CS 4310 Operating Systems
Project #1 Simulating Job Scheduler and Performance Analysis

Due: 6/23
(Total: 100 points)

Student Name: _Jasmine Dao_

Date: __06/23/2020__

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 (PR-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 use 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 @blackboard.cpp.edu 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)***

Part1
Design & Testing (30 points)

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

// First-Come-First-Serve (FCFS)

```
for (each line in text file) {
        store value in array
}

end time = 0

for (each value in array) {
        if (current iteration of loop is even) {
                print job number = current iteration of loop + 1
        }
        else if (current iteration of loop is odd) {
                get burst value from array
                end time += burst value

                print start time = 0
                print end time
                print which job was completed at end time
        }
}
```

// Shortest-Job-First (SJF)

```
for (each line in text file) {
        store values from even numbered lines in array (burst values only)
}

end time = 0

for (each value in array) {
        for (each burst value in array) {
                compare each value to the one next to it to find minimum
        }

        end time += minimum
        remove current minimum value in array

        print job number = current iteration of loop + 1
        print start time = 0
        print end time
        print which job was completed at end time
}
```

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

```
for (each line in text file) {
        store values from even numbered lines in array (burst values only)
}

end time = 0

while (completed number of jobs != total number of jobs) {
        for (each burst value in array) {
                subtract 2 from each value

                if (current value > 0) {
                        end time += 2
                        print job number = current iteration of while loop + 1
                        print start time = end time - 2
                        print end time
                }

                if (current value == 0) {
                        end time += 2
                        add 1 to number of completed jobs
                        print job number = current iteration of while loop + 1
                        print start time = end time - 2
                        print end time
                        print which job was completed at end time
                        remove job from circulation
                }

                if (current value < 0) {
                        end time += 1
                        add 1 to number of completed jobs
                        print job number = current iteration of while loop + 1
                        print start time = end time – 1
                        print end time
                        print which job was completed at end time
                        remove job from circulation
                }
        }
}
```

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

```
for (each line in text file) {
        store values from even numbered lines in array (burst values only)
}

end time = 0
```

```
while (completed number of jobs != total number of jobs) {
        for (each burst value in array) {
                subtract 5 from each value

                if (current value > 0) {
                        end time += 5
                        print job number = current iteration of while loop + 1
                        print start time = end time - 5
                        print end time
                }

                if (current value == 0) {
                        end time += 5
                        add 1 to number of completed jobs
                        print job number = current iteration of while loop + 1
                        print start time = end time - 5
                        print end time
                        print which job was completed at end time
                        remove job from circulation
                }

                if (current value < 0) {
                        find absolute value of current value
                        end time += absolute value
                        add 1 to number of completed jobs
                        print job number = current iteration of while loop + 1
                        print start time = 5 – absolute value
                        print end time
                        print which job was completed at end time
                        remove job from circulation
                }
        }
}
```

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.

Testing case #	Input (table of jobs with its job# and length	Expected output for FCFS (√ if Correct after testing in Part 3)	Expected output for SJF (√ if Correct after testing in Part 3)	Expected output for RR-2 (√ if Correct after testing in Part 3)	Expected output for RR-5 (√ if Correct after testing in Part 3)
1 (5 jobs)	Job 1: 7 Job 2: 18 Job 3: 10 Job 4: 4 Job 5: 12	31.4 √	24 √	36.4 √	36 √
2 (10 jobs)	Job 6: 5 Job 7: 15 Job 8: 8 Job 9: 3 Job 10: 9	53.6 √	37.2 √	64 √	60.8 √
3 (15 jobs)	Job 11: 13 Job 12: 16 Job 13: 2 Job 14: 6 Job 15: 11	76.6 √	53.8 √	94.8 √	92.67 √

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):

$$\textit{Average Turnaround Time} = \frac{\sum_{i=1}^{20} Xi}{20}$$

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

To test the algorithms, a program can be created to generate random lists of jobs that are written to a file, the number of which depend on user input. Then, different data sizes can be used to determine which algorithms are most efficient for small or large numbers of jobs and lesser or greater job lengths.

