Part I.a) : FCFS (First Come First Serve) Algorithm: (Unsure if I also had to explain but included it)

FCFS is a non-preemptive CPU scheduling algorithm that essentially executes processes in the order they arrive in the ready queue. So they follow the concept of First In First Out (FIFO).

The steps of FCFS are:

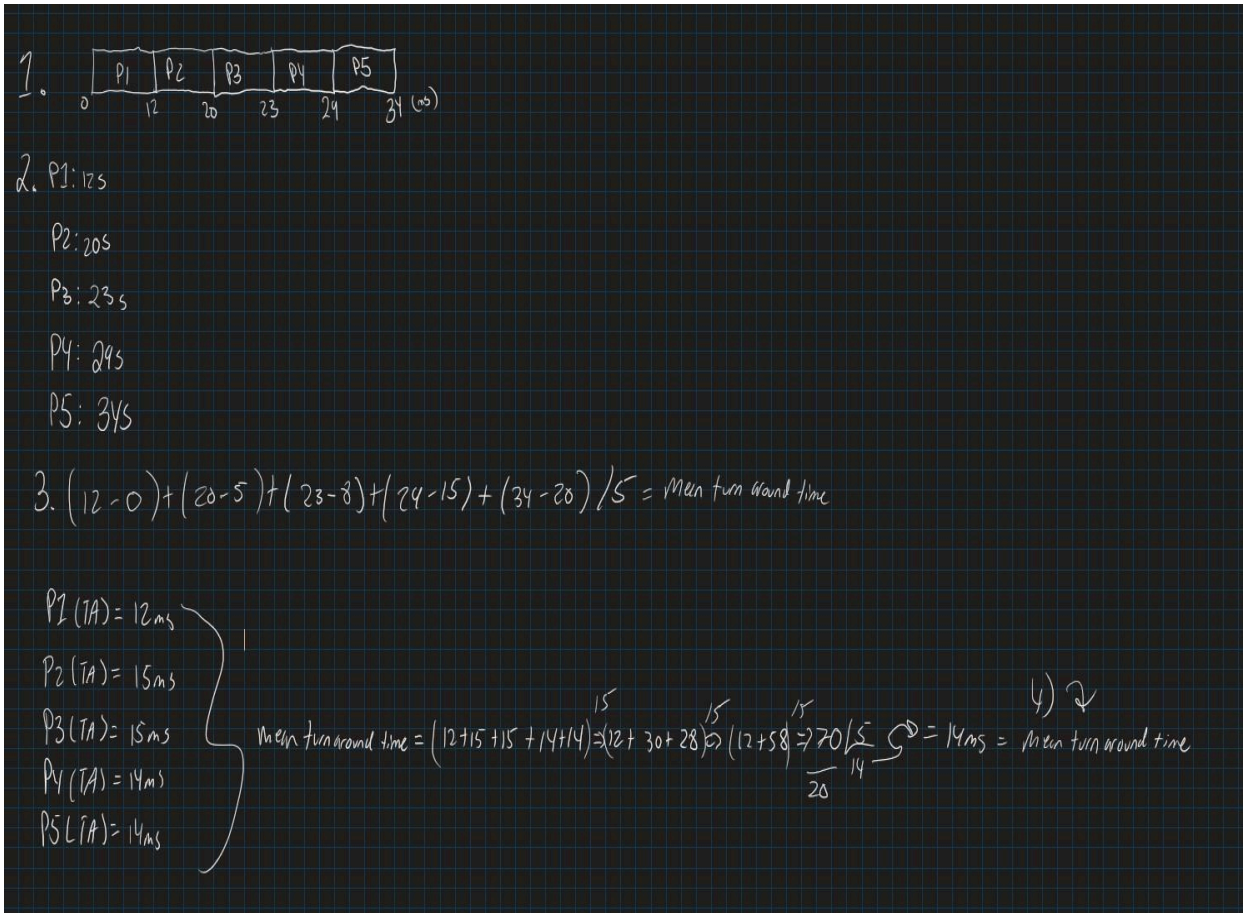
1. The processes are added to the ready queue in their order of arrival
2. The CPU scheduler (short-term scheduler) will then select the process at the head of the queue.
3. The selected process will execute until its completion or I/O request.
4. When completed or blocked, the next process in the ready queue will get the CPU.
5. This process will continue until all the processes are complete.

