Homework 4: Indexing and Query Execution

1.

1. Process:

node with 43 → left pointer → left internal node with 6, 10, 17 and 19 → 3rd pointer → 3rd leaf node with 10 and 13 → last pointer → 4th leaf node with 17 and 18 → last pointer → 5th leaf node with 19, 20 and 30 → last pointer → 6th leaf node with 43 and 44 → last pointer → 7th leaf node with 56 and 58.

Cost:

Seven blocks I/O’s are needed for the process.

1. Inserting 31 and 32:

Diagram

Description automatically generated

1. Deleting 18:

Diagram

Description automatically generated

2.

1. Block-based nested-loop join with R as the outer relation

* 1000 blocks of R / 100 blocks of Memory = 10 chunks, 100 blocks of R/chunk
* For each chunk of R
* Input: 100 blocks of R all loaded to Memory + 1 block of S loaded to Memory
* Output: 1 block of joined tuple
* Loop until all blocks of S are loaded to Memory
* Loop until all chunks of R are loaded to Memory

Cost

* 1 pass for R
* 1 pass for S in each chunk, so a total of 10 passes for S
* B(R) + B(R)/(M-2) \* B(S) = 1000 + 1000/(102-2) \* 500 = 6000 blocks

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

* 500 blocks of S / 100 blocks of Memory = 5 chunks, 100 blocks of S/chunk
* For each chunk of S
* Input: 100 blocks of S all loaded to Memory + 1 block of R loaded to Memory
* Output: 1 block of joined tuple
* Loop until all blocks of R are loaded to Memory
* Loop until all chunks of S are loaded to Memory

Cost

* 1 pass for S
* 1 pass for R in each chunk, so a total of 5 passes for R
* Total: B(S) + B(S)/(M-2) \* B(R) = 500 + 500/(102-2) \* 1000 = 5500 blocks

1. Sort-merge join

* Sort
  + 1000 blocks of R / 100 blocks of Memory = 10 runs, 100 blocks of R/run
    - For each run of R
      * + Input: 100 blocks of R all loaded to memory
        + Output: 100 sorted blocks of R all written back
    - Repeat until all runs of R are sorted
  + 500 blocks of S / 100 blocks of Memory = 5 runs, 100 blocks of S/run
    - For each run of S
      * + Input: 100 blocks of S all loaded to memory
        + Output: 100 sorted blocks of S all written back
    - Repeat until all runs of S are sorted
* Merge
  + Input: 1st block from each of the 10 sorted runs of R and 5 sorted runs of S
  + Output: 1 block of merged result
  + Repeat until all runs are loaded to Memory

Cost

* 1 pass for R & S in sorting, but sorting requires writing the results
* 1 pass for R & S in merging
* Total: 2B(R) + 2B(S) + B(R) + B(S) = 3B(R) + 3B(S) = 3 \* 1000 + 3 \* 500 = 4500 blocks

1. Partitioned-hash join

* Hash
  + 1000 blocks of R / 100 buckets = 10 blocks of R/bucket
    - Input: 1 block of R
    - Output: 100 blocks that corresponds to 100 buckets for R
    - Repeat until all blocks of R are hashed into corresponding buckets
  + 500 blocks of S / 100 buckets = 5 blocks of S/bucket
    - Input: 1 block of S
    - Output: 100 blocks that corresponds to 100 buckets for S
    - Repeat until all blocks of S are hashed into corresponding buckets
* Join
  + Input: a pair of corresponding R & S buckets
  + Output: a block of joined result
  + Repeat until all buckets are loaded to Memory

Cost

* 1 pass for R & S in hashing, but hashing requires writing the results
* 1 pass for R & S in joining
* Total: 2B(R) + 2B(S) + B(R) + B(S) = 3B(R) + 3B(S) = 3 \* 1000 + 3 \* 500 = 4500 blocks

The sort-merge join and partitioned-hash join algorithms seem to be the most efficient

because they need the smallest number of block I/O’s, which is 4500 blocks.