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

Due: 6/23
(Total: 100 points)

Student Name: Christopher Saba

Date: 6/19/20

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.

<Insert answers here>

```
Create Job.txt with random entries
Hashtable hashTable = new Hashtable(30)
Hashtable.put(job.txt entries with job#, time)

First Come first Serve algorithm:
For (I = 1; I <= size; i++){
Time = get hashtable(job)
totalTime += time
}
Return totalTime / size

Shortest Job First algorithm:
Sort hashtable by time() //or use another method to sort time
For (I = 1; I <= size; i++){
Time = get hashtable(job) // we can import time into an array and sort array
totalTime += time
}
Return totalTime / size

Round robin 2ms timeslice:

While(jobsDone < size){
        jobTimeLeft = jobtime – roundRobinTime

        if(jobTimeLeft > 0)
        print time left on job
        timeAdd+= roundRobinTime;

        else if(jobTimeLeft == -1)
        jobsDone++
        timeAdd ++
        totalTime += timeAdd

        else
        timeAdd += roundRobinTime
        JobsDone++
        totalTime += timeAdd

increment next job
}



Round Robin 5ms:
While(jobsDone < size){
        jobTimeLeft = jobtime – roundRobinTime

        if(jobTimeLeft > 0)
        print time left on job
        timeAdd+= roundRobinTime;
```

```
            else if(jobTimeLeft == -1)
            jobsDone++
            timeAdd += 4
            totalTime += timeAdd

            else if(jobTimeLeft == -2)
            jobsDone++
            timeAdd += 3
            totalTime += timeAdd

            else if(jobTimeLeft == -3)
            jobsDone++
            timeAdd += 2
            totalTime += timeAdd

            else if(jobTimeLeft == -4)
            jobsDone++
            timeAdd += 1
            totalTime += timeAdd

            else
            timeAdd += roundRobinTime
            JobsDone++
            totalTime += timeAdd

    increment next job
    }
```

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.

<Complete the following table>

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)	<insert answers here> Job1 20 Job2 2 Job3 37 Job4 38 Job5 4	<insert answers here> 59.8√	<insert answers here> 39.6√	<insert answers here> 56.8√	<insert answers here> 56.6√

2 (10 jobs)	<insert answers here> Job1 20 Job2 2 Job3 37 Job4 38 Job5 4 Job6 29 Job7 15 Job8 14 Job9 31 Job10 37	<insert answers here> 115√	<insert answers here> 88.5√	<insert answers here> 150.4√	<insert answers here> 146.7√
3 (15 jobs)	<insert answers here> Job1 20 Job2 2 Job3 37 Job4 38 Job5 4 Job6 29 Job7 15 Job8 14 Job9 31 Job10 37 Job11 6 Job12 10 Job13	<insert answers here> 163.3√	<insert answers here> 106.53√	<insert answers here> 188.8√	<insert answers here> 186√

	15 Job14 13 Job15 24				

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).

<Insert answers here>

Using a Random Number Generator, we must first create a file that can be used and read by a data structure to store our new generated job lists. In a loop bound by the amount of jobs selected, we can write to the first line, Job + (i) so that it would read Job1 since "i" is starting at 1, and we end the line and start a new one, but this time we utilize RNG to give us a time for the job above, in my case I will bound this from 1 – 50. Once we completely fill out our new random job file, we can use a file reader to implement this file into our data structure, such as an array, hash table, or other form of storage. Suppose our storage method is complete, now all we must do is utilize our algorithms to calculate the job times and average turnaround times etc. Print out the times and we have a completely new data set for testing average algorithm turnaround times.

Part 2
Implementation (30 points)

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

<Generate four programs and stored them in four files, needed to be submitted>

b. Document the program appropriately.

