1.

a.

Assignment #2

1

a)

	Max	-	Allocation	=	Need
P_0	[0,0,1,2]	-	[0,0,1 2]	=	[0,0,0,0]
P_1	[1,7,5,0]	-	[1,0,0,0]	=	[0,7,5,0]
P_2	[2,3,5 6]	-	[1,3,5,4]	=	[1,0,0,2]
P_3	[0,6,5,2]	-	[0,6,3,2]	=	[0,0,2,0]
P_4	[0,6,5,6]	-	[0,0,1,4]	=	[0,6,4,2]

Assignment #2

b.

1. b)

Available	Allocation	Need
[1,5,2,0]	P_0 [0,0,1,2]	P_0 [0,0,0,0]
	P_1 [1,0,0,0]	P_1 [0,7,5,0]
	P_2 [1,3,5,4]	P_2 [1,0,0,2]
	P_3 [0,6,3,2]	P_3 [0,0,2,0]
	P_4 [0,0,1,4]	P_4 [0,6,4,2]

- Work = [1,5,2,0] = Available
- Safety Sequence = [P_0, P_2, P_3, P_4, P_1]

i_0 $Need_0 \leq Work$

[0,0,0,0] ≤ [1,5,2,0]

Work = [1,5,2,0] + $Allocation_0$ = [1,5,2,0] + [0,0,1,2]

Work = [1,5,3,2]

i_1 $Need_1 \leq Work$

[0,7,5,0] $\nleq$ [1,5,3,2]

i_2 $Need_2 \leq Work$

[1,0,0,2] ≤ [1,5,3,2]

Work = Work + $Allocation_2$ = [1,5,3,2] + [1,3,5,4]

Work = [2,8,8,6]

i_3 $Need_3 \leq Work$

[0,0,2,0] ≤ [2,8,8,6]

Work = Work + $Allocation_3$ = [2,8,8,6] + [0,6,3,2]

Work = [2,14,11,8]

i_4 $Need_4 \leq Work$

[0,6,4,2] ≤ [2,14,11,8]

Work = Work + $Allocation_4$ = [2,14,11,8] + [0,0,1,4]

Work = [2,14,12,12]

i_1 $Need_1 \leq Work$

[0,7,5,0] ≤ [2,14,12,12]

- The system is in a safe state the safe sequence is [P_0, P_2, P_3, P_4, P_1].

c.

Assignment #2

1. c) P_1 Request = [0,4,2,0]

- $Request_1 \le Need_1$

[0,4,2,0] ≤ [0,7,5,0]

- $Request_1 \le Available$

[0,4,2,0] ≤ [1,5,2,0]

Possible New State

Work = New Available = [1,5,2,0] − [0,4,2,0] = Available − $Request_1$

Work = [1,1,0,0]

$Need_1$ = $Need_1$ − $Request_1$ = [0,7,5,0] − [0,4,2,0]

$Need_1$ = [0,3,3,0]

$Allocation_1$ = Allocation + $Request_1$

= [1,0,0,0] + [0,4,2,0]

$Allocation_1$ = [1,4,2,0]

	New Need Matrix	New Allocation Matrix
P_0	[0,0,0,0]	[0,0,1,2]
* P_1	[0,3,3,0]	[1,4,2,0]
P_2	[1,0,0,2]	[1,3,5,4]
P_3	[0,0,2,0]	[0,6,3,2]
P_4	[0,6,4,2]	[0,0,1,4]

* Work = [1,1,0,0]

Safety Algo   Safety Seq = [P_0, P_2, P_3, P_4, P_1]

i_0 $Need_0 \le Work$

[0,0,0,0] ≤ [1,1,0,0]

Work = Work + $Allocation_0$ = [1,1,0,0] + [0,0,1,2]

Work = [1,1,1,2]

i_1 $Need_1 \le Work$

[0,3,3,0] ≰ [1,1,1,2]

i_2 $Need_2 \le Work$

[1,0,0,2] ≤ [1,1,1,2]

Work = Work + $Allocation_2$ = [1,1,1,2] + [1,3,5,4]

Work = [2,4,6,6]

Assignment #2

1. c)

i_3 $Need_3 \le Work$

[0,0,2,0] ≤ [2,4,6,6]

Work = Work + $Allocation_3$ = [2,4,6,6] + [0,6,3,2]

Work = [2,10,9,8]

i_4 $Need_4 \le Work$

[0,6,4,2] ≤ [2,10,9,8]

Work = Work + $Allocation_4$ = [2,10,9,8] + [0,0,1,4]

