CS623 Project - Theory

1.

Number of pages per pass
6 -1 = 5

log 5 (N) = log (N) / log (5)

log 5 (1000000) = log (1000000) / log (5)
log (1000000) / log (5) = 8.5840

9 Passes

2.

Starting with the root node, we can see that all keys in the root's left subtree (10) are smaller than 9*. As a result, we must traverse the appropriate subtree.
The first node in the right subtree is 20, which has no keys between 9* and 19*. However, because the root's second child (30) is greater than 19*, hence no need to traverse its subtree. The only node that contains keys between 9* and 19* is the left child of the root (10)

From the root to the left child: 1 parent-to-child pointer
From the left child to its right sibling (20): 1 sibling-to-sibling pointer
From 20 to its left child (11): 1 parent-to-child pointer
From 11 to its right sibling (12): 1 sibling-to-sibling pointer
From 12 to its right sibling (13): 1 sibling-to-sibling pointer

5 pointers

3.

25

When a key that triggers a split is inserted, the number of primary pages increases by one. At level 0, there are four primary pages, each with a capacity of three. As a result, without a split, the maximum number of keys that can be inserted is 12 (4 pages x 3 keys per page). To discover the bucket where 25 would hash to using h1(x) and h0(x), we need to locate the greatest key less than 25 that will trigger a split. The hash value of 25 using h1(x) is 001, and the hash value using h0(x) is 01. As a result, 25 would be assigned to the bucket that now holds 9, 25, and 5. If we try to insert a key into this bucket, it will

result in an error. Therefore, the largest key less than 25 that will cause a split is the largest key currently in the bucket, which is 25.

4.
level 1: Nodes with keys (1, 2), (3, 4), (5, 6), (7, 8), (9, 10), (11, 12), (13, 14), (15, 16), (17, 18), (19)
level 2: Nodes with keys (5, 9, 13), (17)
level 3: Root node with keys (9, 17)

15 nodes

5.
Plan I:

1. Join R and S on b: $R \bowtie_{b=b} (S)$
2. Select rows where c=3: $\sigma_{c=3}(R \bowtie_{b=b} (S))$
3. Project only column a: $\pi_a(\sigma_{c=3}(R \bowtie_{b=b} (S)))$

Plan II:

1. Select rows from S where c=3: $\sigma_{c=3}(S)$
2. Join R with the result of step 1 on b: $R \bowtie_{b=b} \sigma_{c=3}(S)$
3. Project only column a: $\pi_a(R \bowtie_{b=b} \sigma_{c=3}(S))$

Plan II is more efficient because it selects rows from S based on the condition c=3 before performing the join operation; this reduces the size of the intermediate result that needs to be joined with R. In Plan I, the join is performed first, which could result in a larger intermediate result before filtering out rows where $c \neq 3$.

6.
True

The vectorized processing model allows for more efficient processing by reducing the overhead of function calls and control flow. When receiving input from multiple children, each operator requires multi-threaded execution to generate the next output tuples from each child.

7.
Partitioning: By dividing the input data into smaller parts, you can increase performance by lowering memory utilization and increasing cache locality. The join key can be used to partition the input data, and each partition can be processed independently.

Memory Management: The hash join approach necessitates a substantial amount of memory in order to construct the hash tables. Memory management can increase performance by minimizing the requirement for disk I/O. A multi-pass method that spills intermediate results to disk when memory use surpasses a particular threshold is one approach to accomplish this.

Parallelism: Because each division can be handled individually, the hash join algorithm is intrinsically parallelizable. The approach can take use of modern computing power by employing many threads or processes to process the partitions in parallel.

Caching: By reducing the need to recompute intermediate results, you can enhance performance. For example, when connecting multiple tables, the hash table created from one input table can be cached and reused.

8.
3720 page I/Os.

The cost of the query plan can be determined by adding the number of page I/Os necessary to perform each operator.

i). File scan on schools - Schools has 10 pages, the cost is 10 page I/Os.
ii). Selection - Since there are 100 schools and we assume a uniform distribution of srank values, about 10 schools will have srank < 10. Therefore, the cost of this operator is about 10 page I/Os.
iii). Sort-merge join - This join will use a merge join algorithm. Since there are 10 qualifying schools and 200 applicants who want to major in CSE, we will need to read each of the 10 pages of Schools once and each of the 200 pages of Applicants once. Therefore, the cost of this operator is about 210 page I/Os.
iv). Index nested loop join - Since there are 3000 applicants who want to major in CSE and there is an index on Major.id, we will need to read each of the 3000 index pages and

each of the 100 pages of Applicants once. Therefore, the cost of this operator is about 3100 page I/Os.
v). Selection - Since there are 3000 applicants who want to major in CSE and we assume a uniform distribution of majors, about 300 applicants will want to major in CSE. Therefore, the cost of this operator is about 300 page I/Os.
vi). Projection - Only the name attribute is being projected, read each of the 300 pages of the result of the previous operator once. Therefore, the cost of this operator is 300 page I/Os.

9.
a) External Hashing - External hashing is a method for executing hash-based operations on data that is too large to fit in main memory. It involves dividing the data into memory-sized blocks and then applying the hash function to each one separately. This decreases the quantity of data that must be processed at once while also ensuring that the hash function is applied consistently.

b)
R: 2,400 pages * 1 block per page = 2,400 blocks
S: 1,200 pages * 10 blocks per page = 12,000 blocks

2,400 * 12,000 = 28,800,000
2,400 blocks * 1 write per block = 2,400
28,800,000 + 2,400 + 2,400 = 28,804,800

cost = 28,804,800

10.
number of leaf nodes = number of total nodes - number of internal nodes
(2n + 1) - (2n)

n + 1 leaf nodes

11.
a.) Insert 12, Insert 13