Sample Solutions

Question 1

Number of blocks of R = 500,000/500 = 1000 = |R|
Number of blocks of S = 100,000/100 = 1000 = |S|
Time for one random I/O = Tr
Time for one sequential I/O = Ts

We read R one block at a time.
Cost = |R| random I/Os (because read head needs to access S relation when checking a tuple from R)

For every tuple of R, we retrieve all the tuples of S that have a matching B value. Assuming that all R.B values are also found in column S.B of relation S, there will be (number of S tuples/number of distinct B values in S) matches for each tuple of R. Since the index on S.B is not clustered, there will be one IO for each tuple.

Cost = 500,000 * (number of S tuples/number of distinct B values) random I/Os

IOplan1 = |R| + number of R tuples * (number of S tuples/number of distinct B values) = 1000 + 500,000*100,000/40,000 = 1,251,000 I/Os.

All these I/Os are random I/Os. Thus, Timeplan1 = 1,251,000*Tr.

Question 2

We read R one block at a time, so |R| I/Os, so 1000 random I/Os

For each tuple of R, we retrieve all the tuples of S that have a matching C value. Assuming that all the R.C values are also found in the S.C column of relation S, there will be (number of S tuples)/(number of distinct C values in S) matches for each tuple of R. However, since there is clustering on attribute C, this will take only (|S|/number of distinct C values) IO's. So,

IOplan2 = |R| + number of R tuples * 1 will be random whereas the remaining (|S|/number of distinct C values – 1) will be sequential. Hence, we have:

Timeplan2 = (|R| + number of R tuples * 1) Tr + (number of R tuples * [|S|/#distinct C values – 1]) * Ts
= 501,000 Tr + 12,000,000 Ts

Comparison:

In terms of IOs, clearly IOplan2 > IOplan2

But what is more important (most of the time) is access time. Now plan 1 is better than plan 2 if

Timeplan1 < Timeplan2
1,251,000 Tr < 501,000 Tr + 12,000,000 Ts

$750{,}000\ Tr < 12{,}000{,}000\ Ts$
$Tr < 16\ Ts$

Hence, as long as random access is no more than 16 times more time consuming than sequential IO, plan 1wins - otherwise plan 2 wins.

Question 3

R = 1000000/20 = 50000 pages
S = 2000000/40 = 50000 pages

a)
For Block-nested loops join, cost = $50000 + \lceil 50000/98 \rceil * 50000 = 50000 + 511 * 50000 = 25{,}600{,}000$
For Sort-merge join, sort R = sort S = $2*50000*(1 + \lceil \log_{99} \lceil 50000/100 \rceil \rceil) = 300{,}000$
So, cost of sort-merge join = 300,000 + 300,000 + 50,000 + 50,000 = 700,000
If tables are already sorted, then it costs only 100,000.

For hash join, cost = 5*(|R| + |S|) = 5*100,000 = 500,000:
Since $\sqrt{50k} > 100$, assuming uniform distribution of data, we need to recursively partitions the file.
So, partitioning phase involves: read R, write partitions of R; read partitions of R, write more partitions of R.
Total cost = 4*50000 = 200,000
Similar cost for S in partitioning = 200,000
Joining phase only reads partitions of R and the corresponding partitions of S. So, its one read of R and S. So, another 100,000.

b)
Essentially, for each R-tuple, it will probe the index for a match.
Cost to read R = 50000

In the worst case, R is not sorted and we are not able to buffer any record of S.
The probing cost = 1000000*(3+1) = 4,000,000. This would also be the result if the index is unclustered.
Even if we assume we can buffer the top 2 or 3 levels, we still need at least 1,000,000 I/Os.

The best case happens when R.a has only a single value. The probing cost = 3 + 1 = 4.

Question 4
Basic idea:
Generate sorted runs of R. Instead of merging the runs into a single sorted run, stop the merging when the number of runs is $< \lfloor (B-1)/2 \rfloor$ where B is the number of buffer pages.
Repeat the above step for S.
Allocate 1 buffer page for each run of R and S. Allocate 1 buffer page for join output. As tuples of the sorted R and S are produced, they can be checked whether the join predicate is satisfied.

Savings: Cost of reading and writing a single sorted run of R and S