Part 2
Implementation (30 points)

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

(Algorithms are all in one file with selection menu to execute algorithm of choice.)

b. Document the program appropriately.

Done

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.

Done

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

```
C:\Users\Jasmine\Desktop\CS 4310\DaoJasmine_CS4310_Project>java JobSchedulingSimulator

Which of the following scheduling algorithms would you like to simulate?
1. First-Come-First-Serve (FCFS)
2. Shortest-Job-First (SJF)
3. Round-Robin with Time Slice = 2 (RR-2)
4. Round-Robin with Time Slice = 5 (RR-5)

1

+----------+------------+------------+------------+------------------------+
| Job #    | Start Time | End Time   | Burst Time | Job Completion         |
+----------+------------+------------+------------+------------------------+
| 1        | 0          | 7          | 7          | Job 1 completed @7     |
+----------+------------+------------+------------+------------------------+
| 2        | 0          | 25         | 18         | Job 2 completed @25    |
+----------+------------+------------+------------+------------------------+
| 3        | 0          | 35         | 10         | Job 3 completed @35    |
+----------+------------+------------+------------+------------------------+
| 4        | 0          | 39         | 4          | Job 4 completed @39    |
+----------+------------+------------+------------+------------------------+
| 5        | 0          | 51         | 12         | Job 5 completed @51    |
+----------+------------+------------+------------+------------------------+

Average Turnaround Time: 31.4
```

```
C:\Users\Jasmine\Desktop\CS 4310\DaoJasmine_CS4310_Project>java JobSchedulingSimulator

Which of the following scheduling algorithms would you like to simulate?
1. First-Come-First-Serve (FCFS)
2. Shortest-Job-First (SJF)
3. Round-Robin with Time Slice = 2 (RR-2)
4. Round-Robin with Time Slice = 5 (RR-5)

2

+----------+------------+------------+------------+------------------------+
| Job #    | Start Time | End Time   | Burst Time | Job Completion         |
+----------+------------+------------+------------+------------------------+
| 4        | 0          | 4          | 4          | Job 4 completed @4     |
+----------+------------+------------+------------+------------------------+
| 1        | 0          | 11         | 7          | Job 1 completed @11    |
+----------+------------+------------+------------+------------------------+
| 3        | 0          | 21         | 10         | Job 3 completed @21    |
+----------+------------+------------+------------+------------------------+
| 5        | 0          | 33         | 12         | Job 5 completed @33    |
+----------+------------+------------+------------+------------------------+
| 2        | 0          | 51         | 18         | Job 2 completed @51    |
+----------+------------+------------+------------+------------------------+

Average Turnaround Time: 24.0
```

```
C:\Users\Jasmine\Desktop\CS 4310\DaoJasmine_CS4310_Project>java JobSchedulingSimulator

Which of the following scheduling algorithms would you like to simulate?
1. First-Come-First-Serve (FCFS)
2. Shortest-Job-First (SJF)
3. Round-Robin with Time Slice = 2 (RR-2)
4. Round-Robin with Time Slice = 5 (RR-5)

3

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

Average Turnaround Time: 36.4
```

```
C:\Users\Jasmine\Desktop\CS 4310\DaoJasmine_CS4310_Project>java JobSchedulingSimulator

Which of the following scheduling algorithms would you like to simulate?
1. First-Come-First-Serve (FCFS)
2. Shortest-Job-First (SJF)
3. Round-Robin with Time Slice = 2 (RR-2)
4. Round-Robin with Time Slice = 5 (RR-5)

4

+----------+------------+------------+------------+------------------------+
| Job #    | Start Time | End Time   | Burst Time | Job Completion         |
+----------+------------+------------+------------+------------------------+
| 1        | 0          | 5          | 2          |                        |
+----------+------------+------------+------------+------------------------+
| 2        | 5          | 10         | 13         |                        |
+----------+------------+------------+------------+------------------------+
| 3        | 10         | 15         | 5          |                        |
+----------+------------+------------+------------+------------------------+
| 4        | 15         | 19         | 0          | Job 4 completed @19    |
+----------+------------+------------+------------+------------------------+
| 5        | 19         | 24         | 7          |                        |
+----------+------------+------------+------------+------------------------+
| 1        | 24         | 26         | 0          | Job 1 completed @26    |
+----------+------------+------------+------------+------------------------+
| 2        | 26         | 31         | 8          |                        |
+----------+------------+------------+------------+------------------------+
| 3        | 31         | 36         | 0          | Job 3 completed @36    |
+----------+------------+------------+------------+------------------------+
| 5        | 36         | 41         | 2          |                        |
+----------+------------+------------+------------+------------------------+
| 2        | 41         | 46         | 3          |                        |
+----------+------------+------------+------------+------------------------+
| 5        | 46         | 48         | 0          | Job 5 completed @48    |
+----------+------------+------------+------------+------------------------+
| 2        | 48         | 51         | 0          | Job 2 completed @51    |
+----------+------------+------------+------------+------------------------+

Average Turnaround Time: 36.0
```