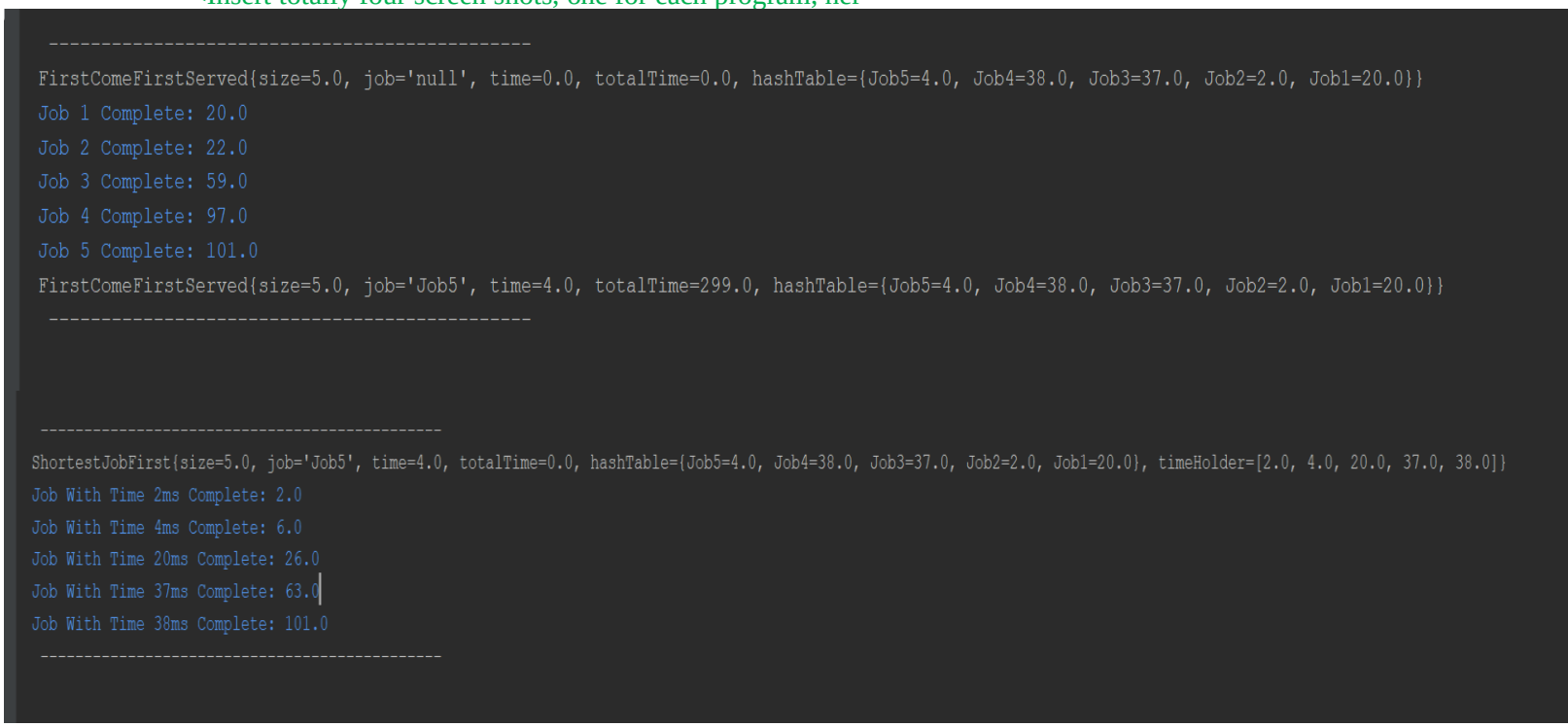
<Generate documentation inside the four program files>

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.

