Assignment 03(1174066)

Analysis and Conclusion (Outputs and Metrics can be found in next sections)

The implementation of fitting algorithms, specifically the First Fit algorithm, in conjunction with CPU scheduling, provides a comprehensive understanding of how FIFO scheduling and memory management operate.

From the output screenshots, it is evident that each new job assigned to a partition in the main memory is allocated the first partition with a size large enough (greater than or equal to the job's requirements). Any remaining memory from the partition is added back into the main memory as a new partition.

The algorithm also checks for sufficient memory before assigning a job to the queue. Upon completion of a job's execution, a release function is invoked to make the partition available again, dissociating it from the job ID it was previously tied to.

Performance metrics over time indicate that as the algorithm processes more jobs, the number of memory fragments increases, leading to a rise in the number of partitions from 19 initially to 24. This fragmentation could become problematic over time, resulting in numerous small, unusable memory segments.

To enhance the implementation's performance, techniques such as compaction or more advanced memory management methods like paging could be employed. Additionally, the formula for calculating segmentation percentage over time should be revisited to provide more accurate metrics.

Outputs

```
Size ('128KB', 131072),process id None , availability False
Size ('129KB', 132096),process id None , availability True
Size ('895.00KB', 916480),process id None , availability True
Size ('2MB', 2097152),process id 3 , availability False
Size ('1.00MB', 1048576),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('8MB', 8388608),process id None , availability True
Size ('256KB', 262144),process id None , availability False
Size ('7MB', 7340032),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('256KB', 262144),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('128KB', 131072),process id None , availability True
Memory fragmentation percentage: 22.66987645348837%
At time 11: switching to Job 3
-----------------starting JOB 3------------------
Job status Running
At time 12: Running Job 3, 2.0 time units remaining.
At time 12: CPU available: False, Ready queue: [4]
At time 12: Job 5 arrived and added to ready queue.
Ready queue: [4, 5]
At time 13: Running Job 3, 1.0 time units remaining.
At time 13: CPU available: False, Ready queue: [4, 5]
Job status exit
-----------------finishing JOB 3------------------
At time 14: CPU available: True, Ready queue: [4, 5]
Difference is ('383.00KB', 392192)
>>>>>>>>>>>>>>>>>Memory available: True for job 4
>>>>>>>>>>>>>>>We have 23 Partitions and their status:
Size ('128KB', 131072),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('129KB', 132096),process id None , availability True
Size ('512KB', 524288),process id 4 , availability False
Size ('383.00KB', 392192),process id None , availability True
Size ('2MB', 2097152),process id None , availability True
```

```
---------------------------------FIFO START HERE---------------------------------------------
Difference is ('3.00MB', 3145728)
>>>>>>>>>>>>>>>>>Memory available: True for job 1
>>>>>>>>>>>>>>>We have 20 Partitions and their status:
Size ('128KB', 131072),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('1MB', 1048576),process id 1 , availability False
Size ('3.00MB', 3145728),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('8MB', 8388608),process id None , availability True
Size ('256KB', 262144),process id None , availability False
Size ('7MB', 7340032),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('256KB', 262144),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('128KB', 131072),process id None , availability True
Memory fragmentation percentage: 13.333333333333334%
At time 0: -----------------starting JOB 1------------------
Job status Running
At time 1: Running Job 1, 9 time units remaining.
At time 1: CPU available: False, Ready queue: []
At time 2: Running Job 1, 8 time units remaining.
At time 2: CPU available: False, Ready queue: []
At time 2: Job 2 arrived and added to ready queue.
Ready queue: [2]
At time 3: CPU available: False, Ready queue: [2]
At time 4: CPU available: False, Ready queue: [2]
At time 4: Job 3 arrived and added to ready queue.
Ready queue: [2, 3]
At time 5: CPU available: False, Ready queue: [2, 3]
At time 6: CPU available: False, Ready queue: [2, 3]
At time 7: CPU available: False, Ready queue: [2, 3]
At time 8: CPU available: False, Ready queue: [2, 3]
At time 8: Job 4 arrived and added to ready queue.
```

