Homework 4

# Question 1

## A

1. Start at the root node [43].
2. Check if 15 is less than the most minor key in the node, which is 6.
3. Since 15 is greater than 6, follow the pointer to the right child node.
4. Check if 15 is less than the most minor key in the node, 10.
5. Since 15 is greater than 10, follow the pointer to the right child node again.
6. Now, at the leaf node [19,20,30].
7. Scan the keys in this node and find that only 19 and 20 satisfy the query condition.
8. Follow the right sibling pointer to the next leaf node [43,44].
9. Scan the keys in this node and find that only 43 satisfy the query condition.
10. The keys that satisfy the query condition are 19, 20, and 43.

The total number of block I/O's needed for this process is three blocks to be read from the disk (the root node, the leaf node containing 19 and 20, and the leaf node containing 43).

# Question 2

## A

Block-based nested-loop joins with R as the outer relation:

For each block of R, we can bring it into the main memory and compare it with each block of S. Therefore, we need to perform 20,000 \* 50,000 = 1 billion blocks I/Os to complete the join. The output size is not explicitly given, but it will be, at most, the product of the number of tuples in R and S, assuming no duplicates.

## B

Block-based nested-loop join with S as the outer relation:

Similarly, for each block of S, we can bring it into the main memory and compare it with each block of R. Therefore, we need to perform 50,000 \* 20,000 = 1 billion blocks I/Os to complete the join. The output size is the same as in part (a).

## C

Sort-merge join:

To perform the sort-merge join, we first need to sort R and S on the join attribute (a) using external sorting. Since R has 20,000 blocks and S has 50,000 blocks, we can assume that each relationship will require two-way merging, which can be done in two passes. We can use 50 pages (half of the available memory) for each merge pass. The number of block I/Os for sorting each relation is given by:

R: ceil(log2(20,000)) \* (2 \* 50) = 1600

S: ceil(log2(50,000)) \* (2 \* 50) = 3200

Once the relations are sorted, we can merge them based on the join attribute (a). We can use 101 pages for the merge pass. The number of block I/Os for merging is given by:

merge: 2 \* (20,000 + 50,000) = 140,000

Therefore, the total number of the block I/Os needed is 1600 + 3200 + 140,000 = 145,800. The output size is, at most, the product of the number of tuples in R and S, assuming no duplicates.

## D

Partitioned-hash join:

To perform the partitioned-hash join, we first need to partition R and S into buckets based on the hash value of the join attribute (a). Since we have 101 pages available for partitioning, we can assume that each relation can be partitioned into 100 buckets using a single pass. The number of block I/Os for partitioning each relation is given by:

R: ceil(20,000 / 100) = 200

S: ceil(50,000 / 100) = 500

Once the relations are partitioned, we can join the buckets based on the join attribute (a) by bringing each bucket of R and S into memory and performing an in-memory hash join. Since we have 102 pages available in memory, we can assume that we can fit one bucket from each relation into memory at a time. Therefore, we must perform 100 \* (200 + 500) = 70,000 blocks I/Os to complete the join.

Therefore, the partitioned-hash join is the most efficient algorithm for block I/O, as it only requires 70,000 block I/Os compared to the 1 billion blocks I/Os needed for the nested-loop join algorithms and the 145,800 blocks I/Os required by the sort-merge join algorithm. The output size is, at most, the product of the number of tuples in R and S, assuming no duplicates.