<Complete the four columns of the four algorithms in the table @Part 1(b)>

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

<Insert totally four screen shots, one for each program, her



```
 ------------------------------------------------
RoundRobin2{roundRobinTime=2.0, size=5.0, job='Job5', time=4.0, totalTime=0.0, jobsDone=0.0, jobTimeLeft=0.0, hashTable={Job5=4.
||Time left on current job 20ms ||

Job with time left 2ms Complete: 4.0
||Time left on current job 37ms ||
||Time left on current job 38ms ||
||Time left on current job 4ms ||
||Time left on current job 18ms ||
||Time left on current job 35ms ||
||Time left on current job 36ms ||

Job with time left 2ms Complete: 18.0
||Time left on current job 16ms ||
||Time left on current job 33ms ||
||Time left on current job 34ms ||
||Time left on current job 14ms ||
||Time left on current job 31ms ||
||Time left on current job 32ms ||
||Time left on current job 12ms ||
||Time left on current job 29ms ||
||Time left on current job 30ms ||
||Time left on current job 10ms ||
||Time left on current job 27ms ||
||Time left on current job 28ms ||
||Time left on current job 8ms ||
||Time left on current job 25ms ||
||Time left on current job 26ms ||
||Time left on current job 6ms ||
||Time left on current job 23ms ||
||Time left on current job 24ms ||
||Time left on current job 4ms ||
||Time left on current job 21ms ||
||Time left on current job 22ms ||

Job with time left 2ms Complete: 62.0
||Time left on current job 19ms ||
||Time left on current job 20ms ||
||Time left on current job 17ms ||
||Time left on current job 18ms ||
||Time left on current job 15ms ||
||Time left on current job 16ms ||
||Time left on current job 13ms ||
||Time left on current job 14ms ||
||Time left on current job 11ms ||
||Time left on current job 12ms ||
||Time left on current job 9ms ||
||Time left on current job 10ms ||
||Time left on current job 7ms ||
||Time left on current job 8ms ||
||Time left on current job 5ms ||
||Time left on current job 6ms ||
||Time left on current job 3ms ||
||Time left on current job 4ms ||

Job with time left 1ms Complete: 99.0

Job with time left 2ms Complete: 101.0
 ------------------------------------------------
```

```
 ---------------------------------------------
RoundRobin5{roundRobinTime=5.0, size=5.0, job='Job5', time=4.0, totalTime=0.0, jobsDone=0.0, jobTimeLeft=0.0, hash
||Time left on current job 20ms ||

Job with time left 2ms Complete: 7.0
||Time left on current job 37ms ||
||Time left on current job 38ms ||

Job with time left 4ms Complete: 21.0
||Time left on current job 15ms ||
||Time left on current job 32ms ||
||Time left on current job 33ms ||
||Time left on current job 10ms ||
||Time left on current job 27ms ||
||Time left on current job 28ms ||

Job with time left 5ms Complete: 56.0
||Time left on current job 22ms ||
||Time left on current job 23ms ||
||Time left on current job 17ms ||
||Time left on current job 18ms ||
||Time left on current job 12ms ||
||Time left on current job 13ms ||
||Time left on current job 7ms ||
||Time left on current job 8ms ||

Job with time left 2ms Complete: 98.0

Job with time left 3ms Complete: 101.0
 ---------------------------------------------
Here are the averages for the four different algorithms using this many tests: 1 for this many jobs: 0

First Come First Served Average turnaround time: 59.8ms

Shortest Job First Average turnaround time: 39.6ms

Round Robin with a time splice of 2ms Average turnaround time: 56.8ms

Round Robin with a time splice of 5ms Average turnaround time: 56.6ms

Done!

Process finished with exit code 0
```

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

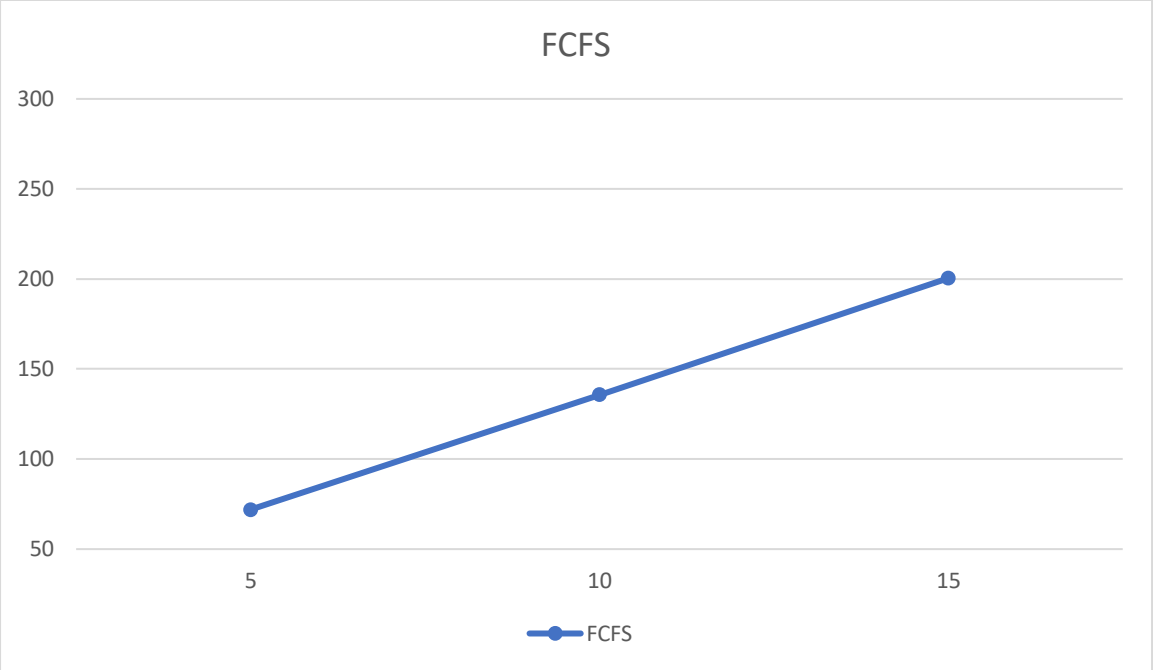
Part 3
Performance Analysis (40 points)