It is important to take into consideration those characteristics in FCFS. First, non-preemptive means that once a process gets the CPU, it will keep running until it terminates (so completion) or until it blocks/waits due to I/O (leaves CPU to wait for I/O). Also, every process gets CPU time. Since every process is served strictly in arrival order, it could lead to convoy effect. This occurs when a large process takes on the CPU for a large amount of time leaving a trail of smaller processes waiting for its completion (not efficient).

When a process requests for I/O, its state changes from running to waiting. Then the process is moved from the CPU to the I/O waiting queue. After that, the next process in ready queue gets the CPU. When the I/O completes, the process will return to the tail of ready queue and the state will change from waiting to ready.

Advantages of FCFS are that it's simple to implement and understand and that it has a predictable execution order. The disadvantages are the convoy effect and longer average turnaround times.

Gantt Chart + Calculations :



ii)

Round Robin is a preemptive CPU scheduling algorithm and it is made to give each process a fair slic oc CPU time. It is one of the moste used scheduling techniques in time-sharing systems. It enforces a strict time quantum/time slice for each process.

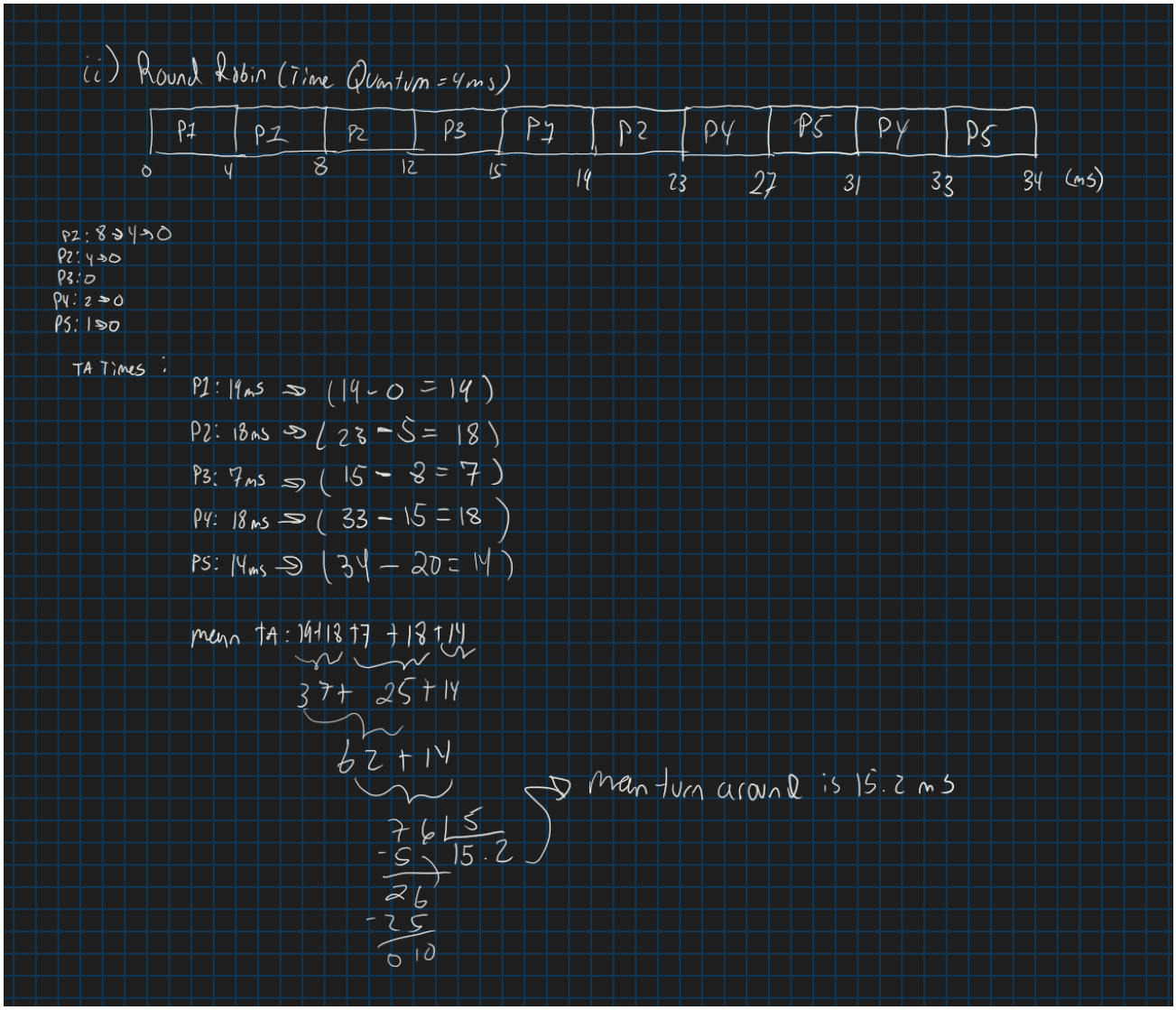
The steps of Round Robin are:

