Q#2

cache 8 bytes

$\uparrow$ LRU

set#		
0	0	0^M ~~0^H~~ 20^M
	1	~~124^M~~ 64^M
1	2	~~1^M~~ 41^M
	3	125^M
2	4	~~2^M~~ 10^M
	5	126^M 126^H
3	6	3^M
	7	127^M

	Set 0	Set 1
0	0^M ~~0^H~~ 20^M	~~124^M~~ 64^M
1	~~1^M~~ 41^M	125^M
2	~~2^M~~ 10^M	126^M , 126^H
3	3^M	127^M

Q #2 LFU

	counter	DATA	counter	DATA
0	~~0~~ 1	0^M 0^H	0	~~124^M~~, ~~64^M~~ 20^M
1	0	~~1^M~~ 41^M	0	125^M
2	0	~~2^M~~ 10^M	0	126^M, 126^H
3	0	3^M	0	127^M

Q #3

Addresses Sequence Generated by CPU

i, B[9], A[0], i, B[8], A[1], i, B[7], A[2], i, B[6], A[3],
i, B[5], A[4], i, B[4], A[5], i, B[3], A[6], i, B[2], A[7],
i, B[1], A[8], i, B[0], A[9]. $\Rightarrow$ so total request are 30

• above addresses shows both arrays are accessed sequentially in order. so it follows the concept of spatially locality

first let consider block size : 8 $\rightarrow$ LRU

0	i A[0,1,2,3,4,5,6,7]
1	B[9,8,7,6] ~~5,4,3,2,1~~]. i

when consider block size - 8 - it creates problem, as 3 variable values are ~~accessed~~ accessed simultaneously, all of 3 resides in different block. only two can be stored at a times in caches. so 8 is not suitable

Total Miss = 6
Total Hit = $\frac{24}{~~30~~}$
Hit rate $\frac{24}{30}$

block size = 4

0	$[i, 30, 31, sum[9]]^M, i^H, i^H, i^H, i^H, i^H, i^H, i^H, i^H, i^H$
1	~~$B[9,8,7,6]^M, B(8)^H, B(7)^H, B(6)^H$~~ $B[5,4,3,2]^M, B(4)^H, B(3)^H, B(2)^H$
2	~~$A[0,1,2,3]^M, A(1)^H, A(2)^H, A(3)^H$~~, $A[4,5,6,7]^M, A[5]^H, A(6)^H, A[7]^H$
3	$[A(8), A(9), B(0), B(1)]^M, A(8)^H, B(9)^H, B(1)^H$

Block size = 2

0	$[1,31]^M, 1^H, 1^H, 1^H, 1^H, 1^H, 1^H, 1^H, 1^H, 1^H,$	= $\boxed{9}$
1	~~$B[8,9]^M, B[8]^H$~~ $A[6,7]^M, A[7]^H$	2
2	~~$A[0,1]^M, A[1]^H$~~ $B[0,1]^M, B[0]^H$	2
3	$B[6,7]^M, B[6]^H, A[8,9]^M, A[9]^H$	2
4	$A[2,3]^M, A[3]^H$	1
5	$B[4,5]^M \quad B[4]^H$	1
6	$A[4,5]^M \quad A[5]^H$	1
7	$B[2,3]^M, B[2]^H$	1
		19

Total hit = 19

Total Miss = 11

$$\text{Hit} = \frac{19}{30} < \frac{24}{30}$$

So block size = 4 is feasible

Q #4

a)

WAW	RAW	WAR
(1,5)	(1,6)	(5,6)
(2,4)	(2,3)	(3,4)
	(5,6)	(2,3)
		(1,3)

b). i)

Raw	Structural
(1,6)	(2,6)
(2,3)	(1, 4)
(5,6)	(3,5)

ii)

fetch	Execute *	Execute +	Execute And	write	cycles
1, 2					1
3, 4	2	1			2
3 4	2	1			3
3 4.	2			1	4
5, 6		4	3	2	5
6		4	5	3	6
				4, 5	7
	6				8
	6				9
	6			6.	10