Student Id: 1605029

Question 9-1

a. Partition vector, V = [2016, 2017, 2018, 2019, 2020]

Partitions:

P_1 = [2015]

P_2 = [2016]

⋮

P_6 = [2020, 2021]

b. We'll perform the join operation in the following way:

1. We'll repartition $r_1, r_2, \ldots, r_{10}$ into new partitions $r_1', r_2', \ldots, r_6'$. We'll do the same thing for s. Each new partition r_i and s_i will contain entries having yearAdmit in partition P_i.
2. We'll assign r_i' to node N_i. Again, same thing for s.
3. We'll perform $r_i' \bowtie s_i'$ at node N_i.

Question 9-2

a. We'll compute the join by performing $r_i \bowtie s_i$ at node N_i.

b. Since number of partitioning nodes is equal to number of query nodes and the partitioning attribute and the query attribute are same, repartitioning is redundant.

Question 10-1

a. There are 4 runs. If we take 5 as our memory size, we'll be able to keep an input buffer of size 4 (equal to the run count), and will be able to perform external sort in one pass.

b. If we take a memory size equal to the number of elements we need to sort, quicksort can be used.

c. If we increase memory size, number of merge pass will decrease and sort performance will increase consequently.

Question 10-2

a. Since the relation is range-partitioned on the attribute on which it is to be sorted, we can sort each partition separately and concatenate the result to get the full sorted relation.

b. We can do either of the two things below:

1. We can range-partition the relation on the sort attributes and sort each partition separately.
2. We can use a parallel version of the external merge sort algorithm.

Question 11-1

a. Here, the query is:

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4 = [(r_1 \bowtie r_2) \bowtie r_3] \bowtie r_4$$

$$= (temp_1 \bowtie r_3) \bowtie r_4$$

$$= temp_2 \bowtie r_4$$

The 30 nodes can be assigned to process the following things:

~~$N_1 = 10$~~:

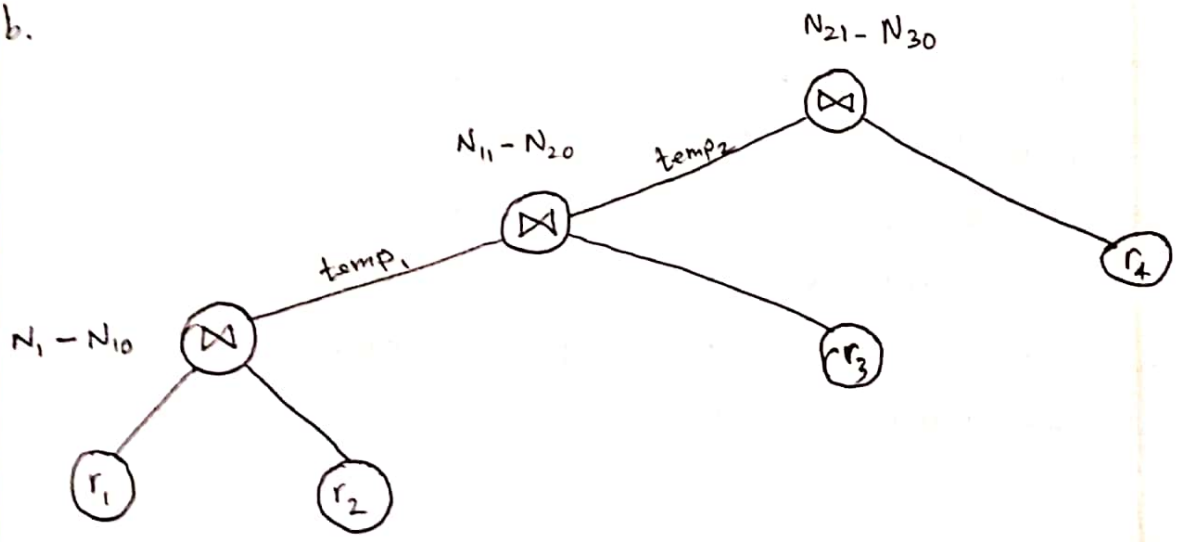
$N_1 - N_{10}$: $r_1 \bowtie r_2$

$N_{11} - N_{20}$: $temp_1 \bowtie r_3$

~~N_1~~ $N_{21} - N_{30}$: $temp_2 \bowtie r_4$

We'll repartition r_1 and r_2 from the 30 nodes to ~~N_1~~ $N_1, N_2, \ldots, N_{10}$. We'll perform similar partitioning on the pairs $(temp_1, r_3)$ and $(temp_2, r_4)$

b.



Question 11-2

a. The operations at each of the nodes are:

N_1: $temp_1 = S_1 \bowtie S_2$

N_2: $temp_2 = S_3 \bowtie S_4$

N_3: $temp_3 = S_5 \bowtie S_6$

} Independent parallelism

N_4: $temp_4 = temp_1 \bowtie temp_2$

N_5: $result = temp_3 \bowtie temp_4$

} Pipeline parallelism

When execution of both N_1 and N_2 ~~are~~ is finished, execution of N_4 will begin and this node will run in parallel with N_3. Execution of N_5 will begin only ~~after~~ when both N_3 and N_4 have finished executing.

b. Pipeline parallelism is faster than independent parallelism.

Execution takes place in stages in independent parallelism. At first, only the leaf nodes are allowed to run. When all of them are finished, the parent nodes can start running. This wastes a lot of CPU time since many parent nodes have to stay idle even when their children have finished executing.

Even though pipeline parallelism is slower than the combined one, it's still faster than the independent since a group of nodes don't block out other nodes here. As soon as the node in control finishes execution, the next node takes control.