1. Processes enter the ready queue in their order of arrival, just like FCFS.
2. The CPU scheduler then chooses the process at the head of the ready queue.
3. The selected process then executes for at most one time quantum. If the process completes before the quantum expires, it will leave the system. If the process requests for I/O before the quantum ends, it will black and move to the i/o waiting queue. If the quantum ends and the process is not finished, it will be preempted and placed at the tail of the ready queue.
4. The CPU then chooses the next process at the head of the queue.
5. When a blocked process terminates its I/O, it returns to the tail of the ready queue.
6. All of this will happen over and over again until all processes are complete.

It is important to take into account a few aspects in Round Robin. First, a process can't monopolize the CPU due to the preemptive nature and its fairness. Once the time slice is over, the current process is preempted and the next process runs for its time slice. There are also more context switches because processes are preempted often. Finally, when a process requires I/O, it leaves teh CPU and the next process runs while the I/O occurs. Once the I/O is complete, the process goes back to the tail of the ready queue.

Advantages of Round Robin are first that each process gets equal CPU time slices, which is great in terms of interactive systems. Short processes can also finish quickly and there are no convoy effects.

Disadvantages of Round Robin include high amounts of context switch, which leads to high context switch overhead due to frequent preemptions. This can reduce efficiency. Also we have to pick a right quantum because if too large, RR could act like FCFS. If too small, it could act way too many context switches.



iii)