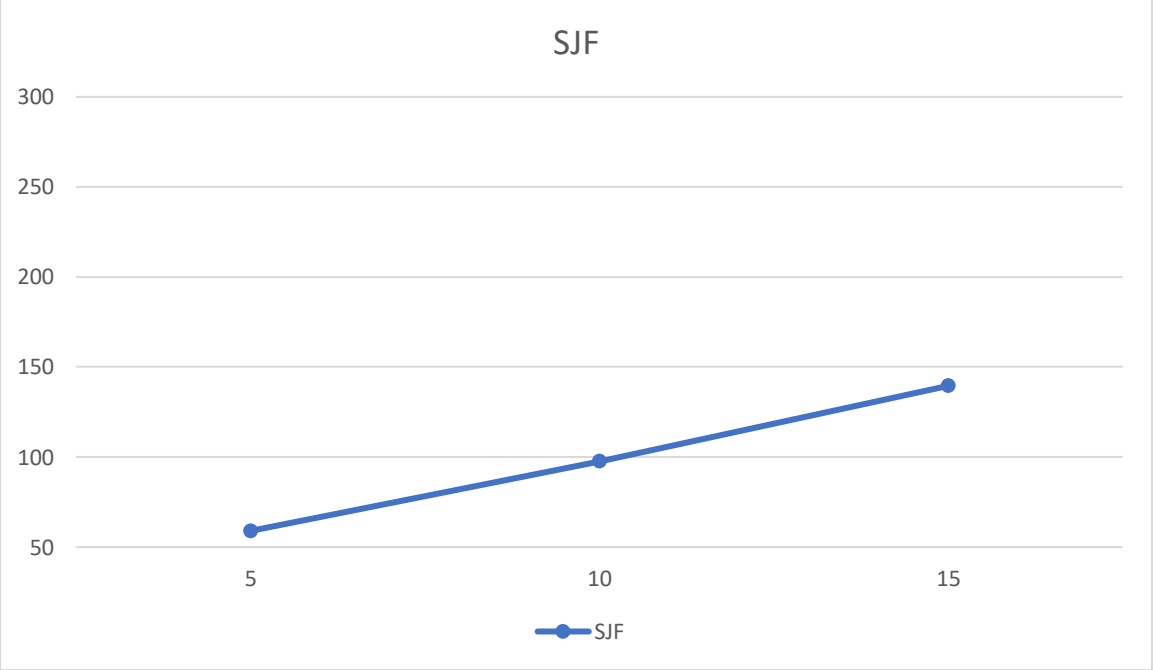
a. Run each program with the designed randomly generated input data given in Part 1(c). Generate a table for all the experimental results for performance analysis as follows.