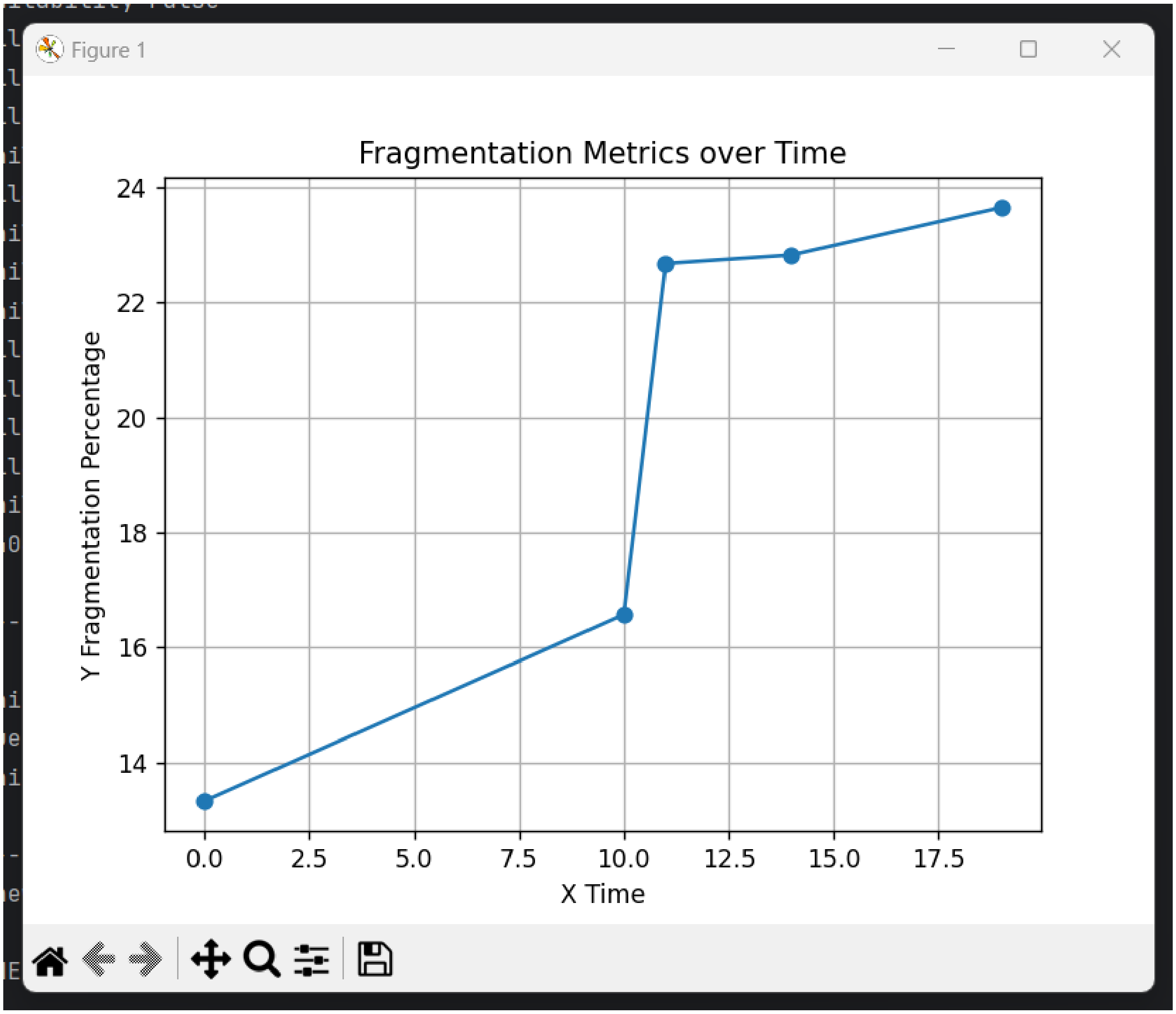
```
At time 8: Job 4 arrived and added to ready queue.
Ready queue: [2, 3, 4]
At time 9: CPU available: False, Ready queue: [2, 3, 4]
Job status exit
-----------------finishing JOB 1-----------------
At time 10: CPU available: True, Ready queue: [2, 3, 4]
Difference is ('895.00KB', 916480)
>>>>>>>>>>>>>>>>>Memory available: True for job 2
>>>>>>>>>>>>>>>We have 21 Partitions and their status:
Size ('128KB', 131072),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('129KB', 132096),process id 2 , availability False
Size ('895.00KB', 916480),process id None , availability True
Size ('3.00MB', 3145728),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('8MB', 8388608),process id None , availability True
Size ('256KB', 262144),process id None , availability False
Size ('7MB', 7340032),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('256KB', 262144),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('128KB', 131072),process id None , availability True
Memory fragmentation percentage: 16.574054731564654%
At time 10: switching to Job 2
-----------------starting JOB 2-----------------
Job status Running
At time 11: Running Job 2, 0 time units remaining.
Job status exit
-----------------finishing JOB 2-----------------
At time 11: CPU available: True, Ready queue: [3, 4]
Difference is ('1.00MB', 1048576)
>>>>>>>>>>>>>>>>>Memory available: True for job 3
>>>>>>>>>>>>>>>We have 22 Partitions and their status:
Size ('128KB', 131072),process id None , availability True
Size ('128KB', 131072),process id None , availability False
```

```
Size ('128KB', 131072),process id None , availability False
Size ('129KB', 132096),process id None , availability True
Size ('895.00KB', 916480),process id None , availability True
Size ('2MB', 2097152),process id 3 , availability False
Size ('1.00MB', 1048576),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('8MB', 8388608),process id None , availability True
Size ('256KB', 262144),process id None , availability False
Size ('7MB', 7340032),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('256KB', 262144),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('128KB', 131072),process id None , availability True
Memory fragmentation percentage: 22.66987645348837%
At time 11: switching to Job 3
-----------------starting JOB 3------------------
Job status Running
At time 12: Running Job 3, 2.0 time units remaining.
At time 12: CPU available: False, Ready queue: [4]
At time 12: Job 5 arrived and added to ready queue.
Ready queue: [4, 5]
At time 13: Running Job 3, 1.0 time units remaining.
At time 13: CPU available: False, Ready queue: [4, 5]
Job status exit
-----------------finishing JOB 3------------------
At time 14: CPU available: True, Ready queue: [4, 5]
Difference is ('383.00KB', 392192)
>>>>>>>>>>>>>>>>>Memory available: True for job 4
>>>>>>>>>>>>>>>We have 23 Partitions and their status:
Size ('128KB', 131072),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('129KB', 132096),process id None , availability True
Size ('512KB', 524288),process id 4 , availability False
Size ('383.00KB', 392192),process id None , availability True
Size ('2MB', 2097152),process id None , availability True
```

```
Size ('1.00MB', 1048576),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('8MB', 8388608),process id None , availability True