(iii) Shortest job first with preemption

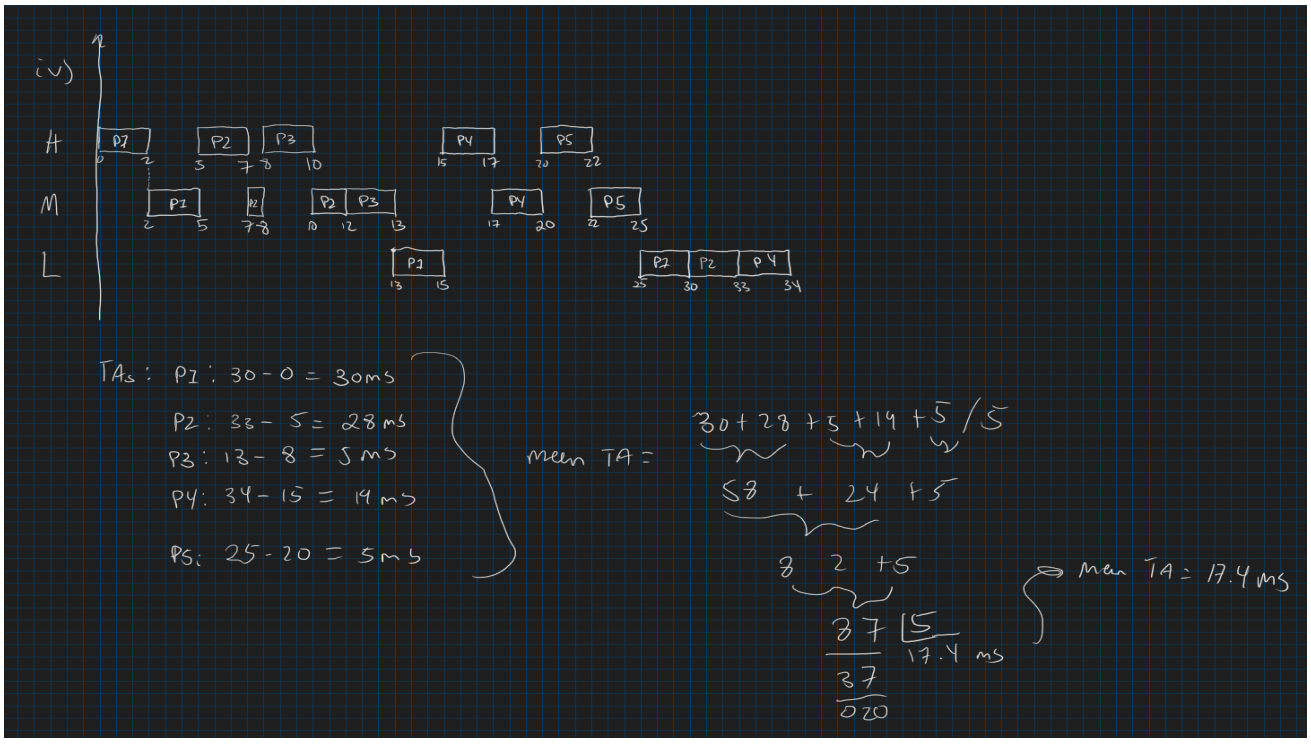
P1	P3	P1	P4	P5	P2

0 8 11 15 21 26 34 ← Gantt chart

Completion Time	Turnaround Time
P1 : 15	P1 : 15 - 0 = 15
P2 : 34	P2 : 34 - 5 = 29
P3 : 11	P3 : 11 - 8 = 3
P4 : 21	P4 : 21 - 15 = 6
P5 : 26	P5 : 26 - 20 = 6

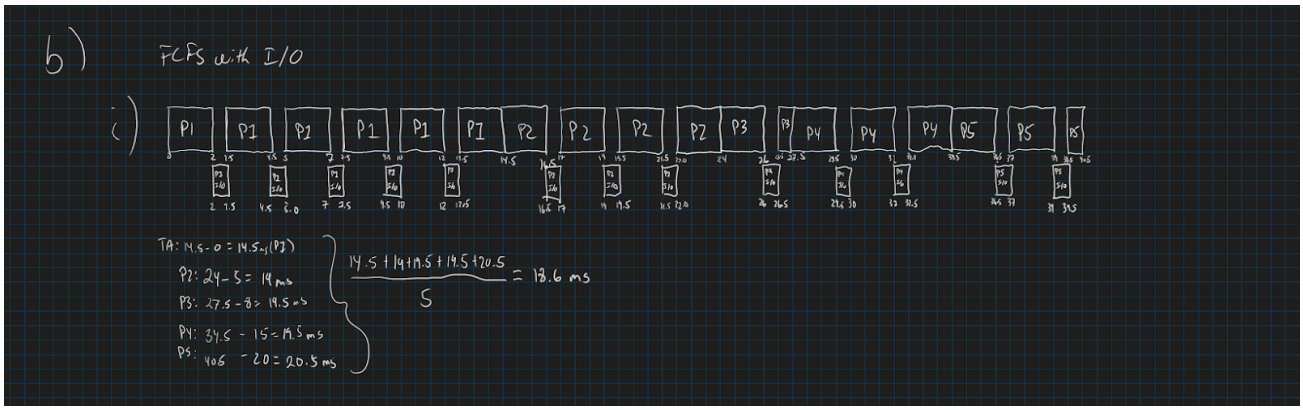
Mean turnaround time
= 15 + 29 + 3 + 6 + 6 / 5
⇒ 11.8 ms

iv)

