Assignment 3

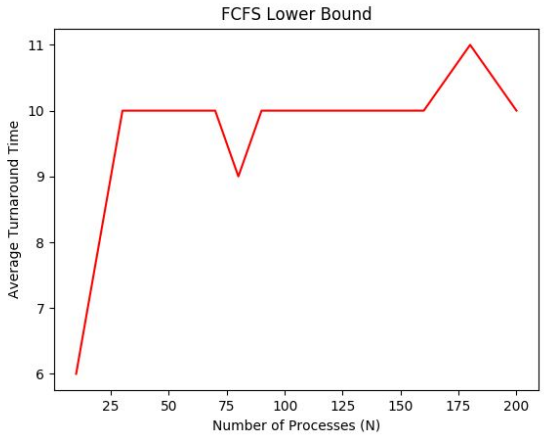
Theoretical analysis of FCFS and other plots

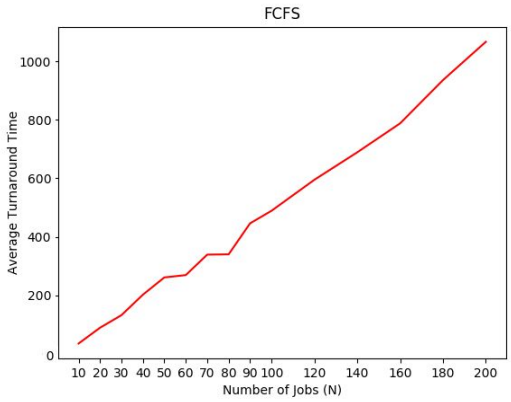
Q. for FCFS (non-preemptive) determine the theoretically expected lower bound on the turn-around time and check whether that is satisfied by your simulation.

A. To find the lower bound we have to consider the extremely favourable scenario in which each job gets the cpu as soon as it arrives and hence the sum of waiting time for all the processes will be zero. In this case the average turnaround time would be the sum of all cpu burst divided by number of processes, which will be the theoretically expected lower bound.

So, theoretically expected lower bound on the turn-around time = (sum of cpu bursts of all processes)/(no of processes)

In the case of our simulation, to compare the theoretical lower bound on turnaround time with the turnaround time seen in our simulations, we have plotted a graph showing expected and actual turnaround times for various processes from 10 to 200.

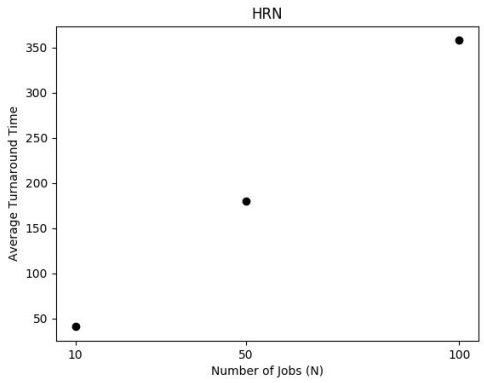


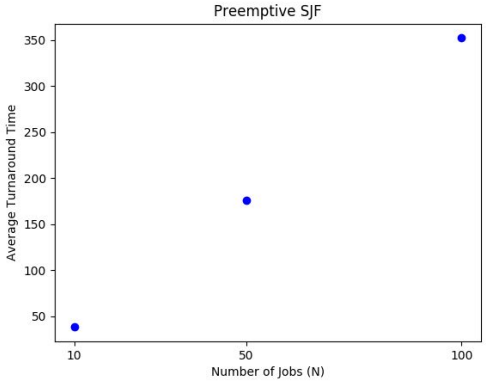


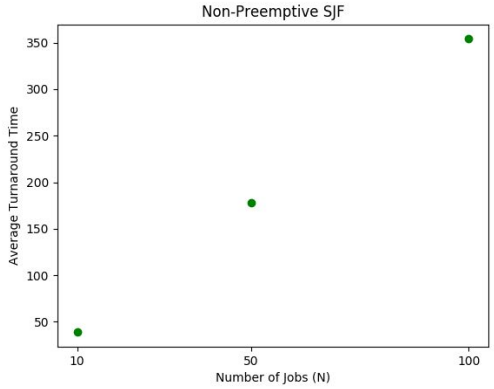
As is clear, the obtained turnaround time is always more than the theoretical expected lower bound. Note that the theoretical expected lower bound time hovers around 10.5 while actual turnaround times are much higher. The theoretical lower bound will converge to 10.5 because this is the value to which the average of the random variable will come to. These results are as expected.