Size ('256KB', 262144),process id None , availability False
Size ('7MB', 7340032),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('256KB', 262144),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('128KB', 131072),process id None , availability True
Memory fragmentation percentage: 23.647513440860216%
At time 19: switching to Job 5
-----------------starting JOB 5-----------------
Job status Running
At time 20: Running Job 5, 1 time units remaining.
At time 20: CPU available: False, Ready queue: []
At time 21: Running Job 5, 0 time units remaining.
Job status exit
-----------------finishing JOB 5------------------
All jobs in ready queue need memory and no new jobs are coming in. Terminating.
Jobs in ready queue needing memory:
---------------------------------FIFO ENDS HERE------------------------------------------------
Total context switching time is 0
Total time is: 21
job is Job #1 - Arrival: 0.00, Execution Time: 10.00, Priority: 3, Queue: 1, Status: exit, Remaining time: 0, Exit time: 10
job is Job #2 - Arrival: 2.00, Execution Time: 1.00, Priority: 2, Queue: 1, Status: exit, Remaining time: 0, Exit time: 11
job is Job #3 - Arrival: 4.00, Execution Time: 3.00, Priority: 1, Queue: 1, Status: exit, Remaining time: 0.0, Exit time: 14
job is Job #4 - Arrival: 8.00, Execution Time: 5.00, Priority: 5, Queue: 1, Status: exit, Remaining time: 0, Exit time: 19
job is Job #5 - Arrival: 12.00, Execution Time: 2.00, Priority: 4, Queue: 1, Status: exit, Remaining time: 0, Exit time: 21
```

```
Size ('2MB', 2097152),process id None , availability True
Size ('1.00MB', 1048576),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('8MB', 8388608),process id None , availability True
Size ('256KB', 262144),process id None , availability False
Size ('7MB', 7340032),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('512KB', 524288),process id None , availability False
Size ('256KB', 262144),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('1MB', 1048576),process id None , availability True
Size ('1MB', 1048576),process id None , availability False
Size ('128KB', 131072),process id None , availability True
Memory fragmentation percentage: 22.817595108695652%
At time 14: switching to Job 4
-----------------starting JOB 4------------------
Job status Running
At time 15: Running Job 4, 4 time units remaining.
At time 15: CPU available: False, Ready queue: [5]
At time 16: Running Job 4, 3 time units remaining.
At time 16: CPU available: False, Ready queue: [5]
At time 17: CPU available: False, Ready queue: [5]
At time 18: CPU available: False, Ready queue: [5]
Job status exit
-----------------finishing JOB 4------------------
At time 19: CPU available: True, Ready queue: [5]
Difference is ('256.00KB', 262144)
>>>>>>>>>>>>>>>>>Memory available: True for job 5
>>>>>>>>>>>>>>>We have 24 Partitions and their status:
Size ('128KB', 131072),process id None , availability True
Size ('128KB', 131072),process id None , availability False
Size ('129KB', 132096),process id None , availability True
Size ('256KB', 262144),process id 5 , availability False
Size ('256.00KB', 262144),process id None , availability True
Size ('383.00KB', 392192),process id None , availability True
Size ('2MB', 2097152),process id None , availability True
Size ('1.00MB', 1048576),process id None , availability True
Size ('2MB', 2097152),process id None , availability False
```