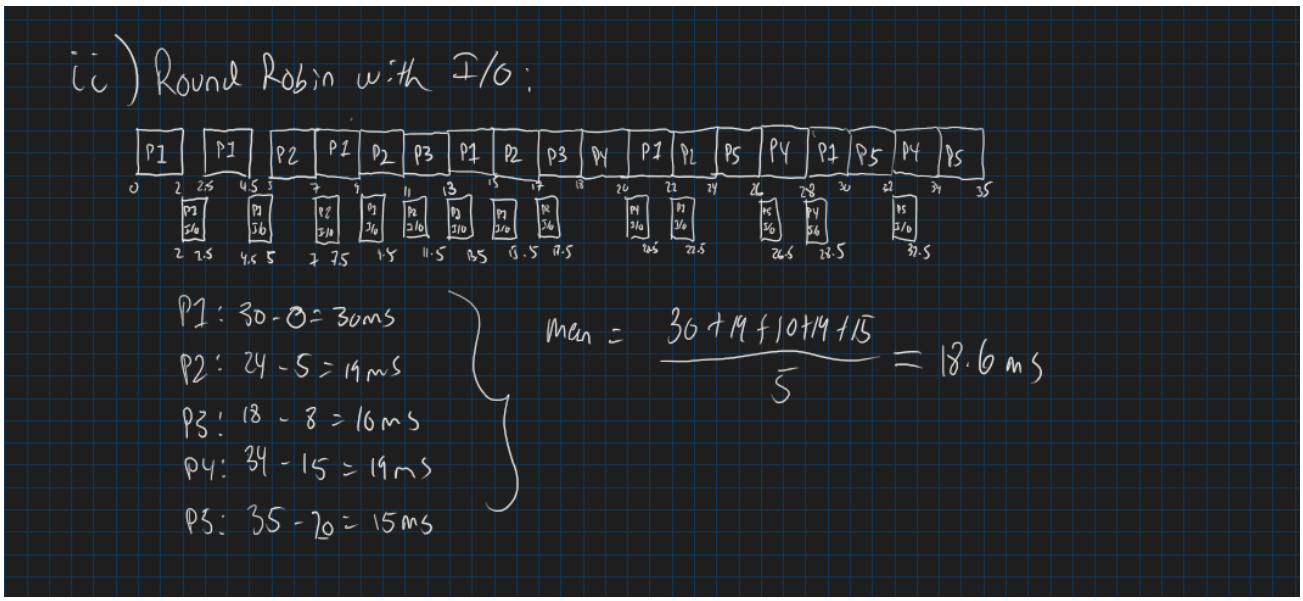


b)

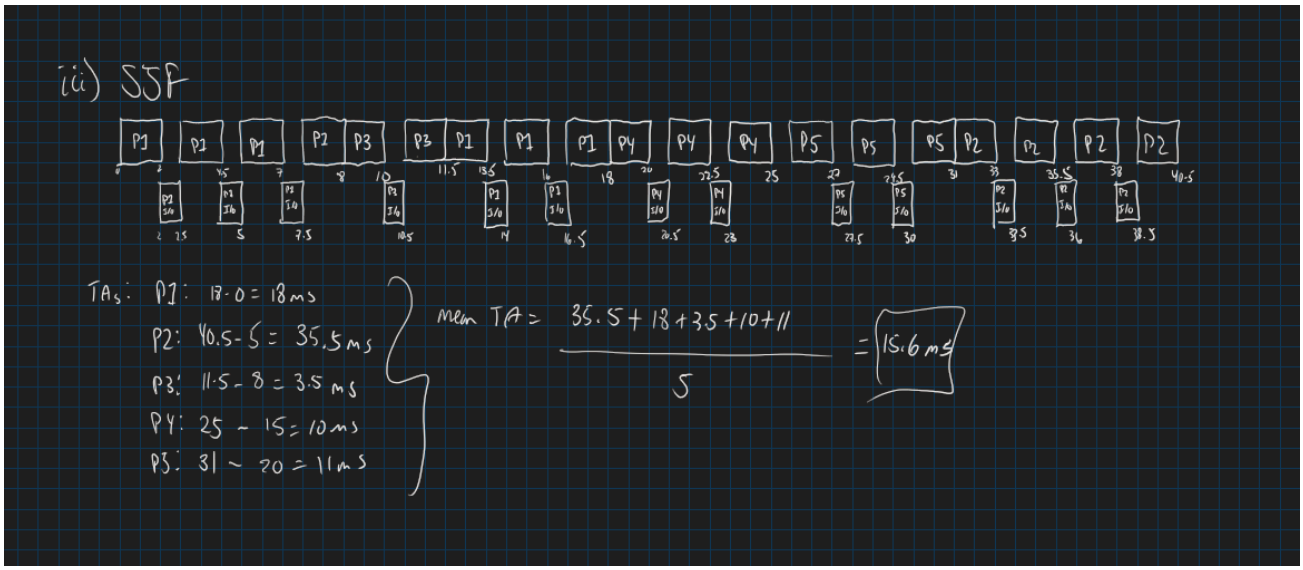
i)


b)
FCFS with I/O
i)
TA: 14.5 - 0 = 14.5 ms (P1)
P2: 24 - 5 = 19 ms
P3: 27.5 - 8 = 19.5 ms
P4: 34.5 - 15 = 19.5 ms
P5: 40.5 - 20 = 20.5 ms
14.5 + 19 + 19.5 + 19.5 + 20.5 / 5 = 18.6 ms