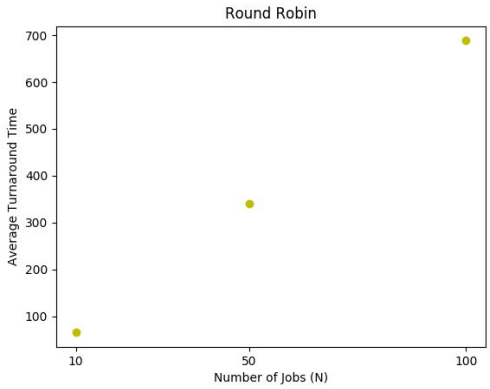
Plots

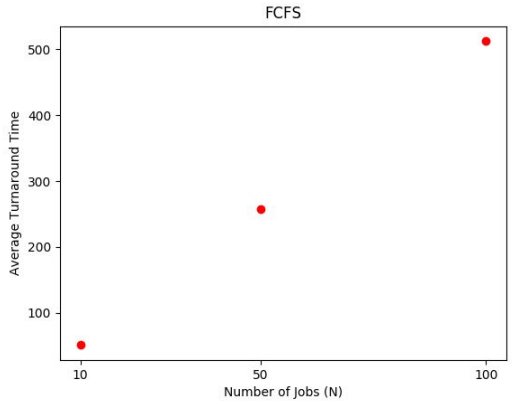
We have plotted the result of our simulations in order to easily interpret the obtained results. The first group of plots displayed here are scatter plots showing the **average turn-around time vs number of process** (10, 50 and 100) for various CPU scheduling algorithms.