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

Part 3
Performance Analysis (40 points)

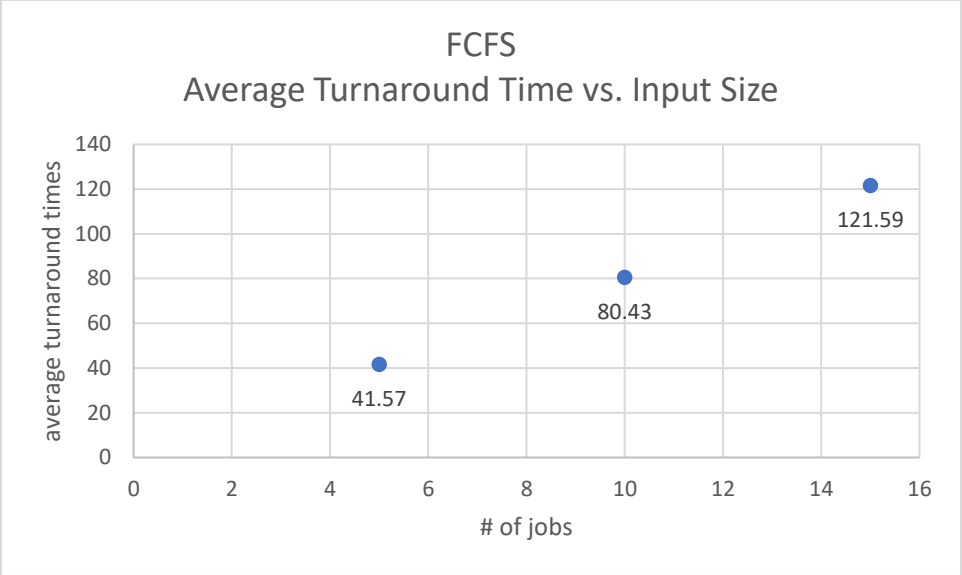
a. Run each program with the designed randomly generated input data given in Part 1(c). Generate a table for all the experimental results for performance analysis as follows.