Performance Metrics



Overview of the code

Main.py: The main function initializes job objects, manages their memory allocation using the MemorySizes class, schedules them with a FIFO scheduler, collects metrics, and visualizes performance and memory fragmentation using the Plot class.

FIFO Class (FIFOScheduler.py): Implements a First-In-First-Out scheduling algorithm, managing job completion and context switching, and calculates various performance metrics encapsulated in a SchedulerMetrics instance.

Job Class (Job.py): Represents a job with attributes like ID, arrival time, execution time, priority, status, and memory requirements, and uses an enumeration for job states.

LinearQueue Class (LinearQueue.py): Implements a basic FIFO queue structure for job scheduling, providing methods to add, remove, and reorder jobs based on memory needs.

MemoryManager Class (MemoryManager.py): Manages memory allocation and deallocation for jobs using algorithms like first-fit, best-fit, and worst-fit, tracking memory usage and fragmentation.

SchedulerMetrics Class (SchedulerMetrics.py): Encapsulates various performance metrics of a scheduler, such as average turnaround time, waiting time, CPU utilization, and throughput, for systematic analysis and comparison.

MemorySizes Class (MemorySizes.py): Provides functionality to parse and compare memory sizes in human-readable formats, converting them to bytes and computing differences.

Plot Class (Plot.py): Generates visualizations for scheduling metrics and memory fragmentation data, illustrating various aspects of the scheduling process for analysis and optimization

Memory Manager Python Code

Main.py

```
from FIFOScheduler import FIFO
from Job import Job, Status
from MemorySizes import MemorySizes
from Plot import Plot



def main():
    MS = MemorySizes()
    jobs = []
    job1 = Job(1, 0.0, 10, 3, 1, Status.CREATED, MS.get_memory_size("1MB"))
    jobs.append(job1)
    job2 = Job(2, 2, 1, 2, 1, Status.CREATED, MS.get_memory_size("129KB"))
    jobs.append(job2)
    job3 = Job(3, 4, 3.0, 1, 1, Status.CREATED, MS.get_memory_size("2MB"))
    jobs.append(job3)
    job4 = Job(4, 8, 5, 5, 1, Status.CREATED , MS.get_memory_size("512KB"))
    jobs.append(job4)
    job5 = Job(5, 12, 2, 4, 1, Status.CREATED, MS.get_memory_size("256KB"))
    jobs.append(job5)

    # Create separate lists for job 2 and job 3
    # jobs_2 = jobs.copy()
    # jobs_3 = jobs_2.copy()

    jobs_2 = []
    job1 = Job(1, 0.0, 10, 3, 1, Status.CREATED, MS.get_memory_size("1MB"))
    jobs_2.append(job1)
    job2 = Job(2, 2, 1, 2, 1, Status.CREATED, MS.get_memory_size("129KB"))
    jobs_2.append(job2)
    job3 = Job(3, 4, 3.0, 1, 1, Status.CREATED, MS.get_memory_size("9MB"))
    jobs_2.append(job3)
    job4 = Job(4, 8, 5, 5, 1, Status.CREATED , MS.get_memory_size("512KB"))
    jobs_2.append(job4)
    job5 = Job(5, 12, 2, 4, 1, Status.CREATED, MS.get_memory_size("256KB"))
    jobs_2.append(job5)



    # print(f"1. First Fit\n2. Best Fit\n3. Worst Fit")
    # fitting_algorithm = int(input(f"Selecting fitting algorithm from above(Enter
number): "))
```

```python
    # context_switching_time = int(input(f"Enter Context Switching Time(should be integer): "))
    # context_switching_time = 0

    metrics_list = []

    content_switching_times = [0, 1 , 2]

    # for context_switching_time in content_switching_times:
    #     fifo = FIFO()
    #     metrics = fifo.run_scheduler(jobs, context_switching_time)
    #     metrics_list.extend(metrics)

    fifo = FIFO()
    metrics = fifo.run_scheduler(jobs, content_switching_times[0])
    metrics_list.extend(metrics)

    # fifo1 = FIFO()
    # metrics = fifo1.run_scheduler(jobs_2, content_switching_times[0])
    # metrics_list.extend(metrics)

    # fifo2 = FIFO()
    # metrics = fifo2.run_scheduler(jobs_3, content_switching_times[2])
    # metrics_list.extend(metrics)

    # fifo = FIFO()
    # metrics_list = fifo.run_scheduler(jobs, context_switching_time)

    # print(f"metrics_list: {metrics_list[0]}")

    # plotting
    my_plot = Plot()  # Create an instance ofPlot.
    # my_plot.plot_optimization(metrics_list)

    my_plot.plot_fragmentation_metrics(fifo.fragmentation_metrics)


if __name__ == "__main__":
    main()
```