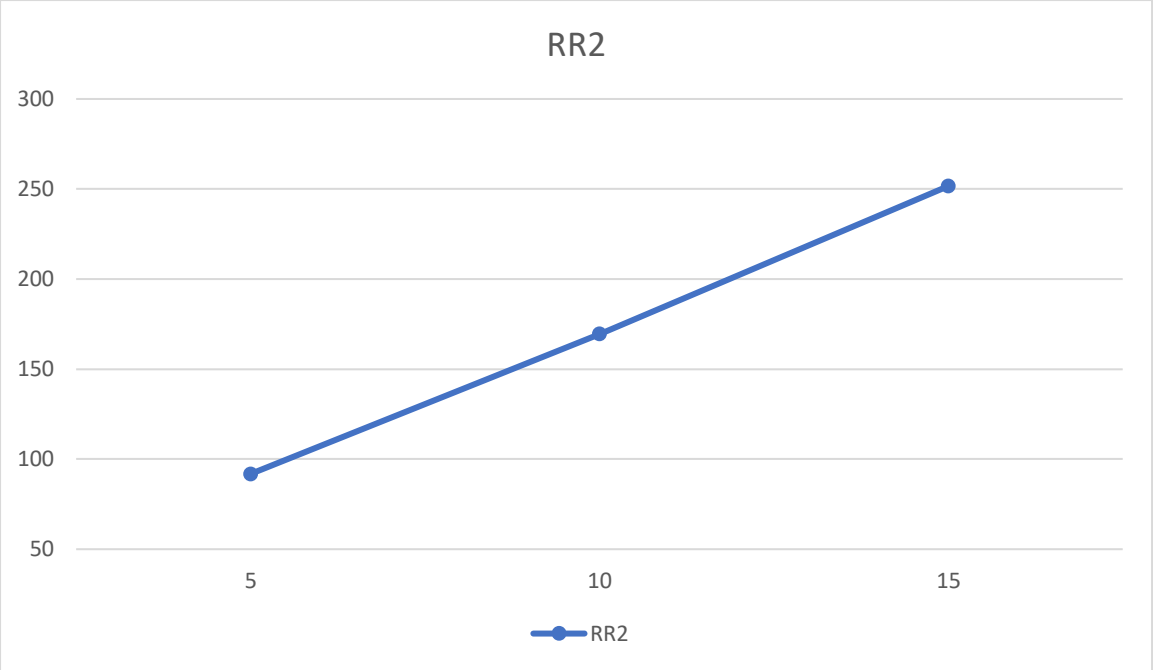
Input Size n jobs	Average of average turnaround times (FCFS Program)	Average of average turnaround times (SFJ Program)	Average of average turnaround times (RR-2)	Average of average turnaround times (RR-5)
5 jobs	*its average with 20 trials* *71.95ms*	*its average with 20 trials* *59.01*	*its average with 20 trials* *91.8*	*its average with 20 trials* 91.27
10 jobs	*its average with 20 trials* *135.685*	*its average with 20 trials* 97.585	*its average with 20 trials* 169.485	*its average with 20 trials* 170.535
15 jobs	*its average with 20 trials* 200.41	*its average with 20 trials* 139.537	*its average with 20 trials* 251.613	*its average with 20 trials* 252.383

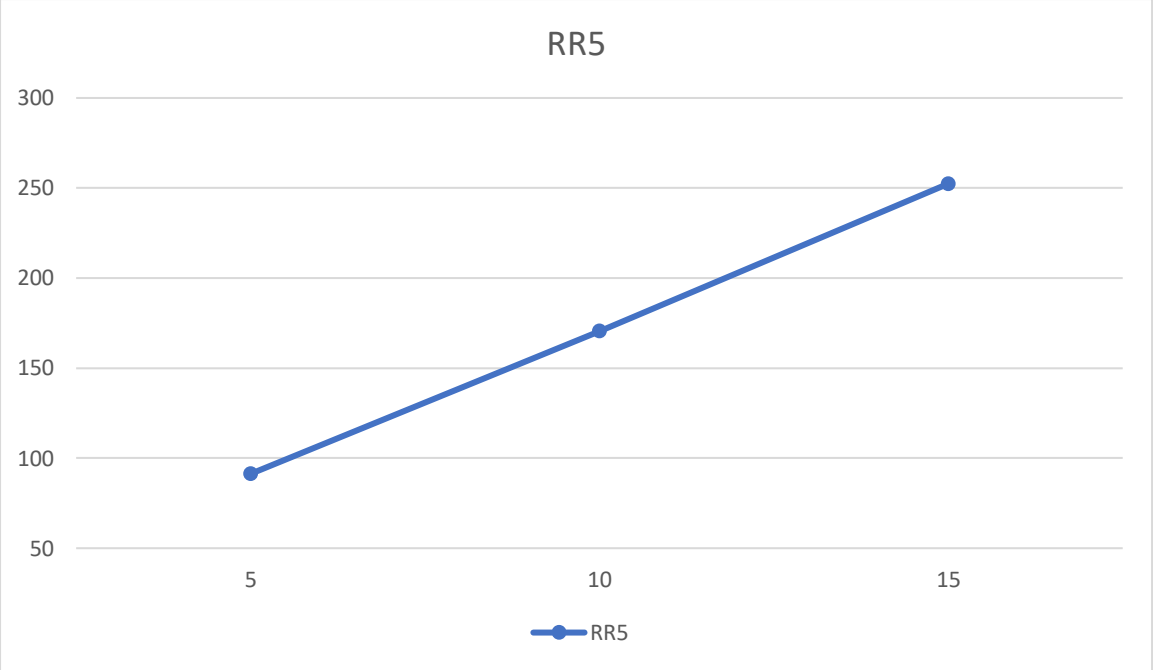
b. Plot a graph of each algorithm, average turnaround time vs input size (# of jobs), and summarize the performance of each algorithm based on its own graph.

