The relations will be joined on A.

a) Assume that there are 150 buffers (pages) available in the main memory to execute the join. (1 point)

B(R) = 80,000; B(S) = 20,000

Block nested loop cost would be: B(R) + B(S)[B(R) / (M – 2)]

That's 80,000 + 20,000(80,000 / 148)

80,000 + 20,000(540)

10,880,000

Sort Merge: 5B(R) + 5B(S)

500,000

Optimized Sort Merge: 3B(R) + B(S)

60,000 + 240,000 = 300,000

B(R) + B(S) < M^2

The memory requirements is not met. 100,000 > 22,500

Hash Join: 3B(R) + B(S)

260,000

Memory Requirements: B(R) < M^2, which is B(R) < 22,500

The memory requirement is not met if we use R(A, B) as the R, but if we switch the order, we'll get an I/O complexity of 140,000. Since B(R) = 20,000, where S(A,C) is R, the memory requirement is met. This is the best join algorithm for this case.

b) Assume that there are 100 buffers (pages) available in the main memory to execute the join. (1 point)

Hash join will no longer work since B(R) and B(S) > 100^2. Our guaranteed option is block nested loop.

Block nested loop cost would be: B(R) + B(S)[B(R) / (M – 2)]

That's 80,000 + 20,000(80,000 / 98)

80,000 + 20,000(816)

16,400,000

We could however use Sort Merge: 5B(R) + 5B(S)

500,000

The memory requirement is met, M = 100 > 2

This is our best option since it has the lowest I/O complexity, and the memory requirement is met.

c) Assume that there are 320 buffers (pages) available in the main memory to execute the join. (1 point)

We may be able to use Optimized Sort Merge with the greater buffer size.

B(R) + B(S) is 100,000 and 320^2 is 102,400. Since 100,000 < 102,400 the memory requirement for optimized sort merge.

By using S(A, C) as R in the algorithm we can get an I/O complexity of 140,000. Hash Join would have a similar number of I/O accesses, but we should use Sort Merge because we get the added benefit of a sorted output.