Fifoscheduler.py

```python
import time
from Job import Status, Job
from LinearQueue import LinearQueue
from MemoryManager import MemoryManager
from SchedulerMetrics import SchedulerMetrics
class FIFO(LinearQueue):
    def __init__(self):
        super().__init__()
        self.fifo_metrics = None
        self.aging_threshold = 3  # Set aging threshold
        self.tracked_jobs = []  # for tracking number of completed jobs
        self.total_context_switching_time = 0
        self.total_time_for_all_jobs_to_run = 0
        self.fragmentation_metrics = []


    def run_scheduler(self, jobs, context_switching_time):
        self.process_jobs_Fifo(jobs, context_switching_time)
        # print(f"FIFO avg_turn_t is {self.get_avg_turn_t()}")
        # print(f"FIFO avg_wt is {self.get_avg_wt()}")
        # print(f"FIFO cpu utilization is {self.get_cpu_utilization()}%")
        # print(f"FIFO Maximum turnaround time is {self.get_max_turn_around_time()}")
        # print(f"FIFO Maximum wait time {self.get_max_waiting_time()}")
        # print(f"FIFO CPU throughput is {self.get_cpu_throughput()}")


        # Creating an instance of SchedulerMetrics for FIFO
        # Collect metrics
        avg_turnaround_times = self.get_avg_turn_t()
        avg_waiting_times = self.get_avg_wt()
        cpu_utilizations = self.get_cpu_utilization()
        max_turnaround_times = self.get_max_turn_around_time()
        max_waiting_times = self.get_max_waiting_time()
        cpu_throughputs = self.get_cpu_throughput()


        # Create an instance of SchedulerMetrics with computed values
        self.fifo_metrics = SchedulerMetrics(
            name="FIFO",
            avg_turn_t=avg_turnaround_times,
```

```
            avg_wt=avg_waiting_times,
            cpu_utilization=cpu_utilizations,
            max_turnaround_time=max_turnaround_times,
            max_wait_time=max_waiting_times,
            cpu_throughput=cpu_throughputs
        )

        # Adding the metrics to a list
        metrics_list = [self.fifo_metrics]
        return metrics_list

    def process_jobs_Fifo(self, jobs, context_switching_time):
        current_time = 0
        running_job = None
        cpu_available = True
        MM = MemoryManager(1)

        # print(f"jobs are {jobs}")
        print(f"---------------------------------FIFO START HERE-------------------------
-----------------------")
        # Process the first job outside the loop
        if jobs:
            current_job = jobs.pop(0)

            # Check if we have enough memory from a memory manager
            memory_is_available = MM.first_fit(current_job)
            print(f">>>>>>>>>>>>>>>>Memory available: {memory_is_available} for job
{current_job.job_number}")
            if memory_is_available:
                MM.get_partitions()
                print(f"Memory fragmentation percentage:
{MM.calculate_fragmentation_percentage()}%")

self.fragmentation_metrics.append((current_time,MM.calculate_fragmentation_percentage()))
                current_job.memory_needed = False
                current_job.status = Status.RUNNING
                running_job = current_job
                cpu_available = False
                print(f"At time {current_time}: -----------------starting JOB
{running_job.job_number}-----------------")
                print(f"Job status {running_job.status.value}")
                minimum_execution_time = 1
                time.sleep(minimum_execution_time)
                running_job.remaining_time -= minimum_execution_time  # tracking
remaining time
                running_job.running_time += minimum_execution_time
                print(
                    f"At time {current_time + 1}: Running Job {running_job.job_number},
```

```
{running_job.remaining_time} time units remaining.")

                if running_job.remaining_time <= 0:
                    MM.release_memory(running_job.job_number)  # releasing memory when
job finishes executing
                    running_job.status = Status.EXIT
                    self.tracked_jobs.append(running_job)
                    cpu_available = True
                    print(f"Job status {running_job.status.value}")
                    print(f"-----------------finishing JOB {running_job.job_number}------
------------")
                    running_job = None

                current_time += 1

            elif not memory_is_available:
                current_job.memory_needed = True
                print(f">>>>>>>>>>>>>>>>>>>>Not enough memory for job
{current_job.job_number}")
                if not any(job.job_number == current_job.job_number for job in
self.ready_queue):
                    self.enqueue(current_job)  # add to ready queue to wait for memory
                    print(
                        f"At time {current_time}: job {current_job.job_number} added to
ready queue to wait for memory. "
                        f"Ready queue: {[job.job_number for job in self.ready_queue]}")

        while jobs or not self.is_empty() or running_job:
            # Print the current time and queue status
            print(
                f"At time {current_time}: CPU available: {cpu_available}, "
                f"Ready queue: {[job.job_number for job in self.ready_queue]}")

            # Add arriving jobs to the queue
            while jobs and jobs[0].arrival_time <= current_time:
                current_job = jobs.pop(0)
                self.enqueue(current_job)
                print(f"At time {current_time}: Job {current_job.job_number} arrived and
added to ready queue.")
                print(f"Ready queue: {[job.job_number for job in self.ready_queue]}")

