Question 1. Pseudocode for Multi-Level Queue Dispatcher

1. Intialise the queues (RT job dispatch Q, Normal job dispatch Q, **Level 0 Q**, **Level 1 Q** and **Level 2 Q**)
2. Populate RT job dispatch Q and Normal job dispatch Q from the input file
3. Ask the user to enter an integer value for **time_quantum**
4. While there is a currently running process or any of the queues are not empty
    i. Unload all arrived processes into their respective Level Qs
    ii. If there is a process running
        a. If the priority value is **0** (**Level 0** process running)
            A. **SIGINT** to terminate process
            B. Calculate turnaround_time and wait_time for process
            C. Free process struct memory
        b. If the priority value is **1** (**Level 1** process running)
            A. Decrease <remaining_cpu_time> by **time_quantum**
            B. If time has been exhausted
                - **SIGINT** to terminate process
                - Calculate turnaround_time and wait_time for process
                - Free process struct memory
            C. Else
                - **SIGTSTP** to suspend process
                - Set priority to **2**
                - Enqueue to tail of **Level 2 Q**
        c. If the priority value is **2** (**Level 2** process running)
            A. Decrease <remaining_cpu_time> by 1
            B. If the time has been exhausted
            C. If there is another process in **Level 0 Q**
                - **SIGTSTP** to suspend process
                - Enqueue to head of **Level 2 Q**
                - Dequeue process from the head of **Level 0 Q**
                - Start and set as the currently running process
            D. If there is another process in **Level 1 Q**
                - **SIGTSTP** to suspend process
                - Enqueue to head of **Level 2 Q**
                - Dequeue process from the head of **Level 1 Q**
                - Start and set as the currently running process
            E. Else
                - If time has been exhausted
                    - **SIGINT** to terminate process
                    - Calculate turnaround_time and wait_time for process
                    - Free process struct memory

iii. If there is no process running and at least one of the Level Qs aren't empty (we schedule the next job with highest priority)
   a. If **Level 0 Q** is not empty
      A. Dequeue process from the head of **Level 0 Q**
      B. Start and set as the currently running process
   b. Else if **Level 1 Q** is not empty
      A. Dequeue process from the head of **Level 1 Q**
      B. Start and set as the currently running process
   c. Else if **Level 2 Q** is not empty
      A. Dequeue process from the head of **Level 2 Q**
      B. If the process is a suspended process
         - Send **SIGCONT** to resume it
         - and set as the currently running process
      C. Else
         - Start and set as the currently running process

iv. If there is a current process
   a. Sleep for **decrease_time** (different for each level process)
   b. Increase timer by **decrease_time**

v. Else
   a. Sleep for 1
   b. Increase timer by 1

vi. Go back to 4.

Question 2. Example and Explanation of 3 'Job Dispatch List Files' and their Outputs

Example 1. Testing **Level 0 Q**

The purpose of this example is to **test that the Level 0 Q is always prioritised**. For this example we assume a **time_quantum** of 3 and then compare the results with 4 and 5.

Process	<arrival time>	<cpu_time>	<priority>
1	0	5	1
2	4	2	0
3	6	3	1
4	9	4	0
5	10	3	1
6	11	2	1

When this example is first read in, assuming that the initial initialisation of queues are correct, process 1 will already have been dispatched and enqueued onto the **Level 1 Q** since it has an <arrival time> of 0. It will then be run for 3 seconds (**time_quantum**), then pre-empted and enqueue onto the **Level 2 Q**. Since process 1 is at the front of the Level 2 Q and there are no other processes with higher priority, it will run for 1 second until process 2 arrives. Here the program should preempt process 1 and allow process 2 to run until completion as its priority is 0. Then process 1 waits until all the other processes finish running as they all have higher priority, and none of them ever get appended to the level 2 Q because they all have a <cpu_time> of less than **time_quantum**. Finally process 1, the last process, runs for the <remaining_cpu_time> of 1 second.

A **time_quantum** of 3 is selected to ensure that process 2, the job with higher priority, interrupts and enforces the program to preempt process 1 and start running process 2 at the time of its arrival. Furthermore, it ensures that process 4 runs until completion without interruption even though the **time_quantum** is less than its <cpu_time>, ensuring that all jobs with priority 0 are always prioritised.

When running this job list with a **time_quantum** of 5 or greater, the order of jobs remains in a FCFS fashion hence, no preempting is evident thus, we cannot test for priority in level 0.

Expected outcomes:

- **Time Quantum: 3 & 4 (both give the same results)**
  - Order of finishing jobs → 2,3,4,5,6,1
  - Average Turnaround Time → 6.83 seconds
  - Average Wait Time → 3.67 seconds
- **Time Quantum: 5**
  - Order of finishing jobs → 1,2,3,4,5,6
  - Average Turnaround Time → 5.33 seconds
  - Average Wait Time → 2.17 seconds

Example 2. Testing Level 2 Q

The purpose of this example is to **test that the Level 2 Q enqueues jobs back on the head of the queue rather than the tail**. For this example we assume a time_quantum of 3 and then compare the results with 4.

Process	<arrival time>	<cpu_time>	<priority>
1	0	5	1
2	1	5	1
3	6	1	0
4	7	1	0
5	8	1	0
6	10	3	1

When this example is first read in, assuming that the initial initialisation of queues are correct, process 1 will already have been dispatched and enqueued onto the **Level 1 Q** since it has an <arrival time> of 0. It will then be run for 3 seconds (**time_quantum**), then pre-empted and enqueue onto the **Level 2 Q**. Next, just like process 1, process 2 runs for 3 seconds and gets pre-empted onto the **Level 2 Q**. Furthermore, processes 3, 4 and 5 will run to completion in that order because they have priority 0. Upon completion, process 6 hasn't arrived yet however process 1 is at the head of the **Level 2 Q**, therefore, it can run for 1 second before process 6 arrives. After running for 1 second it should enqueue it back onto the head of the **Level 2 Q** (**this is where the test for this specific example occurs**). After process 6 finishes running, process 1 should run again to completion and finally, process 2 should then finish running.

A **time_quantum** of 3 is selected to ensure the test of whether the process gets enqueued back onto the head of the Level 2 can be tested. Any **time_quantum** larger or smaller than 3 will not be able to test this property.

Expected outcomes:

- **Time Quantum: 3**
  - Order of finishing jobs → 3,4,5,6,1,2
  - Average Turnaround Time → 5.83 seconds
  - Average Wait Time → 3.17 seconds
- **Time Quantum: 4**
  - Order of finishing jobs → 3,4,5,6,1,2
  - Average Turnaround Time → 7.17 seconds
  - Average Wait Time → 4.50 seconds

Although the order of finishing jobs are the same, the property is clearly not tested.