CS 4310 Operating Systems

Project #1 Simulating Job Scheduler and Performance Analysis

Prof. Gilbert Young

Devin Khun

Due Date: October 30, 2025

Important:

- *Please read this document completely before you start coding.*
- *Also, please read the submission instructions (provided at the end of this document) carefully before submitting the project.*

Project #1 Description:

Simulating Job Scheduler of the Operating Systems by programming the following four scheduling algorithms that we covered in the class:

(a) First-Come-First-Serve (FCFS)

(b) Shortest-Job-First (SJF)

(c) Round-Robin with Time Slice = 2 (RR-2)

(d) Round-Robin with Time Slice = 5 (RR-5)

You can use either Java or your choice of programming language for the implementation. The objective of this project is to help student understand how above four job scheduling algorithms operates by implementing the algorithms, and conducting a performance analysis of them based on the performance measure of their average turnaround times (of all jobs) for each scheduling algorithm using multiple inputs. Output the details of each algorithm's execution. You need to show which jobs are selected at what times as well as their starting and stopping burst values. You can choose your display format, for examples, you can display the results of each in Schedule Table or Gantt Chart format (as shown in the class notes). The project will be divided into three parts (phases) to help you to accomplish above tasks in in a systematic and scientific fashion: Design and Testing, Implementation, and Performance Analysis.

The program will read process burst times from a file (job.txt) – this file will be generated by you. Note that you need to generate multiple testing cases (with inputs of 5 jobs, 10 jobs and 15 jobs). A sample input file of five jobs is given as follows (burst time in ms):

```
[Begin of job.txt]
Job1
7
Job2
```

```
18
Job3
10
Job4
4
Job5
12
[End of job.txt]
```

Note: you can assume that

(1) There are no more than 30 jobs in the input file (job.txt).

(2) Processes arrive in the order they are read from the file for FCFS, RR-2 and RR-5.

(3) All jobs arrive at time 0.

(4) FCFS uses the order of the jobs, Job1, Job2, Job3, ...

You can implement the algorithms in your choice of data structures based on the program language of your choice. Note that you always try your best to give the most efficient program for each problem. The size of the input will be limited to be within 30 jobs.

Submission Instructions:

turn in the following @Canvas after the completion of all three parts, part 1, part 2, and part 3

(1) four program files (your choice of programming language with proper documentation)

(2) this document (complete all the answers)

Project 1

Part 1: Design & Testing (30 points)

(a) Design the program by providing pseudocode or flowchart for each CPU scheduling algorithm.

First-Come-First-Serve (FCFS)

```
Input: jobs = [(Job1, burst1), (Job2, burst2), ...]
Output: log, turnaround
begin
    time ← 0;
    log ← [];
    foreach (job, burst) in jobs do
        start ← time;
        remainingBefore ← burst;
        ran ← burst;
        time ← time + ran;
        remainingAfter ← 0;
        append (job, start, ran, remainingBefore, remainingAfter) to log;
        finishTime[job] ← time;
    end
    foreach job do
        turnaround[job] ← finishTime[job] - 0;
    end
end
```

Algorithm 1: First-Come-First-Serve (FCFS)

Shortest-Job-First (SJF)

```
Input: jobs = [(Job1, burst1), (Job2, burst2), ...]
Output: log, turnaround
begin
    time ← 0;
    log ← [];
    remaining[job] ← burst;
    undone ← set of all jobs;
    while undone not empty do
        pick job* in undone with minimal remaining[job*];
        start ← time;
        remainingBefore ← remaining[job*];
        ran ← remainingBefore;
        time ← time + ran;
        remainingAfter ← 0;
        append (job*, start, ran, remainingBefore, remainingAfter) to log;
        finishTime[job*] ← time;
        remove job* from undone;
    end
    foreach job do
        turnaround[job] ← finishTime[job] - 0;
    end
end
```

Algorithm 2: Shortest-Job-First (SJF)