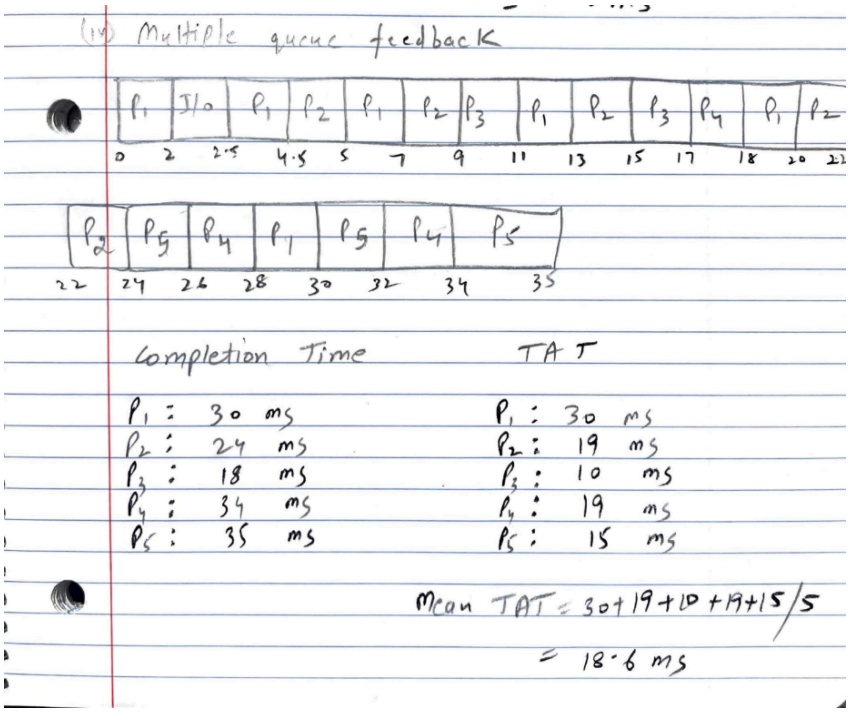
ii)


ii) Round Robin with I/O:
P1: 30 - 0 = 30ms
P2: 24 - 5 = 19ms
P3: 18 - 8 = 10ms
P4: 34 - 15 = 19ms
P5: 35 - 20 = 15ms
mean = 30 + 19 + 10 + 19 + 15 / 5 = 18.6 ms

iii)


iii) SJF
TAs: P1: 18 - 0 = 18ms
P2: 40.5 - 5 = 35.5ms
P3: 11.5 - 8 = 3.5 ms
P4: 25 - 15 = 10ms
P5: 31 - 20 = 11ms
mean TA = 35.5 + 18 + 3.5 + 10 + 11 / 5 = 15.6 ms

iv)


(iv) Multiple queue feedback
P1 I/O P1 P2 P1 P2 P3 P1 P2 P3 P4 P1 P2
0 2 2.5 4.5 5 7 9 11 13 15 17 18 20 22
P2 P5 P4 P1 P5 P4 P5
22 24 26 28 30 32 34 35
Completion Time
P1 : 30 ms
P2 : 24 ms
P3 : 18 ms
P4 : 34 ms
P5 : 35 ms
TAT
P1 : 30 ms
P2 : 19 ms
P3 : 10 ms
P4 : 19 ms
P5 : 15 ms
Mean TAT = 30+19+10+19+15/5
= 18.6 ms

c)

C)

Initial free list (size in KB)

1: 85
2: 340
3: 28
4: 195
5: 55
6: 160
7: 75
8: 280

Total initial free memory = 85+340+28+195+55+160+75+280 = 1,218 KB

Jobs will arrive in order of:
J1 = 140 KB, J2 = 82 KB, J3 = 275 KB, J4 = 65 KB, J5 = 190 KB.
Total requested = 140+82+275+65+190 = 752 KB.
So final total free memory after all allocations (if all allocated) = 1218 − 752 = 466 KB

1) Allocation tables for each algorithm is mentioned below

(i) First Fit

Let's see the free list from position 1 and pick the first hole that is large enough for our job.

- J1 = 140

  340 (position 2) → allocate 140 from hole 2.
  Hole2 new size = 340 − 140 = 200.

- J2 = 82
  85 (position 1) → allocate from hole1.
  Hole1 new size = 85 − 82 = 3.

- J3 = 275
  280 (yes, position 8) → allocate from hole8.
  Hole8 new size = 280 − 275 = 5.

- J4 = 65
  200 (pos2) → allocate from hole2.
  Hole2 new size = 200 − 65 = 135.