            # Check if memory is available for any job in the ready queue
            if cpu_available:
                running_job = None
                for i in range(len(self.ready_queue)):
                    potential_job = self.ready_queue[i]
                    memory_is_available = MM.first_fit(potential_job)
                    print(
```

```
                        f">>>>>>>>>>>>>>>>>Memory available: {memory_is_available} "
                        f"for job {potential_job.job_number}")
                    if memory_is_available:
                        running_job = potential_job
                        MM.get_partitions()
                        print(f"Memory fragmentation percentage:
{MM.calculate_fragmentation_percentage()}%")
                        self.fragmentation_metrics.append((current_time,
MM.calculate_fragmentation_percentage()))
                        running_job.memory_needed = False
                        del self.ready_queue[i]
                        break
                    elif not memory_is_available:
                        potential_job.memory_needed = True
                        print(f">>>>>>>>>>>>>>>>>>>>Not enough memory for job
{potential_job.job_number}")

                if running_job:
                    current_time += context_switching_time
                    self.total_context_switching_time += context_switching_time
                    print(f"At time {current_time}: switching to Job
{running_job.job_number}")
                    running_job.status = Status.RUNNING
                    cpu_available = False
                    print(f"-----------------starting JOB {running_job.job_number}-------
-----------")
                    print(f"Job status {running_job.status.value}")

            # Simulate job execution
            if running_job:
                minimum_execution_time = 1
                time.sleep(minimum_execution_time)
                running_job.remaining_time -= minimum_execution_time  # tracking
remaining time
                running_job.running_time += minimum_execution_time
                if running_job.running_time < self.aging_threshold:
                    print(
                        f"At time {current_time + 1}: Running Job
{running_job.job_number}, {running_job.remaining_time} time units remaining.")

                if running_job.remaining_time <= 0:
                    MM.release_memory(running_job.job_number)  # releasing memory when
job finishes executing
                    running_job.status = Status.EXIT
                    running_job.exit_time = current_time + 1  # Update exit time
                    self.tracked_jobs.append(running_job)
                    cpu_available = True
                    print(f"Job status {running_job.status.value}")
                    print(f"-----------------finishing JOB {running_job.job_number}------
```

```
------------")
                    running_job = None

                    # Check if all jobs in ready queue need memory and no new jobs are
coming in
                    # If so, terminate the simulation
                    if all(job.memory_needed for job in self.ready_queue) and not jobs:
                        print("All jobs in ready queue need memory and no new jobs are
coming in. Terminating.")
                        print("Jobs in ready queue needing memory:")
                        for job in self.ready_queue:
                            print(f"Job {job.job_number} needs {job.memory_size}")
                        break

            current_time += 1

        print(f"---------------------------------FIFO ENDS HERE--------------------------
----------------------")
        print(f"Total context switching time is {self.total_context_switching_time}")
        self.total_time_for_all_jobs_to_run = self.tracked_jobs[-1].exit_time
        print(f"Total time is: {self.total_time_for_all_jobs_to_run}")

    def get_avg_wt(self):
        total_waiting_time = 0
        for job in self.tracked_jobs:
            total_waiting_time += (job.exit_time - job.arrival_time -
job.actual_execution_time)
            print(f"job is {job}")
        average_waiting_time = total_waiting_time / len(self.tracked_jobs)
        return average_waiting_time

    def get_avg_turn_t(self):
        total_turnaround_time = 0
        for job in self.tracked_jobs:
            total_turnaround_time += (job.exit_time - job.arrival_time)
            # print(f"job is {job}")
        average_turnaround_time = total_turnaround_time / len(self.tracked_jobs)
        return average_turnaround_time

    def get_cpu_utilization(self):
        cpu_utilization = ((self.total_time_for_all_jobs_to_run -
self.total_context_switching_time) / (
            self.total_time_for_all_jobs_to_run)) * 100
        return cpu_utilization

    def get_max_turn_around_time(self):
        max_turnaround_time = 0
        for job in self.tracked_jobs:
```

```
            turnaround_time = job.exit_time - job.arrival_time
            if turnaround_time > max_turnaround_time:
                max_turnaround_time = turnaround_time
        return max_turnaround_time

    def get_max_waiting_time(self):
        max_waiting_time = 0
        for job in self.tracked_jobs:
            waiting_time = job.exit_time - job.arrival_time - job.actual_execution_time
            if waiting_time > max_waiting_time:
                max_waiting_time = waiting_time
        return max_waiting_time

    def get_cpu_throughput(self):
        throughput = len(self.tracked_jobs) / self.total_time_for_all_jobs_to_run
        return throughput

    def plot_optimization(self, jobs):
        pass
```