<Insert totally four graphs, one for each program, here>





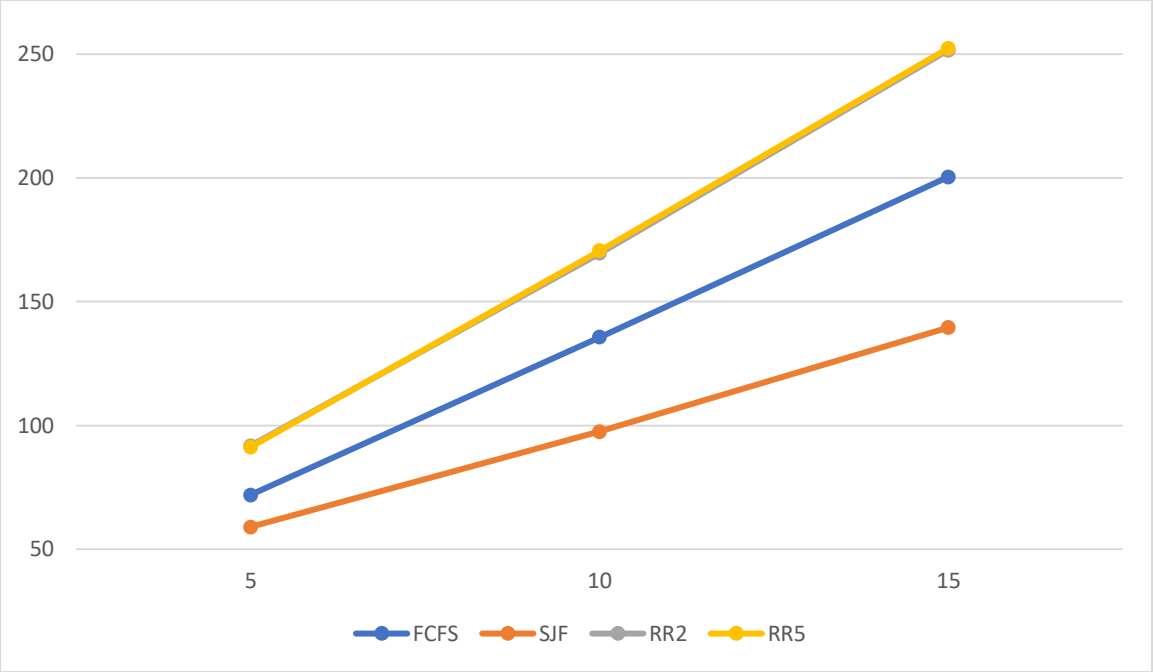




<Write a summary>
When looking at the graphs, FCFS, RR2, and RR5, there does not seem to be an easily sighted difference, while SJF is shown to be lower in slope than the others.

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.

<Insert four-graphs-in-one graph here>



<Write about explaining the results>

Looking at this all-in-one graph, you can see that SJF is clearly ahead of the others, with FCFS right behind. It is harder to see but Round Robin 5 is slightly ahead of RR2 in the first 5 jobs but lacks behind in 10 and 15 but the difference is very slight.

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.)

<Write a conclusion about strength and constraints of your work here.>

There are a few strengths I'd like to point out in my work, the ability to see the algorithms in action, color coded, and the ability to choose your own job list is something Id like to show my appreciation towards. You can clearly see each algorithm working with how much time left in each job. With as many strengths that I provide, there are a few constraints in my work that I would have liked to improve for my next time. In these algorithms I use Hash tables, a very low time complexity data structure, which in theory would prove well for these algorithms, and in terms of performance, it is easy to show that they work very well. However, I also use something called timeHolder; timeHolder is an array that stores times and the algorithm uses those arrays to do the work. This makes it hard to know which job is being worked on, since our only hint is the time left on that certain job. If I were able to implement a sorting algorithm for the hash table, I would, but it is too tedious and time consuming to change something that already works well enough. Also, in round robin 5, I could have grouped up all the cases with a single case loop and cater it towards a specific time slot, and that way I would only have to code 1 program for all round robin cases.