- J5 = 190
  195 (pos4) → allocate from hole4.
  Hole4 new size = 195 − 190 = 5

  Final free list after First Fit:

- `[3, 135, 28, 5, 55, 160, 75, 5]`

  After allocations our final free holes are mentioned below
  Final free holes: 3, 135, 28, 5, 55, 160, 75, 5 (KB)

- Total free memory = 3 + 135 + 28 + 5 + 55 + 160 + 75 + 5 = 466 KB it matches 1218 – 752 so our calculations are right.
- Internal fragmentation = 0 KB because we split holes to exact job sizes.
- External fragmentation = 466 KB.

(ii) Best Fit

Let's allocate each job into the smallest free hole that is greater than or equal to our job size.

Step-by-step

- J1 = 140
  Allocate 140 from hole6 → hole6 becomes 20.
- J2 = 82
  Allocate from hole1 → hole1 becomes 3.
- J3 = 275
  Allocate from hole8 → hole8 becomes 5.
- J4 = 65
  Allocate from hole7 → hole7 becomes 10.
- J5 = 190
  Allocate from hole4 → hole4 becomes 5.

  After allocating our final free holes are mentioned below

- `[3, 340, 28, 5, 55, 20, 10, 5]`

  After allocations:

- Final free holes: 3, 340, 28, 5, 55, 20, 10, 5 (KB)
- Total free memory = 3+340+28+5+55+20+10+5 = 466 KB.
- Internal fragmentation = 0 KB
- External fragmentation = 466 KB

(iii) Worst Fit

Let's Allocate each job into the largest free hole.

- J1 = 140
  Largest hole is 340 (pos2) → allocate from hole2 → hole2 becomes 200.

- J2 = 82
  Now largest hole is 280 (pos8) → allocate from hole8 → hole8 becomes 198.
- J3 = 275

  No hole ≥ 275 (largest = 200 < 275).
  → J3 cannot be allocated at this time (it is blocked/waits).
  We continue with next arriving jobs (J3 remains unallocated).

- J4 = 65
  Choose the current largest hole (which is 200 at pos2) → allocate 65 from 200 → hole2 becomes 135.
- J5 = 190
  current largest hole is 198 (pos8) (198 ≥ 190) → allocate 190 from 198 → hole8 becomes 8.

Result – Worst Fit

- After allocations performed:
  J1 → hole2
  J2 → hole8
  J3 → NOT ALLOCATED (no hole > 275)
  J4 → hole2
  J5 → hole8
- Final free holes: [85, 135, 28, 195, 55, 160, 75, 8]

- Total free memory after allocations = 85+135+28+195+55+160+75+8 = 741 KB.
- Internal fragmentation = 0 KB.
- External fragmentation = 741 KB.

Key numbers summary

- Total free memory after all allocations
  - First Fit: 466 KB
  - Best Fit: 466 KB
  - Worst Fit: 741 KB (J3 left unallocated)
- Internal fragmentation: 0 KB for all three.
- External fragmentation: values are mentioned above but the more meaningful observation is Worst Fit left enough total free memory to satisfy J3 but fragmented so that no single hole ≥ 275 KB — J3 could not be allocated. That explains external fragmentation effect.

Which algorithm is most memory-efficient?

- Best Fit in this trace did well among all three: it allocated small-to-medium jobs into the smallest adequate holes, preserving a large hole for the large job J3. As a result, all jobs were allocated, and the free memory was distributed.
- First Fit also allocated all jobs, it is simple and reasonably effective here. It tends to be faster to search and often gives comparable performance to best fit.
- Worst Fit performed very bad for this arrival sequence: by placing J2 into the largest hole it destroyed the only sufficiently large hole for J3, so J3 could not be allocated even though total free memory was enough. So Worst Fit can perform badly.

Recommendation (based on workload types)

1. System with frequent small allocations
   - Best Fit is generally preferable for frequent small allocations because it places each small job into the smallest hole that fits and keeping larger holes available for bigger future requests. However, best-fit can create many very small holes and may incur more search overhead.
   - First Fit is a good, fast alternative and often nearly as effective.
   - Worst Fit is not usually ideal for many small allocations.
2. System with mixed workload sizes (small + large requests)
   - First Fit often gives a good trade-off between speed and memory utilization and avoids some pathologies.
   - Best Fit can be good if you can afford the search cost and want to preserve large holes for large jobs.
   - Worst Fit can be risky: if large jobs arrive later, worst-fit's can be bad choice to pick