Rashed Abdulla Alyammahi - 780696944
1.
a. I/O bound is more likely to have voluntary context switches because when I/O is bound, the CPU will have to wait for I/O to complete the program, so the CPU will be free and can do other tasks while waiting. I/O like the voluntary context switch that occurs when a resource request is currently unavailable will voluntarily give up control of the CPU.
b. CPU bound is more likely to have non-voluntary context switches because when CPU bound, I/O has to wait for CPU to do the work and it will be interrupted if a process with higher priority enters. like non-voluntary context switch occurs when the CPU is taken out of the process by actions such as using up CPU time or being taken over by a higher priority process.

2.
a. FIFO
With the FIFO algorithm, the process comes first will be executed first. It means, a process that will hold the CPU until it is done will return the CPU. The FIFO algorithm takes up a lot of waiting time, as the time-ordered processes have a large processing difference. In addition, it requires full CPU usage, which is easy to cause system crash if loading too many processes that have large burst times while running. So, it can lead to starvation result.

b. STCF
When the CPU is allocated, the STCF algorithm will prioritize the process with the smallest burst time for processing. The algorithm will give the minimum average waiting time for processes to be processed. However, if there are some very long processes, the STCF will swap the short processes and let the long processes waiting. So, The STCF can lead to starvation.

c. RR
The RR algorithm will allocate to the process a certain unit of CPU time also known as a quantum. When a process uses up its quantum time and still hasn't finished processing it, the rest of it is moved to the back of the queue list. Based on the Ready list, the CPU will take the next process to process. So it can eliminate CPU preemptive. If quantum(q) is too large then it will be similar to FIFO algorithm if q is too small it is mainly to perform context switching. So, no process left behind, the starvation doesn't occur.

d. Lottery
Lottery scheduling can solve the starvation problem by providing each process with at least one ticket guaranteeing that it has a non-zero probability of being selected at each scheduling operation.

3.
CPU performance of RR is better than STCF if we set a sensible quantum value.
The CPU time, memory(overstack), power(not enough power) can affect to os_overhead. For STCF a process with a long burst time may never be executed because the CPU will process the short processes first. RR, each process is executed because it provides the same CPU usage time. With STCF the running time has to be recorded resulting in higher cost to the processor, and RR if the quantum is very small then the very continuous switching of processes will result in overhead. In summary, the process running time is the same, but with RR, the os overhead is better than STCF.

4.
FIFO

```
|        A        |        B        |  C  |    D   |
0                20               40    45       55
```
Waiting time JA = 0
JB = 18
JC = 35
JD = 35
Average waiting time= (0+18+35+35)/4 = 22
Average response time= (20+38+40+45)/4 = 35.75

SJF

```
|        A       |  C  |    D   |        B       |
0               20    25       35               55
```
Waiting time JA = 0
JB = 33
JC = 15
JD = 15
Average waiting time = (0+33+15+15)/4 = 15.75
Average response time = (20+53+20+25)/4 = 29.5

STCF

```
|A| B |  C  |    D   |       B       |        A       |
0 2   5    10       20              37               55
```

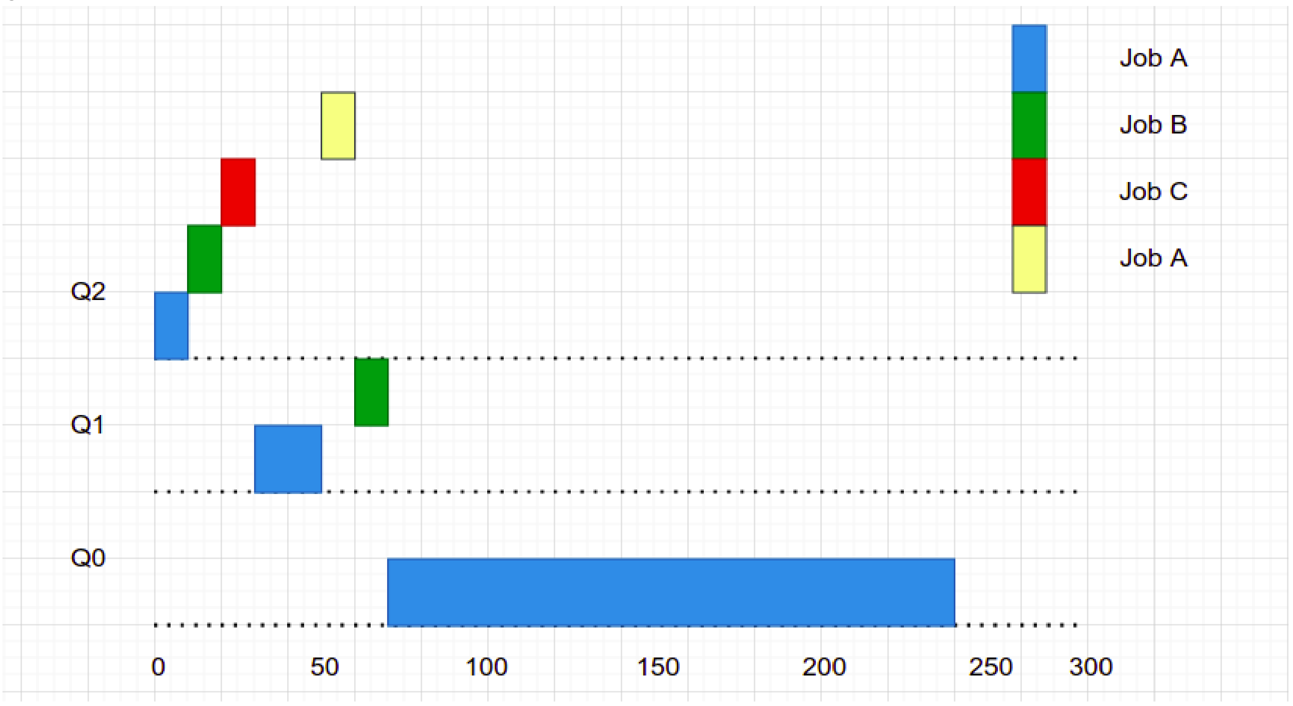
Waiting time JA = 35
JB = 15
JC = 0
JD = 0
Average waiting time= (35+15+0+0)/4= 12.5
Average response time= (45+35+5+10)/4= 23.75

RR (quantum = 5)

```
|  A  |  B  |  C  |  D  |  A  |  B  |  D  |  A  |  B  |  A  |  B  |
0     5    10    15    20    25    30    35    40    45    50    55
```

Waiting time JA = 15+10+5=30
JB = 3+15+10+5=33
JC = 5
JD = 5+10=15
Average waiting time= (30+33+5+15)/4=20.75
Average response time= (50+53+10+25)/4=30.75

5.



6.

$vruntime_A = vruntime_B$

$\Leftrightarrow vruntime_A + weight_0/weight_A * runtime_A = vruntime_B + weight_0/weight_B * runtime_B$

$\Leftrightarrow weight_0/weight_A * runtime_A = weight_0/weight_B * runtime_B$

$\Leftrightarrow runtime_A = weight_A/weight_B * runtime_B$

$\Rightarrow runtime_A = 3121/335 * runtime_B$

$\Rightarrow runtime_A \approx 9.316 * runtime_B$

the $weight_A$ is greater than $weight_B$

So, process A has the longest actual time on CPU.