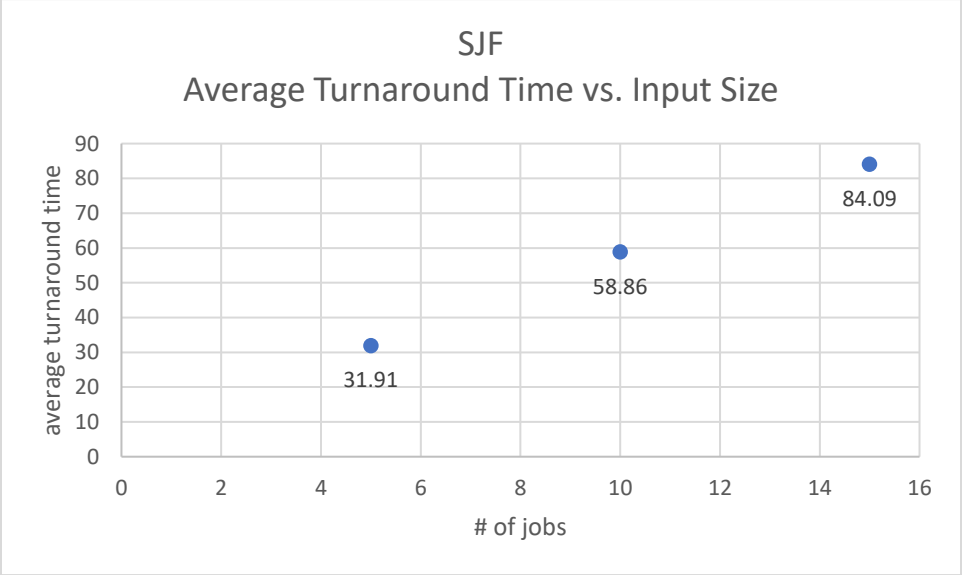
(Maximum job length is 30.)

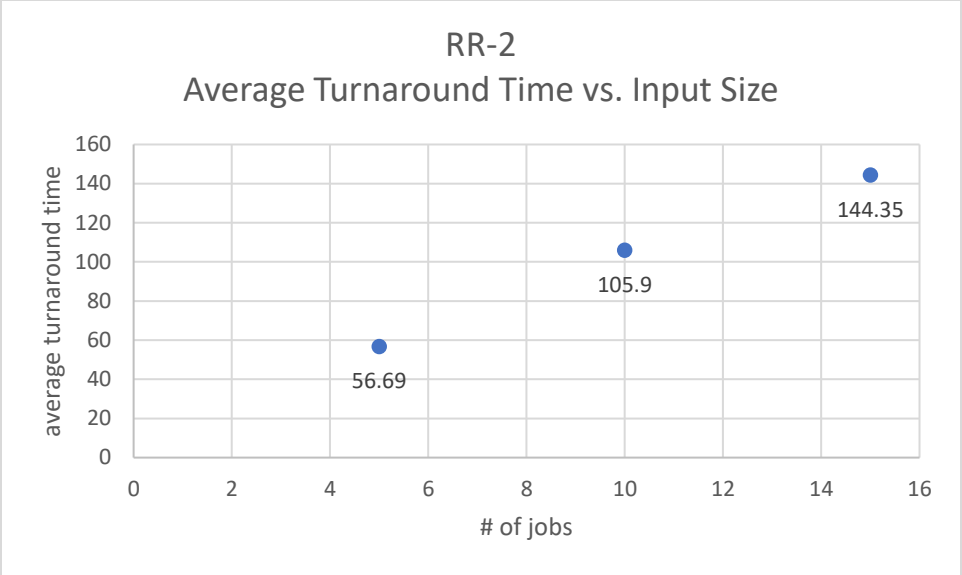
Input Size n jobs	Average of average turnaround times (FCFS Program)	Average of average turnaround times (SFJ Program)	Average of average turnaround times (RR-2)	Average of average turnaround times (RR-5)
5 jobs	42.8 71 53.6 59.6 58 14.4 45.2 25.4 25 38.6 31.6 48.6 43.2 40.8 48.2 34.4 31.4 34.2 57 28.4 **Avg = 41.57**	33.4 24.6 10.6 35.2 48.6 26.2 40.2 45 45.6 44 44.6 31 31 37.8 24 15.6 35.2 11.8 30 23.8 **Avg = 31.91**	12.2 41.8 49.6 64.2 45.6 39.8 50.4 52.6 68.6 54.8 49.4 94.4 127 74 58.6 90.6 38.2 41.8 42 38.2 **Avg = 56.69**	52 73.6 40.6 39.8 50.8 48.8 62.6 35.8 47 28.4 30.4 48.2 92.6 47.4 104.6 38.8 41.4 39.8 32 62.4 **Avg = 50.85**
10 jobs	100.1 68.4 97.6 101.9 110.2 112.1 77.6 77.5 62.7 39.5 50.9 71.6 97.6 80.3 86.4 101.9 64.6	54.1 70.8 37.8 46.3 43 59.9 47.5 64.4 56.9 73.6 72 64.6 64.1 66 47.7 77.6 85.9	104.7 107.6 73.1 109.6 86.2 97.5 83.6 97.5 101.3 143.5 124.9 144.3 71.9 84.1 87 142.8 109.8	109.3 110.2 98.6 67 101.8 102.8 154 165.9 142 67 93.2 147.6 57.3 78.3 107.4 91.4 61.9