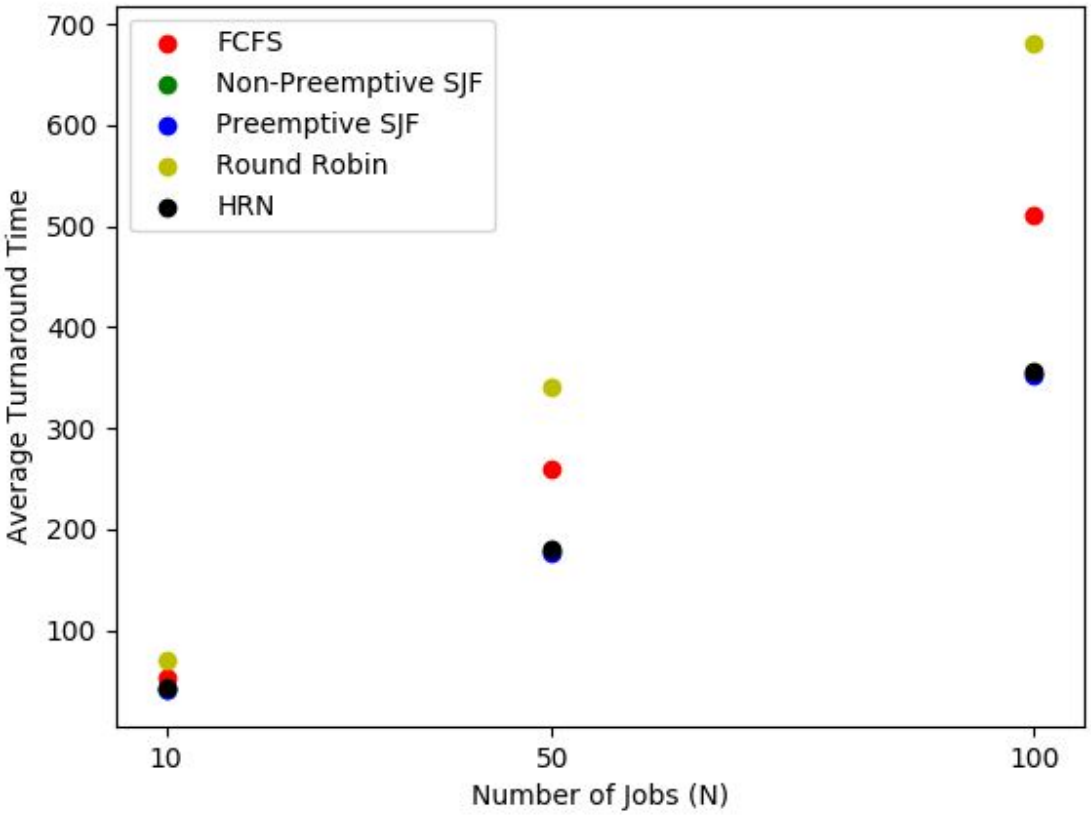




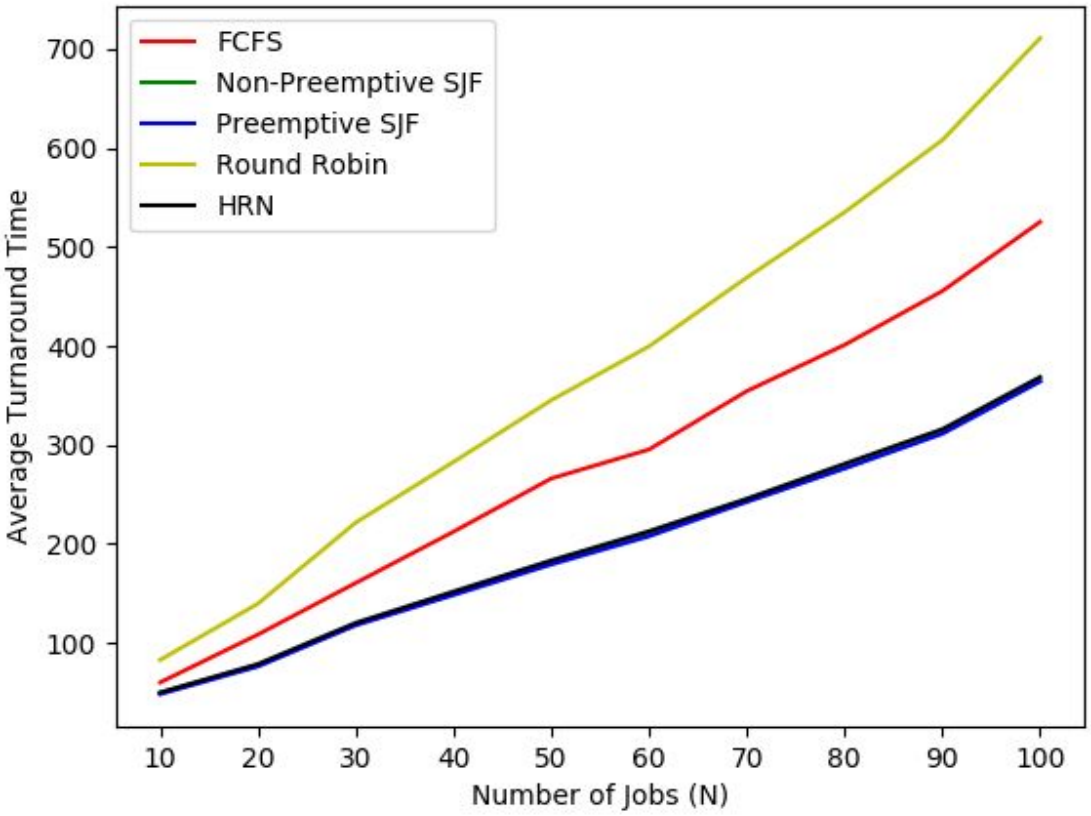


Note- HRN algorithms can be both preemptive and non preemptive. Since no criteria was given to us, we have chosen to implement the non preemptive HRN algorithm.

Do note that the range on the Y-axis is different for all the plots. To compare them properly, we have a scatter-plot depicting the **average turn-around time vs number of process** (10, 50 and 100) for all the CPU scheduling algorithms presented together.



The difference produced by the various job scheduling algorithms is better seen by plotting them for many number of different processes. We have done that and have plotted the obtained results as line graphs to better implement it.



It is clear the round robin is the worst CPU scheduling algorithm amongst the scheduling algorithms part of the test. First come first serve though better than round robin, is not very efficient either.

Preemptive shortest job first and HRN algorithms provide the best performance by giving the best (ie lowest) average turnaround time. We speculate that if preemptive scheduling was implemented then it might have outperformed all of these algorithms.

CODE FOR SCRIPT

```python
import numpy as np
import matplotlib.pyplot as plt
import subprocess
from subprocess import Popen, PIPE

print("Python script to run a3_14.cpp file")  #The main cpp file used by the script is a3_14.cpp

cmd = "a3_14.cpp"
subprocess.call(["g++","-std=c++11",cmd])

color = ["r","g","b","y","k"]                #deciding the various colors, each for various scheduling algorithm
# args = ["10", "20", "30", "40", "50", "60", "70", "80", "90", "100"]#, "120", "140", "160", "180", "200"]
labels = ["FCFS", "Non-Preemptive SJF", "Preemptive SJF", "Round Robin", "HRN"]
args = ["10", "50", "100"]                   #args holds the number of processes for which simulations are done. The above line can be
tn = []
atn = [[0.0 for i in range(len(args))] for j in range(5)]

for i in range(len(args)):                   #i stands for the no of jobs under considerations (is actually index for args array)
    for j in range(10):                  #for a given no of jobs, 10 simulations are done. j is the number of the simulation under work
        result = subprocess.Popen(["./a.out", args[i]], stdout=subprocess.PIPE)
        out = result.stdout.read()    #out hold the result as printed on screen
        tn = out.split('\n')[:-1]     #tn is an array that hold the result in proper formatting
        tn = np.array(map(int, tn))   #tn holds the result for the particular simulation by typecasting to int
        for k in range(5):            #k stands for the various scheduling algorithms
            atn[k][i] += tn[k]    #atn hold the sum of turnaround time for all simulation
    for k in range(5):
        atn[k][i] /= 10.0             #atn now hold the average turnaround time because we have divided the total turnaround time by num

plt.figure()
args = np.array(map(int, args))              #converting various elements of args from string to int
for k in range(5):
    # plt.figure()
    plt.scatter(args, atn[k], c=color[k]) #providing the information to be plotted
    plt.xlabel("Number of Jobs (N)")      #the label for x axis, which hold no of processes
    plt.ylabel("Average Turnaround Time") #the label for y axis, which hold average turnaround time
    plt.xticks(args)                      #label only the relevant number of jobs.
    # plt.title(labels[k])
    plt.legend(labels)                    #Displays the legend in the graph
    plt.savefig("Fig. "+str(k)+".png")

# plt.show()
```


- Nikhil Nayan Jha (16CS30022)
- Sarthak Chakraborty (16CS30044)

Group 14