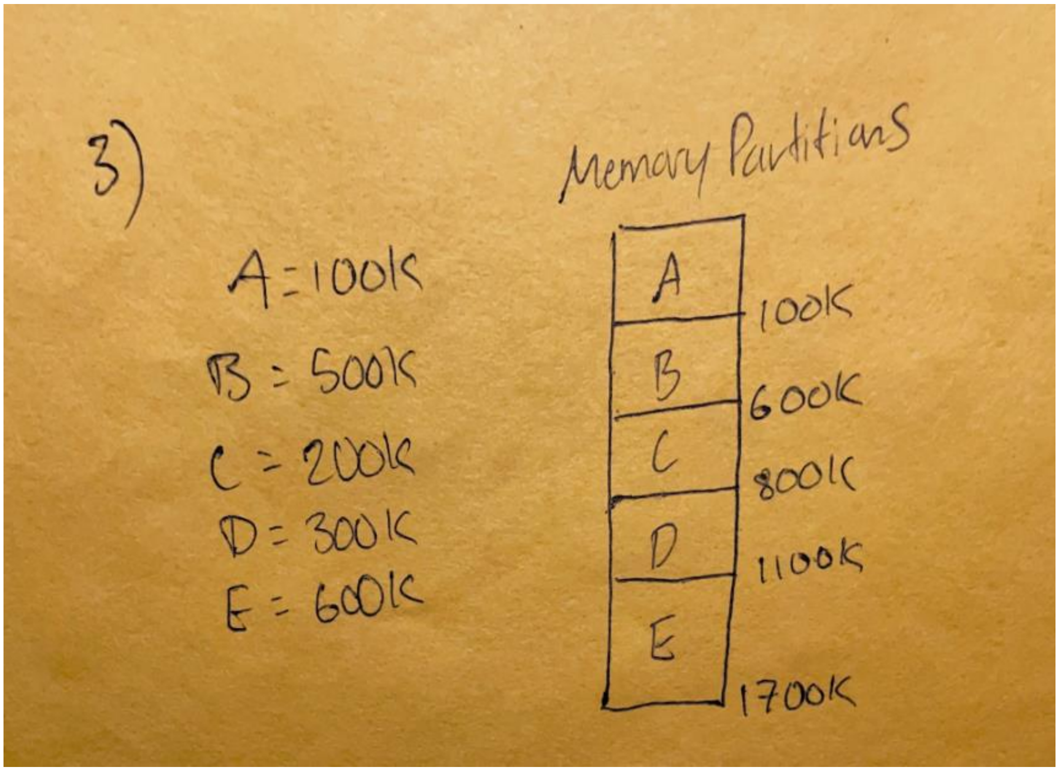
Work = [2,10,10,12]

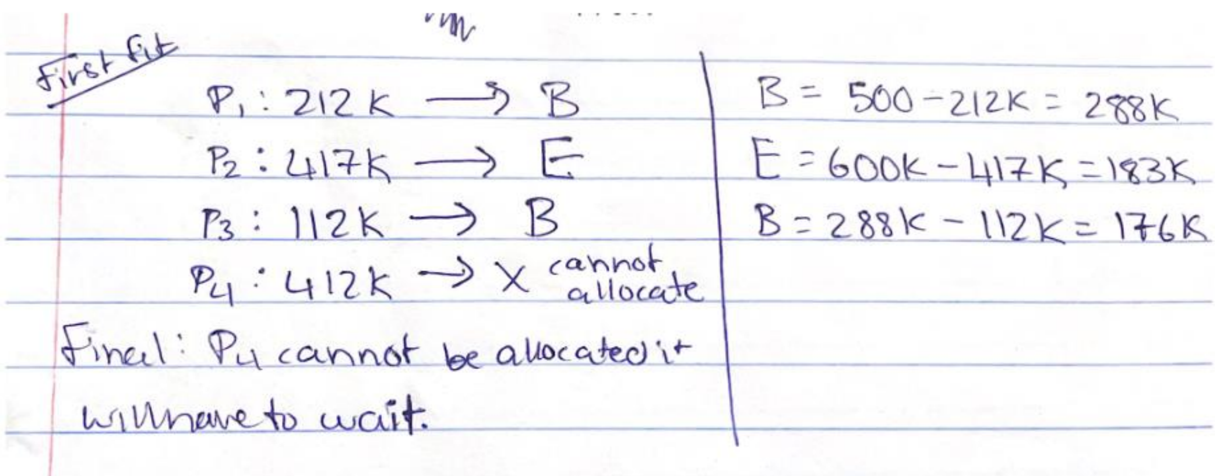
i_1 $Need_1 \le Work$

[0,3,3,0] ≤ [2,10,10,12]