Round-Robin with Time Slice = 2 (RR-2)

```
Input: jobs = [(Job1, burst1), (Job2, burst2), ...], QUANTUM = 2
Output: log, turnaround
begin
    time ← 0;
    log ← [];
    remaining[job] ← burst;
    Q ← empty queue;
    foreach (job, _) in jobs do
        enqueue Q, job
    end
    while Q not empty do
        job ← dequeue Q if remaining[job] = 0: continue;
        start ← time;
        remainingBefore ← remaining[job];
        ran ← min(QUANTUM, remainingBefore);
        time ← time + ran;
        remainingAfter ← remainingBefore - ran;
        append (job, start, ran, remainingBefore, remainingAfter) to log;
        remaining[job] ← remainingAfter;
        if remainingAfter > 0 then
            enqueue Q, job;
        end
        else
            finishTime[job] ← time;
        end
    end
    foreach job do
        turnaround[job] ← finishTime[job] - 0;
    end
end
```

Algorithm 3: Round-Robin with Time Slice = 2 (RR-2)

Round-Robin with Time Slice = 5 (RR-5)

```
Input: jobs = [(Job1, burst1), (Job2, burst2), ...], QUANTUM = 5
Output: log, turnaround
begin
    Same steps as RR-2 but with QUANTUM = 5
end
```

Algorithm 4: Round-Robin with Time Slice = 5 (RR-5)

(b) Design the program correctness testing cases. Give at least 3 testing cases to test your program, and give the expected correct **average turnaround time** (for each testing case)

in order to test the correctness of each algorithm.

Table 1: Input Testing Cases

Testing case #	`Input (table of jobs with its job# and length`	**Expected output for FCFS (✓ if Correct after testing in Part 2)**	**Expected output for SJF (✓ if Correct after testing in Part 2)**	**Expected output for RR-2 (✓ if Correct after testing in Part 2)**	**Expected output for RR-5 (✓ if Correct after testing in Part 2)**
1 (5 jobs)	`(Job1, 7), (Job2, 18), (Job3, 10), (Job4, 4), (Job5, 12)`	31.4✓	24✓	36.4✓	36✓
2 (10 jobs)	`(Job1, 3), (Job2, 8), (Job3, 2), (Job4, 14), (Job5, 7), (Job6, 5), (Job7, 9), (Job8, 4), (Job9, 12), (Job10, 6)`	36.1✓	28.2✓	46.1✓	43.3✓
3 (15 jobs)	`(Job1, 10), (Job2, 1), (Job3, 8), (Job4, 20), (Job5, 5), (Job6, 3), (Job7, 12), (Job8, 7), (Job9, 15), (Job10, 4), (Job11, 9), (Job12, 11), (Job13, 6), (Job14, 2), (Job15, 13)`	67.33✓	45.8✓	79.6✓	78✓

(c) Design testing strategy for the programs. Discuss about how to generate and structure the randomly generated inputs for experimental study later in Part 3.

Hint 1: To study the performance evaluation of the four job scheduling algorithms, this project will use three different input sizes, 5 jobs, 10 jobs and 15 jobs. It is the easiest to use a random number generator for generating the inputs. Note that you need to decide the maximum value of job length (use at least 20). However, student should store each data set in various sizes and use the same data set for each job scheduling algorithm.

The performance of average Turnaround Time of each input data size (5 jobs, 10 jobs and 15 jobs) can be calculated after an experiment is conducted in 20 trail (with 20 input sets of jobs). We can denote the results as the set X which contains the 20 computed Turnaround Times of 20 trails, where $X = \{x_1, x_2, x_3, \ldots, x_{20}\}$, from the simulator.

For each data size (5 jobs, 10 jobs, and 15 jobs):

$$\text{Average Turnaround Time} = \frac{\sum_{i=1}^{20} x_i}{20}$$

The student should decide the maximum value of the job length (at least 20).

Testing Strategy Design

We need to verify the correctness and evaluate the performance (average turnaround time) of each scheduling algorithm under consistent conditions. To do that, we'll create multiple controlled random inputs (job sets) of different sizes (5 jobs, 10 jobs, and 15 jobs) and feed the same inputs to all four algorithms (FCFS, SJF, RR-2, RR-5).

Generate Random Inputs

1. Job Length Range: Each job's burst time will be randomly generated using a uniform distribution:

   ```
   burst_time = random integer between 1 and MAX_BURST
   where MAX_BURST >= 20 (for example, 1-25 ms or 1-30 ms).
   ```

2. Input Sizes: 5 jobs (small-scale), 10 jobs (medium-scale), and 15 jobs (large-scale) This ensures consistency across algorithms and test sizes.

3. Data File Structure (job.txt)

   ```
   Job1
   <burst_time>
   Job2
   <burst_time>
   ...
   JobN
   <burst_time>
   ```

   Each dataset (for 5, 10, 15 jobs) will have its own job set file.

