Question 1. Pseudocode with Swapping

1. Intialise the queues (Swapped Q, Job dispatcher Q, RT job dispatch Q, Normal job dispatch Q, **Level 0 Q**, **Level 1 Q** and **Level 2 Q**)
2. Create arena (i.e. head of doubly linked list of mab)
3. Populate Job dispatcher Q from input file
4. Ask the user to enter an integer value for **time_quantum**
5. While there is a currently running process or any of the queues are not empty
    i. Unload all arrived processes from Job dispatcher Q into RT or Normal job dispatch Q
    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 and mab
            D. Flag that this iteration involved freeing of a process
        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 and mab
                - Flag that this iteration involved freeing of a process
            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
                - **SIGINT** to terminate process
                - Calculate turnaround_time and wait_time for process
                - Free process struct memory and mab
                - Flag that this iteration involved freeing of a process
            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 and mab
  - Flag that this iteration involved freeing of a process

iii. Unload all jobs that can be admitted from RT or Normal Qs into their respective Level Qs (i.e. check if there is free memory and check if there are **Level 2** jobs)

a. Check if there is a first fit

b. If there exists a first fit then

A. Swap out largest job, free **Level 2** job from memory and allocate **Level 0** job to memory

c. If there is no first fit then

A. If current process is a **Level 2** job and **Level 2 Q** is empty

- Swap out and free current (**Level 2**) job from memory
- Allocate **Level 0** job to memory
- Set current process to null

B. If current process is a **Level 2** job and the **Level 2 Q** is **not** empty

- If the current job's <remaining_cpu_time> is larger than the largest <remaining_cpu_time> in **Level 2 Q**
  - Swap out and free current (**Level 2**) job
- Else
  - Swap out and free largest (**Level 2**) job
- Allocate **Level 0** job to memory
- Set current process to null

C. If there is no current process and the **Level 2 Q** is **not** empty

- Swap out and free largest (**Level 2**) job
- Allocate **Level 0** job to memory

d. Allocate jobs from the Normal Q into the **Level 1 Q**

iv. If no process was freed this iteration

a. If there is no process running and at least one of the Level Qs or Swapped Q isn't empty (we schedule the next job with highest priority)

A. If **Level 0 Q** is not empty

- Dequeue process from the head of **Level 0 Q**
- Start and set as the currently running process

B. Else if **Level 1 Q** is not empty

- Dequeue process from the head of **Level 1 Q**
- Start and set as the currently running process

C. Else if **Level 2 Q** is not empty

- Dequeue process from the head of **Level 2 Q**
- If the process is a suspended process

        - Send **SIGCONT** to resume it
        - and set as the currently running process
    - Else
        - Allocate memory to memory linked list
        - Start and set as the currently running process

D. Else if Swapped Q is not empty

- Dequeue process from head of Swapped Q
- Allocate memory to memory linked list
- Start and set as the currently running process

b. If there is a current process

A. Sleep and increase timer by appropriate time

c. Else

A. Sleep and increase timer by 1

v. Unflag the freed variable

vi. Go back to 4.

vii. Calculate the average turnaround and wait times

viii. Terminate dispatcher

Question 2. Example and Explanation of 2 ‘Job Dispatch List Files’ and their Outputs

Example 1. Testing the “Swapping Out/In” property

Process	<arrival time>	<cpu_time>	<priority>	<memory>
1	0	7	1	1023
2	3	2	0	32
3	6	2	0	32
4	9	2	0	32
5	12	2	0	32
6	14	1	1	500

When inputting a **time_quantum** of 3, the follow things are tested:

- Process 1 swaps out at the correct time
    - Here process 1 should swap out at time 3, 6, 9, and 12
- All processes in the level Qs have higher priority than processes in the swapping Q
- That swapping out only occurs for jobs with priority 2
- Furthermore, it also ensures that all tests from part 2 are still working

Inputting a **time_quantum** of 2 also tests for the same properties with more emphasis on the fact that the programs runs correctly. However too large of a **time_quantum** doesn’t test anything (e.g. 7 or greater).

Expected Outcomes:

- **Time Quantum: 3**
    - Order of jobs finishing → 2,3,4,5,6,1
    - Average Turnaround Time → 4.17 seconds
    - Average Wait Time → 1.5 seconds
- **Time Quantum: 2**
    - Order of jobs finishing → 2,3,4,5,6,1
    - Average Turnaround Time → 4.17 seconds
    - Average Wait Time → 1.5 seconds
- **Time Quantum: 7**
    - Order of jobs finishing → 1,2,3,4,5,6
    - Average Turnaround Time → 4.5 seconds
    - Average Wait Time → 1.83 seconds

Example 2. Testing for priority, offsets and SIGCONT

Process	<arrival time>	<cpu_time>	<priority>	<memory>
1	0	5	1	32
2	0	5	1	32
3	0	5	1	32
4	0	5	1	32
5	0	5	1	896
6	1	5	0	32

When inputting a **time_quantum** of 1, the follow things are tested:

- The processes all swap out at the correct time
    - Process 1 should swap out at the beginning of the program and swap back in at the very end
    - Process 2 should be the first non-level 0 job to stop running
    - Followed by process 3,4,5
- SIGCONT signals are made at the correct time
- All the offsets are correct and assigned at the beginning as all jobs arrive at time 0 and have small enough memory
- All processes in the level Qs have higher priority than processes in the swapping Q
- That swapping out only occurs for jobs with priority 2
- Furthermore, it also ensures that all tests from part 2 are still working
- Moreover, by passing these tests it should have also covered the functions I implemented in pcb.c and mab.c as these tests wouldn't work if they weren't working

Inputting a **time_quantum** of 4 further tests the same properties as mentioned above, giving a more robust test.

Expected Outcomes:

- **Time Quantum: 1**
    - Order of jobs starting → 1,2,3,4,5,6
    - Order of jobs finishing → 2,3,4,5,6,1
    - Average Turnaround Time → 19.17 seconds
    - Average Wait Time → 14.17 seconds
- **Time Quantum: 4**
    - Order of jobs starting → 1,2,3,4,5,6
    - Order of jobs finishing → 2,3,4,5,6,1
    - Average Turnaround Time → 24.67 seconds
    - Average Wait Time → 19.67 seconds