Group members:
Xiaoru Chen, chenxia2@oregonstate.edu, chenxia2
Yu-Chien Lin, linyuchi@oregonstate.edu, linyuchi

1.
a)
The fastest join algorithm should be improving nested-loop join.
The cost of this algorithm is: $B(R) + \left[\frac{B(R)}{M-2}\right] \times B(S)$

- Almost $\frac{B(R)B(S)}{M} = \frac{80000 \times 20000}{10} = 160000000$

b)
The fastest join algorithm should be hash join.
The cost of this algorithm is: $3B(R) + 3B(S) = 3 \times 80000 + 3 \times 20000 = 300000$

c)
The fastest join algorithm should be hash join.
The cost of this algorithm is: $3B(R) + 3B(S) = 3 \times 80000 + 3 \times 20000 = 300000$

2.
a)
Cost of Nested-loop join algorithm:
$B(R) + B(R) \times B(S)$

Cost of Improving Nested-loop join algorithm:
$\big(B(R)B(S)\big)/M$

Cost of Sort merge join algorithm:
$Sorting + 2B(R) + 2B(S)$

Cost of Optimized sort-merge join algorithm:
$Sorting + 2B(R) + 2B(S) - Constant$

Cost of Hash join algorithm:
$3B(R) + 3B(S)$

Cost of Index-based (Zig-Zag) join:
$B(R) + B(S)$

We only need to compare the cost of Improving Nested-loop join algorithm and the cost of Index-based (Zig-Zag) join, because the cost of other algorithms are obvious bigger than these two.
Because the entire relation R(A,B) fits in the available main memory but S(A, C) is too large to fit, we got:

$$B(R) < M < B(S)$$

We
Then, $\frac{B(R)\times B(S)}{M} = \frac{B(R)\times B(S)}{B(R)+constant} < B(S)$

Therefore, in this case, $\frac{(B(R)B(S))}{M} < B(R) + B(S)$

Therefore, the fastest join algorithm, i.e., an algorithm with the lowest number of I/O access is Improving Nested-loop join algorithm.

If there is a clustered index on attribute A of relation S, this will not change my answer.

Clustered index on R: $B(S) + T(S)B(R)/V(R,A)$
It is still larger than $B(S)$
Therefore, this will not change my answer.

b)

We are looking for an algorithm that has the lowest time-complexity.

So sort-based join algorithm and hash-based join algorithm are the potential choices.
They both can handle large amount of inputs.

However, if the relations sizes are different, hash wins.

In this case, we have two large relations with the same size of 1 million tuples.

Therefore, hash join is a better choice.