4. Random Dataset Generation: For each input size, generate 20 different job sets (trials). We'll do this using a simple random number generator in Python.

   ```python
   import random
   for t in range(1, 21):
       with open(f"job5_trial{t}.txt", "w") as f:
       for i in range(1, 6):
           f.write(f"Job{i}\n{random.randint(1, 25)}\n")
   ```

Performance Measurement

For each input size $n \in \{5, 10, 15\}$, we will run all four algorithms on the same 20 random input sets. Then record the average turnaround time for each trial:

$$X = \{x_1, x_2, \ldots, x_{20}\}$$

where each x_i is the average turnaround time for one trial. Lastly, compute the final performance metric:

$$\text{Mean Average Turnaround Time} = \frac{\sum_{i=1}^{20} x_i}{20}.$$

We can compare these average values across all scheduling algorithms to analyze efficiency and fairness.

Part 2: Implementation (30 points)

(a) Code each program based on the design (pseudocode or flow chart) in Part 1(a).

(b) Document the program appropriately.

(c) Test you program using the designed testing input data given in the table in Part 1(b), Make sure each program generates the correct answer by marking a "✓" if it is correct for each testing case for each program column in the table. Repeat the process of debugging if necessary.

(d) For each program, capture a screen shot of the execution (Compile & Run) using the testing case in Part 1(b) to show how this program works properly.

By now, four working programs are created and ready for experimental study in the next part, Part 3.

The code is well-documented and runs for each scheduling algorithm. The following image shows the output of the average turnaround times for each testing case.

Figure 1: Output of program execution with average turnaround times

The following are the schedule tables of all four scheduling algorithms for the test case with 5 jobs.

Table 2: First-Come-First-Serve Schedule Table

Job	**Start Time**	**End Time**	**Job Completion**
Job 1	0	7	Job 1 completed @ 7
Job 2	7	25	Job 2 completed @ 25
Job 3	25	35	Job 3 completed @ 35
Job 4	35	39	Job 4 completed @ 39
Job 5	39	51	Job 5 completed @ 51

Table 3: Shortest-Job-First Schedule Table

Job	**Start Time**	**End Time**	**Job Completion**
Job 4	0	4	Job 4 completed @ 4
Job 1	4	11	Job 1 completed @ 11
Job 3	11	21	Job 3 completed @ 21
Job 5	21	33	Job 5 completed @ 33
Job 2	33	51	Job 2 completed @ 51

Table 4: Round-Robin with Time Slice = 2 Schedule Table

Job	**Start Time**	**End Time**	**Job Completion**
Job 1	0	2	—
Job 2	2	4	—
Job 3	4	6	—
Job 4	6	8	—
Job 5	8	10	—
Job 1	10	12	—
Job 2	12	14	—
Job 3	14	16	—
Job 4	16	18	Job 4 completed @ 18
Job 5	18	20	—
Job 1	20	22	—
Job 2	22	24	—
Job 3	24	26	—
Job 5	26	28	—
Job 1	28	29	Job 1 completed @ 29
Job 2	29	31	—
Job 3	31	33	—
Job 5	33	35	—
Job 2	35	37	—
Job 3	37	39	Job 3 completed @ 39
Job 5	39	41	—
Job 2	41	43	—
Job 5	43	45	Job 5 completed @ 45
Job 2	45	47	—
Job 2	47	49	—
Job 2	49	51	Job 2 completed @ 51

Table 5: Round-Robin with Time Slice = 5 Schedule Table

Job	Start Time	End Time	Job Completion
Job 1	0	5	—
Job 2	5	10	—
Job 3	10	15	—
Job 4	15	19	Job 4 completed @ 19
Job 5	19	24	—
Job 1	24	26	Job 1 completed @ 26
Job 2	26	31	—
Job 3	31	36	Job 3 completed @ 36
Job 5	36	41	—
Job 2	41	46	—
Job 5	46	48	Job 5 completed @ 48
Job 2	48	51	Job 2 completed @ 51

Part 3: Performance Analysis (40 points)

