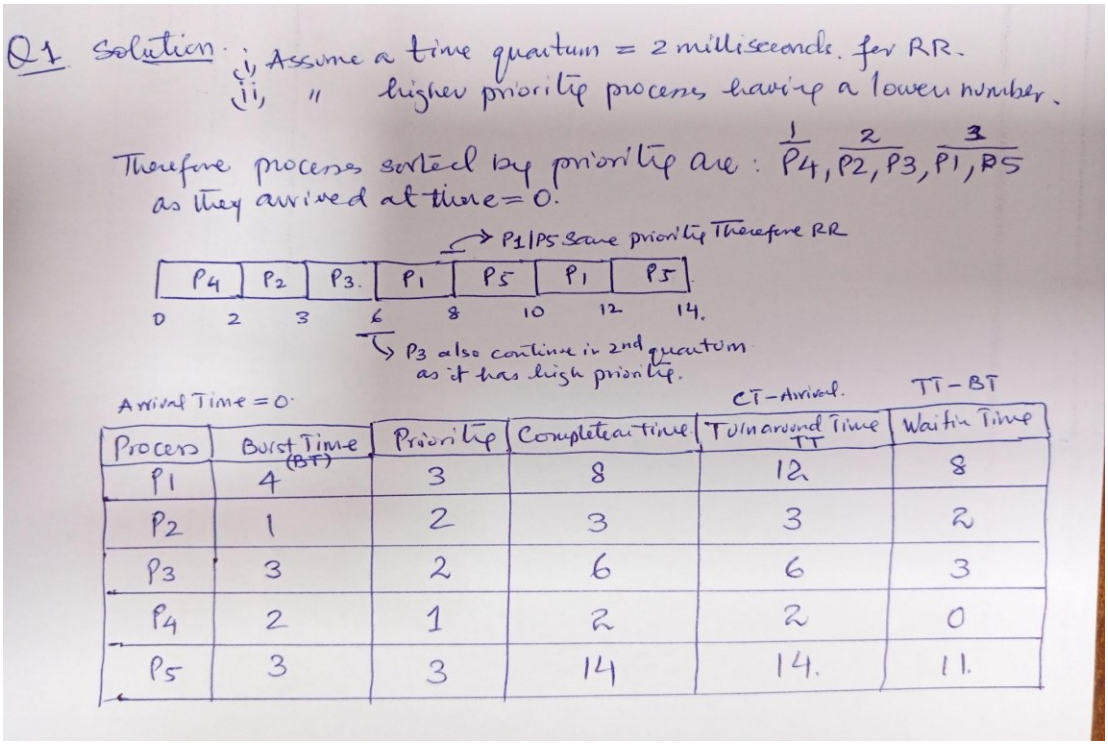
Q1 [10]

(A) Suppose we need to combine round-robin and priority scheduling in such a way that the system executes the highest-priority process and runs processes with the same priority using round-robin scheduling. Draw the Gantt chart and a table showing waiting and turnaround times using the following set of processes that arrive at time =0 and with the burst time in milliseconds as shown.

Processes	Burst Time	Priority
P1	4	3
P2	1	2
P3	3	2
P4	2	1
P5	3	3



Q1 Solution:
i) Assume a time quantum = 2 milliseconds for RR.
ii) " higher priority process having a lower number.

Therefore processes sorted by priority are: P4 (1), P2 (2), P3, P1, P5 (3)
as they arrived at time = 0.

P1/P5 same priority Therefore RR

P4	P2	P3	P1	P5	P1	P5
0	2	3	6	8	10	12	14

P3 also continue in 2nd quantum as it has high priority.

Arrival Time = 0

Process	Burst Time (BT)	Priority	Completion time	Turnaround Time (CT − Arrival) TT	Waiting Time TT − BT
P1	4	3	8	12	8
P2	1	2	3	3	2
P3	3	2	6	6	3
P4	2	1	2	2	0
P5	3	3	14	14	11

Q2[10]

(A)

1. What is a thread-join operation? **[2]**
   A thread-join operation allows the master thread to wait for the created child thread to finish.