- $Request_1$ can be granted because it leaves the system in a safe state. $Request_1$ = [0,4,2,0]
The safe sequence is [P_0, P_2, P_3, P_4, P_1]

2. The three requirements of the critical-section problem are mutual exclusion is assured, progress is assured, and bounded waiting is assured. Dekker's algorithm satisfies all three of these requirements. When Pi is entering its critical section flag[i] will be true and turn == i so the code for Pj will spin in the while loop and not allow Pj to enter its critical section until Pi is finished therefore **mutual exclusion is assured.** After Pi has finished its critical section, it will set turn to j and flag[i] to false. Then Pj can enter its critical section, so **progress is assured.** Pj can enter its critical section after Pi has ran once so **bounded waiting is assured**.

3. 


First fit

P_1 : 212K → B
P_2 : 417K → E
P_3 : 112K → B
P_4 : 412K → X cannot allocate

Final: P_4 cannot be allocated it will have to wait.

B = 500 − 212K = 288K
E = 600K − 417K = 183K
B = 288K − 112K = 176K

Best fit – Allocate the smallest hole that is big enough

P_1: 212K → D	D = 300K − 212K = 88K
P_2: 417K → B	B = 500K − 417K = 83K
P_3: 112K → C	C = 200K − 112K = 88K
P_4: 412K → E	E = 600K − 412K = 188K

Final: A = 100K, B = 83K, C = 88K, E = 188K, D = 88K

Worst fit – Allocate the largest hole that is big enough

P_1: 212K → E	E = 600K − 212K = 388K
P_2: 417K → B	B = 500K − 417K = 83K
P_3: 112K → E	E = 388K − 112K = 276K
P_4: 412K → $\times$ cannot be allocated anywhere at this moment.	

Conclusion: The algorithm that makes the most efficient use of memory in this case is the best-fit algorithm because it is able to allocate memory for all 4 processes at the same time.

4.

a. 2*150ns = **300ns** because the paging system requires two memory accesses one for the page table and one for the data/instruction.

b. Effective Memory Access Time = $s_a + (2\text{-}h)*m_a$ = 20ns + (2-0.8)*150ns = 200ns

5.

5.

a) FIFO

Page ref string	0	1	7	0	1	2	0	1	2	3	2	7	1	0	3	1
Memory 3 frames	0	1	7	7	7	2	0	1	1	3	2	7	1	0	3	3
		0	1	1	1	7	2	0	0	1	3	2	7	1	0	0
			0	0	0	1	7	2	2	0	1	3	2	7	1	1
Page fault?	V	V	V	X	X	V	V	V	X	V	V	V	V	V	V	X

Page Ref String	0	3
Memory 3 frames	3	3
	0	0
	1	1
Page fault?	X	X

$\boxed{\text{Total \# of page faults: 12 page faults for FIFO}}$

5 b)

Optimal - replace page that will not be used for longest time.

Page ref string	0	1	7	0	1	2	0	1	2	3	2	7	1	0
Memory	0	1	7	7	7	2	2	2	2	2	2	7	7	0
		0	1	1	1	1	1	1	1	1	1	1	1	1
			0	0	0	0	0	0	0	3	3	3	3	3
Page fault?	✓	✓	✓	X	X	✓ ~~X~~	X	X	X	✓	X	✓	X	✓

Page ref string	3	1	0	3
Memory	0	0	0	0
	1	1	1	1
	3	3	3	3
Page fault	X	X	X	X

Total # of page faults: 7 for optimal algo.

5 c)

LRU ⇓ Total # of pg faults: 9

Page ref number	0	1	7	0	1	2	0	1	2
Memory	0	1	7	~~7~~ 0	1	~~1~~ 2	0	1	2
		0	1	7	0	~~0~~ 1	2	0	1
			0	1	7	~~7~~ 0	1	2	0
Page fault?	✓	✓	✓	X	X	✓	X	X	X

Page ref num	3	2	7	1	0	3	1	0	3
Memory	3	2	7	1	0	3	1	0	3
	2	3	2	7	1	0	3	1	0
	1	1	3	2	7	1	0	3	1
Page fault?	✓	X	✓	✓	✓	✓	X	X	X

Conclusion for question 5: FIFO -> 12 page faults, Optimal -> 7 page faults, LRU -> 9 page faults.