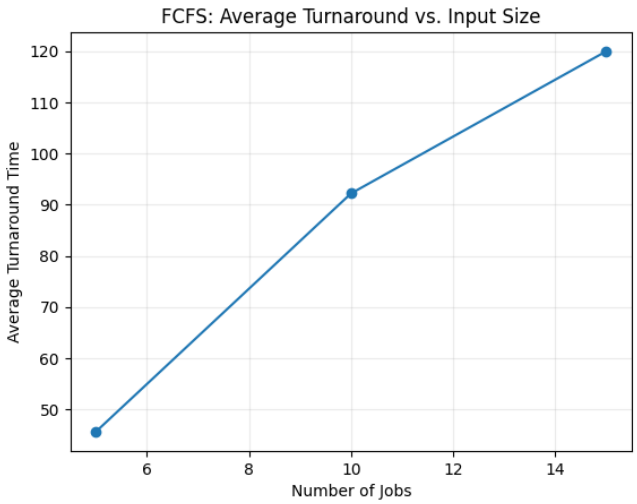
(a) Run each program with the designed randomly generated input data given in Part 1(c). Generate a table for all the experimental results for performance analysis as follows.

Table 6: Means of Average Turnaround Time

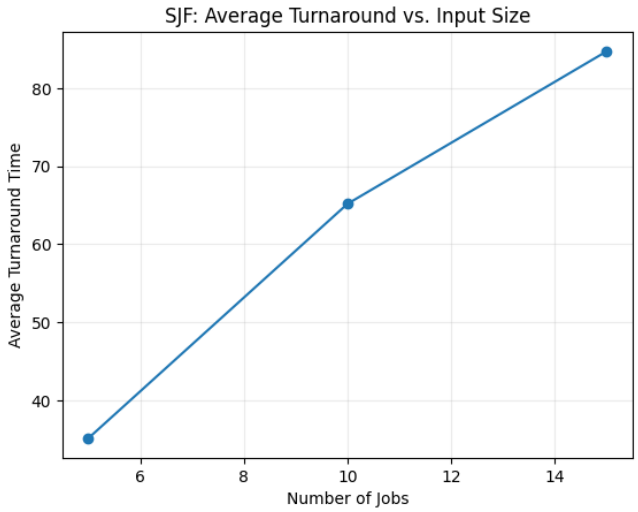
Input Size n jobs	Average of average turnaround times (FCFS Program)	Average of average turnaround times (SJF Program)	Average of average turnaround times (RR-2)	Average of average turnaround times (RR-5)
5 jobs	45.650	35.130	54.260	54.110
10 jobs	92.180	65.180	112.040	111.550
15 jobs	119.933	84.730	150.876	149.293

(b) Plot a graph of each algorithm, average turnaround time vs input size (# of jobs), and summarize the performance of each algorithm based on its own graph.

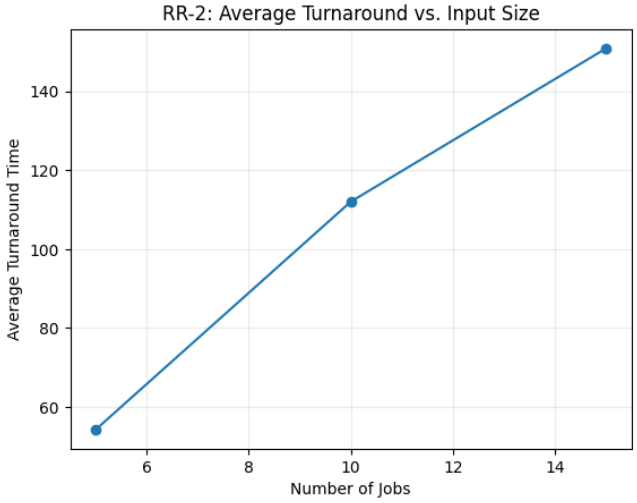
Plot all four graphs on the same graph and compare the performance of all four algorithms. Rank four scheduling algorithms. Try giving the reasons for the findings.



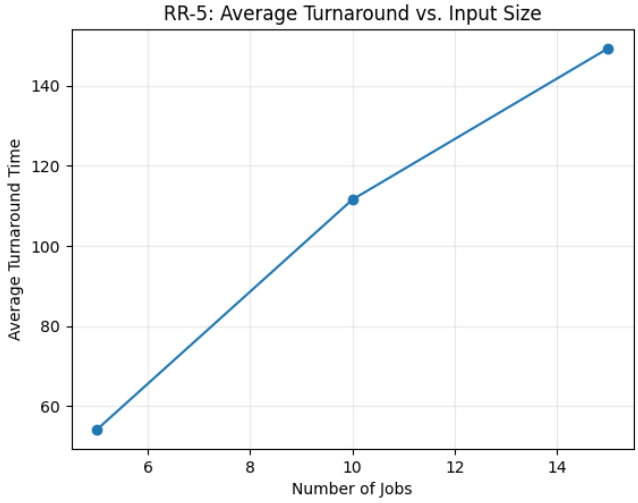
(a) Average Turnaround vs. Input Size (First-Come-First-Serve)