Job.py

```
from enum import Enum
import random

# Enum for job status
class Status(Enum):
    CREATED = "created"
    READY = "ready"
    RUNNING = "Running"
    EXIT = "exit"

# Class to represent a job
class Job:
    def __init__(self, job_number, arrival_time, actual_execution_time, priority,
queue_number, status , memory_size):
        self.job_number = job_number
        self.arrival_time = arrival_time
        self.actual_execution_time = actual_execution_time
        self.priority = priority
        self.queue_number = queue_number
        self.status = status
        self.remaining_time = actual_execution_time  # Initialize remaining time with
actual execution time
        self.running_time = 0  # Track running time
        self.exit_time = 0
```

```python
        self.turn_around_time = 0
        self.memory_size = memory_size
        self.memory_needed = False

    def __str__(self):
        return (f"Job #{self.job_number} - Arrival: {self.arrival_time:.2f}, "
                f"Execution Time: {self.actual_execution_time:.2f}, Priority:
{self.priority}, "
                f"Queue: {self.queue_number}, Status: {self.status.value}, Remaining
time: {self.remaining_time}, "
                f"Exit time: {self.exit_time}")
```

MemoryManager.py

```python
from enum import Enum
from MemorySizes import MemorySizes

class FittingAlgorithm(Enum):
    FIRST_FIT = 1
    BEST_FIT = 2
    WORST_FIT = 3



class Partition:
    def __init__(self, partition_size):
        self.process_id = None
        self.partition_size = partition_size
        self.is_available = True

    def __str__(self):
        return f"Partition(size={self.partition_size} bytes,
process_id={self.process_id}, is_available={self.is_available})"

    def get_partition_size(self):
        return self.partition_size



class MemoryManager:
    def __init__(self, fitting_algorithm):
        MS = MemorySizes()
        self.main_memory_size = MS.get_memory_size("32MB")
        self.fitting_algorithm = fitting_algorithm
        self.partitions = []  # contains partitions
        self.fitting_algorithm = fitting_algorithm
```

```
        self.total_fragments_memory = 0

        # 19 partition sizes that sum to 32MB
        partition_sizes = [
            "128KB",
            "128KB",
            "4MB",
            "2MB",
            "8MB",
            "256KB",
            "7MB",
            "2MB",
            "1MB",
            "512KB",
            "1MB",
            "512KB",
            "256KB",
            "128KB",
            "1MB",
            "1MB",
            "1MB",
            "1MB",
            "128KB"
        ]

        # Create and add partitions and alternatively set them as unavailable and
available
        is_available = True
        for size in partition_sizes:
            partition_size = MS.get_memory_size(size)
            partition = Partition(partition_size)
            partition.is_available = is_available
            self.partitions.append(partition)
            is_available = not is_available

    def check_memory(self, job):
        if self.fitting_algorithm == FittingAlgorithm.FIRST_FIT:
            self.first_fit(job)

    def first_fit(self, job):
        MS = MemorySizes()
        for i, partition in enumerate(self.partitions):
            difference = MS.get_memory_size_difference(partition.partition_size[0],
job.memory_size[0])
            if partition.is_available and MS.compare(partition.partition_size[0],
                                                     job.memory_size[0]) == 1:  # getting
first available partition
                print(f"Difference is {difference}")
```

```python
                partition.is_available = False
                partition.process_id = job.job_number
                partition.partition_size = job.memory_size

                # Add new partition with the size of the difference right after the
current partition
                if difference[1] > 0:
                    new_partition = Partition(partition_size=difference)
                    self.partitions.insert(i + 1, new_partition)
                    self.total_fragments_memory += difference[1]

                return True
        return False

    def release_memory(self, job_number):
        for partition in self.partitions:
            if partition.process_id == job_number:
                partition.is_available = True
                partition.process_id = None

    def get_partitions(self):
        print(f">>>>>>>>>>>>>>>We have {len(self.partitions)} Partitions and their
status: ")

        for partition in self.partitions:
            print(
                f"Size {partition.partition_size},process id {partition.process_id} , "
                f"availability {partition.is_available}")

    # Fragmentation Percentage = (Σ Size of free blocks added) / Total  free memory * 100
    # how many fragments we are adding on top of free memory
    def calculate_fragmentation_percentage(self):
        free_memory = sum(partition.partition_size[1] for partition in self.partitions if
partition.is_available)

        fragmentation_percentage = (self.total_fragments_memory / free_memory) * 100

        # print totaml free memory and total fragmentation memory
        # print(f"Total free memory: {free_memory} bytes")
        # print(f"Total fragmentation memory: {self.total_fragments_memory} bytes")
        return fragmentation_percentage
```