2. Consider the following two threads, to be run **concurrently** in a shared memory (all variables are shared between the two threads): Assume a single-processor system, that load and store are atomic, that x is initialized to 0 before either thread starts, and that x must be loaded into a register before being incremented (and stored back to memory afterwards). The following questions consider the final value of x after both threads have completed. **[2*4=8]**

Thread A	Thread B

for (i=0; i<5; i++)
{
x = x + 1;
}
	for (j=0; j<5; j++)
	{
	x = x + 2;
	}

i) Give a concise proof why x≤15 when both threads have completed.
Given the atomic statements x=x+1 and x=x+2 and alternate execution of threads x takes values based on which tread scheduled to runs first. In case TA runs, it takes values: 1,3,4,6,7,9,10,12,13,15 and for TB running first values are: 2,3,5,6,8,9,11,12,14. So the code shown for TA and TB will always remain <= 15.

ii) Give a concise proof why x≠1 when both threads have completed.
X is initially zero. Complete execution of both threads guarantees that value of x will be either 14 or 15 but not 1. See part (i) for more details.

iii) What needs to be saved and restored on a context switch between two threads in the same process? What if the two threads are in different processes? Be explicit.
All registers (this automatically includes stack pointer) will be stored in TCB and restored. In case threads belongs to two different processes PCBs will saved and restored as well. *Please note that we have not covered thread scheduling beyond this.*

iv) Under what circumstances can a multithreaded program complete more quickly than a non-multithreaded program? Keep in mind that multithreading has context-switch overhead associated with it.
Creating too many threads (e.g. ~100s) each running few instructions will slow things down due to context switching overhead. Therefore, multithreading benefits if a program has sizable parallelizable portions and each of those has ~10s of language statements executing cpu/io bound tasks. This will make total context switch time (in ~10 microseconds per context switch) negligible.

Q3 [10]
(A)
Hint: no-progress problem: No process running outside its critical region may block other processes from entering their critical region.

i. Provide a sample pseudo code of two processes that illustrate the **no-progress problem** ~~when using 'Load and Store Solution'~~ for the critical section problem?

P1
```
enter=0;
do {
  // do some work
  while (enter != 0);
  // Critical Section
  for (j=0; j<5; j++) x = x + 2;
  enter=1; // allow other
  // Remainder section
} while (1)
```

P2
```
enter=1;
do {
  // do some work
  while (enter != 1);
  // Critical Section
  for (j=0; j<5; j++) x = x + 2;
  enter=0; // allow other
  // Remainder section
} while (1)
```

No-progress: The process, which executes its remainder section quicker than other will enter the Critical Section (CS) again without giving other a chance. This violates progress requirement of CS problem.

ii. Additionally, can you explain how Peterson's solution provides an elegant solution to this problem by using pseudo code?
Peterson's solution (also for two processes) uses two variable turn and flag: "The variable turn indicates whose turn it is to enter its critical section. The flag array is used to indicate if a process is ready to enter its critical section." The unique feature of Peterson's solution is that each process (even if it completes its remainder section earlier than other) set the turn variable value to ID of other process in order to allow it to enter the critical section if busy waiting. This altruistic (caring for others) approach help progress avoiding no-progress.

(B) [5 points] There are two concurrent programs, and you must choose one to run. You currently have 16 processors to gauge performance, but you will run on 1024 processors. The following information is collected. Which programme will give the best performance?

Program A			**Program B**		
Processors	Speedup	Serial program	Processors	Speedup	Serial program
2	1.88	0.07	**2**	1.79	0.13
4	3.12	0.10	**4**	2.9	0.13
8	4.5	0.12	**8**	4.3	0.13
6	5.8	0.13	**6**	5.7	0.13

Assumptions: i) last entry is 16 processors. Ii) Serial portion of a program never changes with the addition of more processors.
The serial portion of both programs is same i.e. 0.13, thus parallel portion will be 0.8 for both. Therefore, we can calculate values for 16 and 1024 processors using Amdahl's law equation. Same value represents that any program can be used.