(b) Average Turnaround vs. Input Size (Shortest-Job-First)



(c) Average Turnaround vs. Input Size (Round-Robin (2 ms))



(d) Average Turnaround vs. Input Size (Round-Robin (5 ms))

Figure 2: Average Turnaround vs. Input Size for Four Scheduling Algorithms

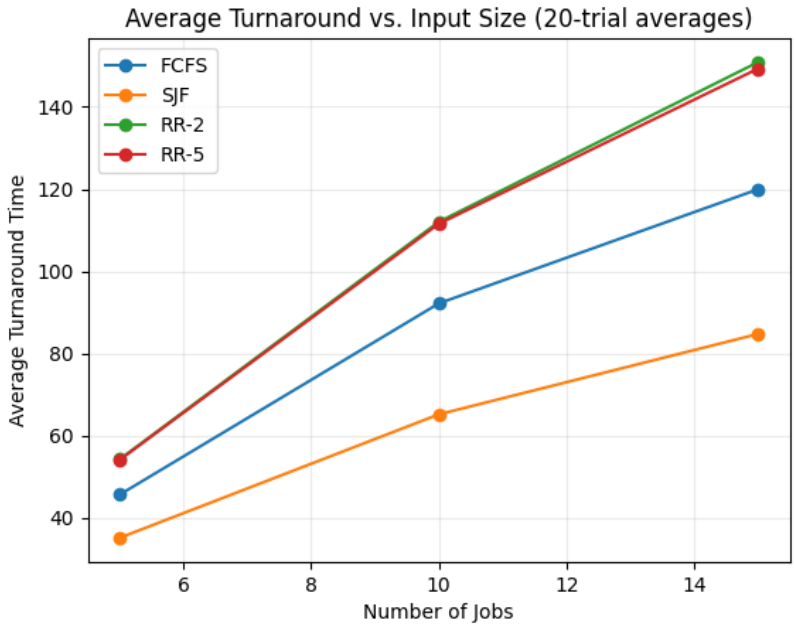


Figure 3: Average Turnaround vs. Input Size (Combined)

The overall ranking for the four scheduling algorithms based on the mean of the average turnaround times:

1. Shortest-Job-First: 61.680
2. First-Come-First-Serve: 85.921
3. Round-Robin with Time Slice = 5 (5 ms): 104.984
4. Round-Robin with Time Slice = 2 (2 ms): 105.726

Some observations I noticed in the average turnaround times were:

- Shortest-Job-First consistently has the lowest average turnaround since it always serves the shortest job first.
- First-Come-First-Serve and Round-Robin (5 ms) are often close. A larger quantum makes Round-Robin behave like First-Come-First-Serve and reduces rotation overhead.
- Round-Robin (2 ms) usually has the highest average turnaround because frequent time slicing increases waiting in the queue.
- As input size grows, all algorithms trend upward because total service time and queuing both increase.

(c) Conclude your report with the strength and constraints of your work. At least 100 words. (Note: It is reflection of this project. If you have a chance to re-do this project again, what you like to keep and what you like to do differently in order get a better quality of results.)

This project reinforced how scheduling policies affect CPU efficiency and turnaround time. Among the four algorithms, Shortest-Job-First performed best, followed by First-Come-First-Serve, Round-Robin (5 ms), and Round-Robin (2 ms). Both Round-Robin algorithms had the highest average turnaround because the frequent time slicing increased the queuing order. The advantage of the Shortest-Job-First algorithm comes from minimizing waiting time by handling shorter processes first, while smaller time slices in Round-Robin increased context switching and waiting. Something I noticed was that as the input size grows, all the algorithms trend upward because the total service time and queuing both increase. Overall, the work in this project demonstrated the trade-off between fairness and performance. If I redid this project, I would like to add more unique test cases to see the variability of waiting and response times in an analysis. This would potentially help better visualize CPU utilization in how each scheduling algorithm balances responsiveness and throughput.