MemorySizes.py

```python
import re


class MemorySizes:
    units = {
        'KB': 1024,
        'MB': 1024 ** 2,
    }

    def __init__(self):
        pass

    def get_memory_size(self, size_str):
        pattern = re.compile(r'(\d+(\.\d+)?)([a-zA-Z]+)')
        match = pattern.match(size_str.strip().upper())
        if not match:
            raise ValueError(f"Invalid memory size format: {size_str}")
        size, _, unit = match.groups()
        size = float(size)
        if unit not in self.units:
            raise ValueError(f"Invalid memory unit: {unit}")
        return size_str, int(size * self.units[unit])

    def get_memory_size_difference(self, size1, size2):
        size1_bytes = self.get_memory_size(size1)[1]
        size2_bytes = self.get_memory_size(size2)[1]
        difference = size1_bytes - size2_bytes
        if abs(difference) < 1024:
            difference_str = f"{difference} bytes"
        elif abs(difference) < 1048576:
            difference_str = f"{difference / 1024:.2f}KB"
        else:
            difference_str = f"{difference / 1048576:.2f}MB"
        size1_str, size1_bytes = self.get_memory_size(size1)
        size2_str, size2_bytes = self.get_memory_size(size2)
        return difference_str, difference

    def compare(self, size1, size2):
        difference = self.get_memory_size_difference(size1, size2)[1]
        if difference < 0:
            return -1
        elif difference > 0 or difference == 0:
            return 1
        else:
            return 0
```

Plot.py

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

class Plot:
    def __init__(self):
        pass

    # Function to plot optimization
    def plot_optimization(self, metrics_list):
        # Convert the list of SchedulerMetrics to a DataFrame
        data = [
            [metric.name, metric.avg_turn_t, metric.avg_wt, metric.cpu_utilization,
metric.max_turnaround_time,
             metric.max_wait_time, metric.cpu_throughput]
            for metric in metrics_list
        ]

        df = pd.DataFrame(data, columns=["Scheduler", "Avg Turnaround Time", "Avg Waiting
Time", "CPU Utilization (%)",
                                         "Max Turnaround Time", "Max Wait Time", "CPU
Throughput"])

        # Plot the data
        ax = df.plot(kind="bar", figsize=(10, 6))
        plt.title("Scheduler Metrics")
        plt.ylabel("Metric")
        plt.legend(loc="upper right")
        plt.grid(axis="y")

        # Add value annotations to each bar
        for p in ax.patches:
            ax.annotate(f"{p.get_height():.2f}", (p.get_x() + p.get_width() / 2.,
p.get_height()),
                        ha='center', va='center', fontsize=10, color='black', xytext=(0,
5),
                        textcoords='offset points')

        # Display plot
        plt.show()

    def plot_fragmentation_metrics(self, metrics_list):

        # Extract x and y values from the data list
```

```
        x_values = [item[0] for item in metrics_list]
        y_values = [item[1] for item in metrics_list]

        # Plot the data
        plt.plot(x_values, y_values, marker='o')  # You can choose a different marker if
needed
        plt.xlabel('X Time')
        plt.ylabel('Y Fragmentation Percentage')
        plt.title('Fragmentation Metrics over Time')
        plt.grid(True)
        plt.show()

    # Call the plot_optimization function with the metrics list
    # plot_optimization(metrics_list)
```

LinearQueue.py

```
# Class representing a linear queue
from collections import deque

from Job import Status



class LinearQueue:
    def __init__(self):
        self.ready_queue = deque()
        self.aging_threshold = 0  # Default aging threshold

    def enqueue(self, job):
        job.status = Status.READY
        self.ready_queue.append(job)
        # if job.memory_needed:
        #     self.ready_queue.appendleft(job)
        # else:
        #     self.ready_queue.append(job)
        # # self.ready_queue.append(job)
        # self.update_queue_for_memory()
```

```
    def dequeue(self):
        if self.is_empty():
            return None
        return self.ready_queue.popleft()

    def is_empty(self):
        return len(self.ready_queue) == 0

    def update_queue_for_memory(self):
        non_memory_jobs = [job for job in self.ready_queue if not job.memory_needed]
        for job in non_memory_jobs:
            self.ready_queue.remove(job)
            self.ready_queue.appendleft(job)
```

SchedulerMetrics.py

```
class SchedulerMetrics:
    def __init__(self, name, avg_turn_t, avg_wt, cpu_utilization, max_turnaround_time,
max_wait_time, cpu_throughput):
        self.name = name
        self.avg_turn_t = avg_turn_t
        self.avg_wt = avg_wt
        self.cpu_utilization = cpu_utilization
        self.max_turnaround_time = max_turnaround_time
        self.max_wait_time = max_wait_time
        self.cpu_throughput = cpu_throughput

    # print values
    def __str__(self):
        return f"Scheduler name: {self.name}\n" \
               f"Average Turnaround time: {self.avg_turn_t}\n" \
               f"Average Wait time: {self.avg_wt}\n" \
               f"CPU Utilization: {self.cpu_utilization}\n" \
               f"Maximum Turnaround Time: {self.max_turnaround_time}\n" \
```

```
            f"Maximum Wait Time: {self.max_wait_time}\n" \
            f"CPU Throughput: {self.cpu_throughput}\n"
```