sum = 0, i = 0; y != 0;)

{

if(temp[i] <= quant && temp[i] > 0)

{

sum += temp[i];

temp[i] = 0;

count = 1;

gantt[gantt\_index] = i + 1;

gantt\_time[gantt\_index] = sum;

gantt\_index++;

}

else if(temp[i] > 0)

{

temp[i] -= quant;

sum += quant;

gantt[gantt\_index] = i + 1;

gantt\_time[gantt\_index] = sum;

gantt\_index++;

}

if(temp[i] == 0 && count == 1)

{

y--;

printf("\nProcess[%d]\t\t%d\t\t%d\t\t%d", i + 1, bt[i], sum - at[i], sum - at[i] - bt[i]);

wt += sum - at[i] - bt[i];

tat += sum - at[i];

count = 0;

}

if(i == NOP - 1)

{

i = 0;

}

else if(at[i + 1] <= sum)

{

i++;

}

else

{

i = 0;

}

}

avg\_wt = (float)wt / NOP;

avg\_tat = (float)tat / NOP;

printf("\nAverage Turnaround Time: %.2f", avg\_tat);

printf("\nAverage Waiting Time: %.2f\n", avg\_wt);

printf("\nGantt Chart:\n|");

for(i = 0; i < gantt\_index; i++)

{

printf(" P%d |", gantt[i]);

}

printf("\n0");

for(i = 0; i < gantt\_index; i++)

{

printf(" %d", gantt\_time[i]);

}

getch();

}

1. **Priority (Non-preemptive)**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time, priority):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.priority = priority

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

self.start\_time = -1 # To track when the process started execution

class PriorityScheduler:

def \_\_init\_\_(self, processes):

self.processes = processes

self.gantt\_chart = []

def schedule(self):

time = 0

completed\_processes = 0

processes = sorted(self.processes, key=lambda x: (x.arrival\_time, x.priority)) # Sort by arrival time and priority

ready\_queue = []

last\_time = 0

while completed\_processes < len(processes):

# Add processes that have arrived by the current time to the ready queue

for p in processes:

if p.arrival\_time <= time and p not in ready\_queue:

ready\_queue.append(p)

# If there's a process in the ready queue, select the one with the highest priority

if ready\_queue:

ready\_queue.sort(key=lambda p: p.priority) # Sort by priority

current\_process = ready\_queue.pop(0) # Process with highest priority

# If the process has not started, set its start time

if current\_process.start\_time == -1:

current\_process.start\_time = time

# Process executes to completion

current\_process.completion\_time = time + current\_process.burst\_time

current\_process.turnaround\_time = current\_process.completion\_time - current\_process.arrival\_time

current\_process.waiting\_time = current\_process.turnaround\_time - current\_process.burst\_time

time = current\_process.completion\_time # Update time to when the current process finishes

completed\_processes += 1

self.gantt\_chart.append((current\_process.pid, last\_time, time))

else:

time += 1 # If no process is in the ready queue, increment time

last\_time = time

return time

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data: (PID, Arrival Time, Burst Time, Priority)

processes = [

Process(pid=1, arrival\_time=0, burst\_time=5, priority=2),

Process(pid=2, arrival\_time=1, burst\_time=3, priority=1),

Process(pid=3, arrival\_time=2, burst\_time=8, priority=4),

Process(pid=4, arrival\_time=3, burst\_time=6, priority=3)

]

scheduler = PriorityScheduler(processes)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tPriority\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.priority}\t\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("Priority Scheduling (Non-preemptive) - Gantt Chart")

1. **Priority (preemptive)**

def priority\_scheduling(processes):

n = len(processes)

processes.sort(key=lambda x: (x['arrival\_time'], x['priority'], x['pid']))

time = 0

completed = 0

gantt\_chart = []

ready\_queue = []

current\_process = None

while completed < n:

# Debugging: Display the time and process states

print(f"Time: {time}, Completed: {completed}, Ready Queue: {[p['pid'] for p in ready\_queue]}")

# Add processes to the ready queue based on arrival time

for process in processes:

if process['arrival\_time'] <= time and not process['is\_completed']:

if process not in ready\_queue:

ready\_queue.append(process)

# Check if the current process is completed

if current\_process and current\_process['burst\_time'] == 0:

current\_process['is\_completed'] = True

current\_process['finish\_time'] = time

current\_process['turnaround\_time'] = current\_process['finish\_time'] - current\_process['arrival\_time']

current\_process['waiting\_time'] = current\_process['turnaround\_time'] - current\_process['original\_burst\_time']

completed += 1

current\_process = None # Reset current process

# Select the next process to execute

if current\_process is None and ready\_queue:

ready\_queue.sort(key=lambda x: (x['priority'], x['arrival\_time'], x['pid']))

current\_process = ready\_queue.pop(0)

# Execute the current process or mark the CPU as idle

if current\_process:

gantt\_chart.append(current\_process['pid'])

current\_process['burst\_time'] -= 1

else:

gantt\_chart.append("Idle")

# Increment time

time += 1

# Safeguard: Prevent infinite loop by ensuring progress

if time > 100: # Timeout for debugging

print("ERROR: Timeout reached. Check the logic for infinite loops.")

break

# Optimize Gantt chart to remove consecutive duplicates

optimized\_gantt\_chart = [gantt\_chart[0]]

for i in range(1, len(gantt\_chart)):

if gantt\_chart[i] != gantt\_chart[i - 1]:

optimized\_gantt\_chart.append(gantt\_chart[i])

# Prepare the results for display

results = []

for p in processes:

results.append({

"PID": p["pid"],

"Priority": p["priority"],

"Arrival Time": p["arrival\_time"],

"Burst Time": p["original\_burst\_time"],

"Finish Time": p["finish\_time"],

"Turnaround Time": p["turnaround\_time"],

"Waiting Time": p["waiting\_time"],

})

return results, optimized\_gantt\_chart

# Input

process\_list = [

{"pid": "P1", "arrival\_time": 0, "burst\_time": 7, "priority": 2, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

{"pid": "P2", "arrival\_time": 2, "burst\_time": 4, "priority": 1, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

{"pid": "P3", "arrival\_time": 4, "burst\_time": 1, "priority": 3, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

{"pid": "P4", "arrival\_time": 5, "burst\_time": 4, "priority": 2, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

]

# Store original burst time

for p in process\_list:

p["original\_burst\_time"] = p["burst\_time"]

# Run simulation

results, gantt\_chart = priority\_scheduling(process\_list)

# Display Results

print("\nGantt Chart:", " -> ".join(gantt\_chart))

print("\nResults:")

for res in results:

print(res)