# DSCI 551 – HW4

## (Indexing and Query Execution)

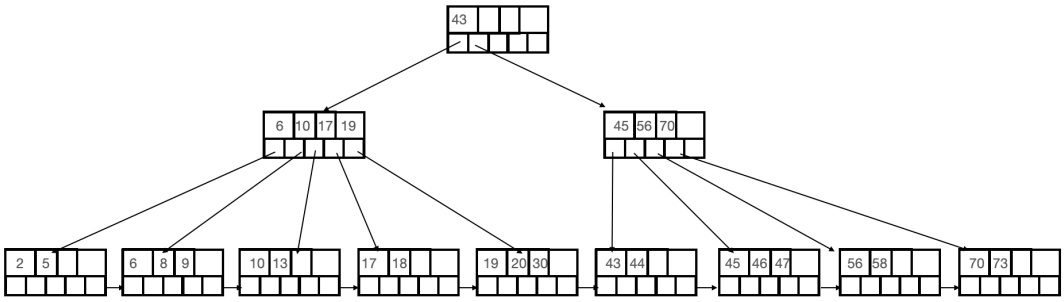
## (Fall 2021)

100 points, Due 11/14, Sunday

1. [40 points] Consider the following B+tree for the search key "age. Suppose the degree d of the tree = 2, that is, each node (except for root) must have at least two keys and at most 4 keys. **Note that sibling nodes are nodes with the same parent.**

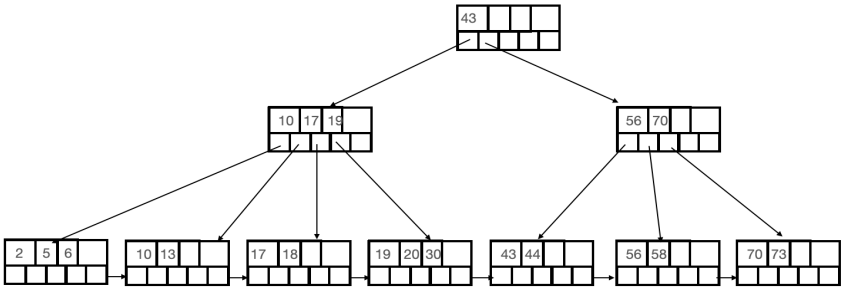
    a. [10 points] Describe the process of finding keys for the query condition "age > 20 and age <= 50". How many blocks I/O's are needed for the process?
    1.Evaluate age > 20, Read the root block and find out 20 is less than 43, go to the left child of 43 Cost 1 IO to read root node
    2. Read the read the left child of root node and find out 20 is greater than 19, go to the right child of 19 Cost 1 IO to read the intermediate node.
    3. Find 30 in the leaf node and traverse along the linked list until the value is greater than 50. Cost 1 IO to read the first leaf node and 2 IO to read the rest of leaf nodes.
    4. Cost 5 block IOs in total.

b. [15 points] Draw the B+-tree after inserting 45, 46, 47 into the tree. Only need to show




the final tree after the insertions.

c. [15 points] Draw the tree after deleting 8 and 9 from the original tree.



2. [60 points] Consider natural-joining tables R(a, b) and S(a,c). Suppose we have the following scenario.

   i. R is a clustered relation with 5,000 blocks.

   ii. S is a clustered relation with 10,000 blocks.

   iii. 102 pages available in main memory for the join.

   iv. Assume the output of join is given to the next operator in the query execution plan (instead of writing to the disk) and thus the cost of writing the output is ignored.

   Describe the steps (including input, output, and their sizes at each step, e.g., **sizes of runs or buckets**) for each of the following join algorithms. What is the total number of block I/O's needed for each algorithm? Which algorithm is most efficient?

   a. [10 points] (Block-based) nested-loop join with R as the outer relation.

      Nested-loop join with R as outer relation
      For each br of every 100 blocks, read each block bs of S to check tuple r in br and tuple s in bs, if r and s join, output (r,s)

      Total block I/O: B(R) + B(S)B(R)/100 = 5000 + 5000*10000/100 = 505,000 blocks

   b. [10 points] (Block-based) nested-loop join with S as the outer relation.

      Nested-loop join with S as outer relation
      For each bs of every 100 blocks, read each block br of R to check tuple s in bs and tuple r in br, if s and r join, output (s,r)
      Total block I/O: B(S) + B(R)B(S)/100 = 10000 + 5000*10000/100 = 510,000 blocks

   c. [20 points] Sort-merge join (assume only 100 pages are used for sorting and 101 pages for merging). Note that if join can not be done by using only a single merging pass, runs from one or both relations need to be further merged, in order to reduce the number of runs. Select the relation with a larger number of runs for further merging first if both have too many runs.

      Steps:

      1. Split R into 50 parts with size of 100 pages each. Load and sort each part in memory and write back to disk.

         Input: R, size: B(R)

Output: sorted R, size: B(R)

2. Split S into 100 parts with size of 100 pages each. Load and sort each run in memory and write back to disk.

Input: S, size: B(S)

Output: sorted S, size: B(S)

3. Total number of runs is 50+100=150 which is still larger than the available pages in memory (101 pages). -> Further merge the runs for table S -> Read the 100 sorted runs produced in step 1, merge it into 1 run and write back to disk.

Input: 100 runs of sorted S, size: B(S)

Output: 1 merged run of S (completed sorted), size: B(S)

4. Read both R and S in sorted order into memory. Join the tuples r and s on their join attributes (a), then output the join results.

Input: R, S, size: B(R)+B(S)

Output: joined tuples (r,s)

Total Block I/O needed for this algorithm:

(2*5,000) + 2*(2*10,000) + (5,000+10,000) = 65,000

d. [20 points] Partitioned-hash join (assume 101 pages used in partitioning of relations and no hash table is used to lookup in joining tuples).

Steps:

1. Hash S into 100 buckets on the join attribute (a), with 100 blocks per bucket. Send all buckets to disk.

Input: S, size: B(S)

Output: hashed S, size: B(S)

2. Hash R into 100 buckets on the join attribute (a), with 50 blocks per bucket. Send all buckets to disk

Input: R, size: B(R)

Output: hashed R, size: B(R)

3. Load all blocks from a bucket of S, S_i, into memory. Then load a matching bucket R_i one block at a time, and output the joined tuples where the join attribute matches

Input: R, S, size: B(R) + B(S)

Output: joined tuples (r, s)

Total Block I/O needed for this algorithm:

(2*5,000) + (2*10,000) + (5,000+10,000) = 45,000

***Most efficient algorithm**