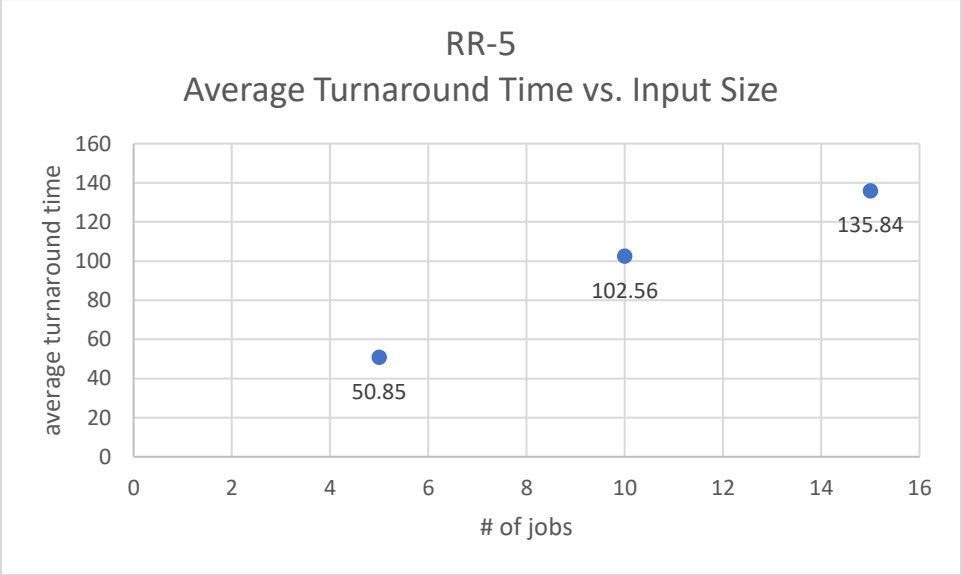
	68.5 86.4 52.8 **Avg = 80.43**	33 64.4 47.6 **Avg = 58.86**	96.8 122 129.8 **Avg = 105.9**	138.5 68.4 88.5 **Avg = 102.56**
15 jobs	150.9 103.6 125 126.1 93.3 103.5 120.5 112.9 129.9 152.2 158.5 134.5 105.3 121.7 111.9 117 136.7 97.5 143.2 87.6 **Avg = 121.59**	123.4 95 99.1 75.3 111.1 41.3 86.7 83.9 113.5 75.3 109.1 103.9 60.5 61.3 78.4 43.9 47.7 92.6 89.1 90.7 **Avg = 84.09**	136.4 146.8 167.1 136.3 95 122.7 132.1 106.4 216.5 161.9 183.2 149 118.5 156.9 89.5 192.3 157.1 129.5 169.9 119.9 **Avg = 144.35**	77.9 114.6 156.9 183.4 74.1 119.7 155.9 136.4 169 156.2 147.7 117.9 117.1 115.2 203.2 131.9 96.7 219.4 100.4 123.1 **Avg = 135.84**

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.



