1. a) For nested-loop, the biggest advantage is that it reguires no indicies and can be used with any kind of join condition. However, it is expensive as it will examine every pair of tuples in r & s, and for the worst case where memory is only enough for holdig one block of each relation, cost will be $(n_r * b_s + b_r) + (n_r + b_r)$

For merge-join, it is efficient when dealing with the situation where both relation are already sorted on the join key. and for seguential disk access pattern, it can be daster and reduce seek time. However, if the input relations are not sorted, the cost of sortig can be significant especially for large relations where sorting might involve additional disk I/O

b) For r

① run 1

load 3 entries in 3 blocks in buffer

(77, c), (77, a), (77, g)

Sort based on B

(77, a), (77, c), (77, g) ⇒ run 1

② run 2

load 3 entries in 3 blocks in buffer

(31, h), (40, f), (3, d)

sort based on B

(3, d), (40, f), (31, h) ⇒ run 2

③ merge $N=2 < M=3$, two blocks to buffer inputs, 1 blocks to buffer output

input	input	output
(77, a)	(3, d)	(77, a)
(77, c)	(3, d)	(77, c)
(77, g)	(3, d)	(3, d)
(77, g)	(40, f)	(40, f)
(77, g)	(31, h)	(77, g)
		(31, h)

For S

For each run: pick 3 entries into 3 blocks of buffer, sort on B

① run 1

(a, 2), (d, 1), (g, 5)

② run 2

(b, 6), (e, 10), (f, 2)

③ run 3

(c, 9), (h, 7)

④ Merge $N = M = 3$

1. merge $M-1 = 2$ runs

take run1 & run2

input	input	output
(a,2)	(b,6)	(a,2)
(d,1)	(b,6)	(b,6)
(d,1)	(e,10)	(d,1)
(g,5)	(e,10)	(e,10)
(g,5)	(f,2)	(f,2)
		(g,5)
		run 4

⑤ merge $N=2 < M$

take run4 & run3

input	input	output
(a,2)	(c,9)	(a,2)
(b,6)	(c,9)	(b,6)
(d.1)	(c,9)	(c,9)
(d,1)	(h,7)	(d,1)
(e,10)	(h,7)	(e,10)
(f,2)	(h,7)	(f,2)
(g,5)	(h,7)	(g,5)
		(h,7)

∴ r

A	B
77	a
77	c
3	d
40	f
77	g
31	h

s

B	C
a	2
b	6
c	9
d	1
e	10
f	2
g	5
h	7

For merge-join

1st I/O: load (a,2)

⇒ 3rd I/O: write (a, 2, 77)

2nd I/O: load (77,a)

4th I/O: load (b,6)

5th I/O: load (77,c)

⇒ 7th I/O: write (c,9,77)

6th I/O: load (c,9)

8th I/O: load (d,1)

⇒ 10th I/O: write (d, 1, 3)

9th I/O: load (3,d)

11th I/O: load (e,10)

12th I/O: load (40,f)

⇒ 14th I/O: write (f, 2, 40)

13th I/O: load (f,2)

15th I/O: load (g,5)

⇒ 17th I/O write (g, 5, 77)

16th I/O: load (77,g)

18th I/O: load (h, 7) ⇒ 20th I/O write (h, 7, 31)

19th I/O: load (31, h)

2、 M = 3 , each 2 records

load 6 records on 3 blocks

10, 11, 1, 5, 90, 1

sort

1, 1, 5, 10, 11. 90 ⇒ run 1

load 2 records on 1 block

2, 101 ⇒ run 2

a) 2 runs

run 1: (1, 1, 5, 10, 11, 90)

run 2: (2, 101)

b) ∵ N = 2 , M = 3 $N < M$

∴ 3 blocks, 1 for run1 input, 1 for run2 input, 1 for output

input	input	output
(1, 1) ← load	(2, 101) ← load	(1, 1)
(5, 10) ← load	(2, 101)	(2, 5)
10	101	10
(11, 90) ← load	101	(11, 90)

101   101

output: 1, 1, 2, 5, 10, 11, 90, 101

usig N-way merge

C) for creating runs

load 3 + 1 = 4 blocks from disk to memory

for merging

run 2 fit in 1 block ⇒ 1 block transfered

run 1 loads 3 blocks ⇒ 3 blocks transferred

∴ 3+1=4

∴ total = merge + create = 4+4= 8 blocks