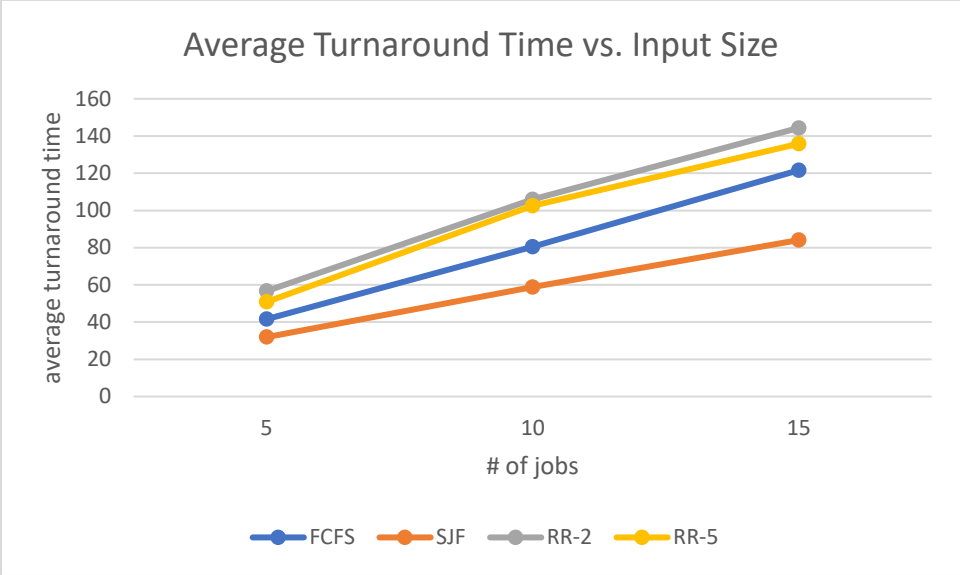






For all the programs, the average turnaround times and the number of jobs for 20 trials seem to have a rather linear progression. Allowing for outliers, these numbers would likely follow the linear pattern more closely if more trials were added. However, it is also possible that if more input sizes were considered, the RR-2 and RR-5 algorithms would begin to show a pattern where the average turnaround times would no longer increase so drastically with the number of jobs (radical function).

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.



Among these algorithms, the Shortest-Job-First (SJF) program consistently has the lowest average turnaround times for each set of jobs, and the Round-Robin with the time slice 2 (RR-2) algorithm has the highest. When comparing the Round-Robin algorithms, the difference does not seem overly drastic, but it can be presumed that the greater the value of the time slice, the more efficient the algorithm will be. While both the SJF and FCFS algorithms show a distinctly linear progression, it is possible that the RR-2 and RR-5 algorithms may eventually taper off and increase only minutely given a greater number of jobs. In this case, then there will be a point when the Round Robin algorithms are more efficient than the other two. However, this is only speculation, as it is possible that the slightly skewed points in the pattern may be caused by outliers.

1. **SJF**
2. **FCFS**
3. **RR-5**
4. **RR-2**

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 change 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.)

For this project, the planning stage was very helpful in designing the algorithms and preparing for the rest of the steps. Having written the pseudo-code and concocted a testing strategy, it was easier to program the algorithms and test the accuracy of the turnaround times. However, the testing portion of the project was very tedious and inefficient. For each input size and algorithm, I had to run both the job generator program and the job scheduling simulator twenty times each and record each average I received, then calculate the average of those. If I could do this differently, I would probably temporarily modify the program so that it ran twenty times and calculated the average itself. Otherwise, I thought that this project was rather interesting and fun to work on. I learned how to format an ASCII table and print it in the command line, as well as how to better design and plan a program.