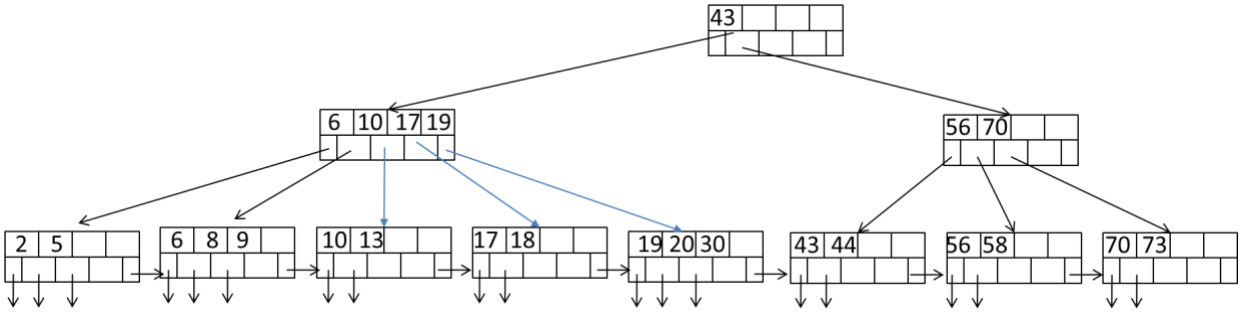
DSCI 551 – HW4

(Indexing and Query Execution)

(Spring 2022)

100 points, Due 4/11, Monday

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.**



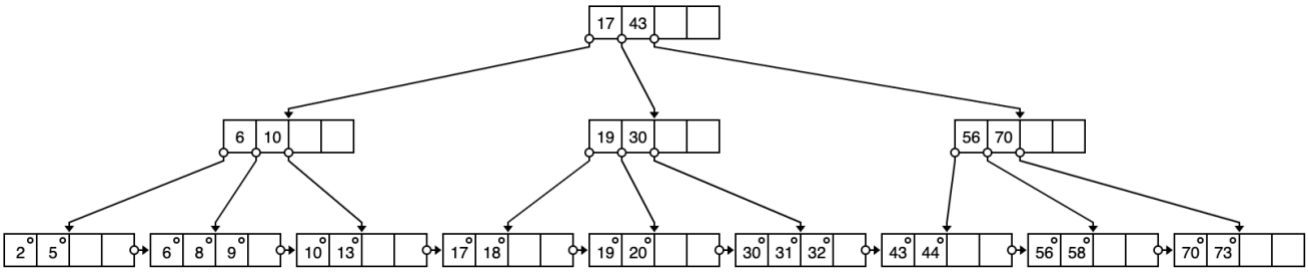
   a. [10 points] Describe the process of finding keys for the query condition "age >= 10 and age <= 50". How many blocks I/O's are needed for the process?

   First read the root node, find the first pointer
   Read intermediate node, use the third pointer redirect to 3rd leaf node
   Read the leaf node, find the starting node value 10, that block is starting block

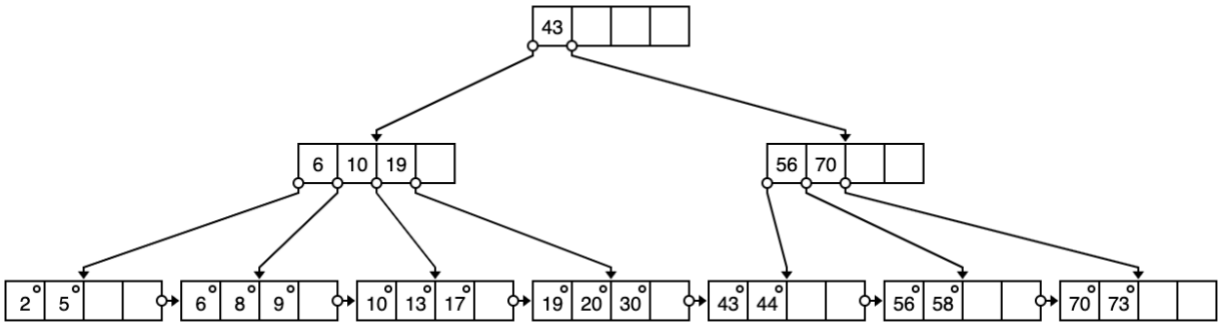
   Then sequential through leaf node following pointer, until reached first node at second to last leaf node (56), which is larger than 50, so we know to stop and not read this node.

   Total blocks: 7 (root + one intermediate node + 5 leaf node (3rd, 4th, 5th 6th, 7th))

   b. [15 points] Draw the B+-tree after inserting 31 and 32 into the tree. Only need to show the final tree after the insertions.



c. [15 points] Draw the tree after deleting 18 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 1000 blocks.
   ii. S is a clustered relation with 500 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.

      One pass for R , 10 passes for S (each pass reload 100 blocks)
      1000 blocks + 1000/(102-2) * 500 = 1000 + 10*500 = 6000

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

      One pass for S, 5 pass for R (each pass reload 100 blocks)
      500 blocks + 500/(102-2) * 1000 = 500 + 5*1000 = 5500

   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.

      Pass 1:

Read R, sort R using 10 runs, 100 blocks/run

Then send back to disk

(10 * 100) * 2 = 2000

Read S , sort S using 5 runs, 100 blocks/run

Then send back to disk

(5 * 100)* 2 = 1000

Pass 2 (merge): B(R) + B(S) = 1000 + 500 = 1500

Total cost : 3 * 1000 + 3 * 500 = 3000 + 1500 = 4500

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).

Step 1: hash R into 100 buckets

1000/100= 10 blocks/bucket (Ri)

Step 2: hash S into 100 buckets

500 / 100 = 5 blocks/bucket (Si)

Step 3 : Join every pair of corresponding buckets

1000 + 500 = 1500

Total cost : 1000 * 2 + 500 * 2 + 1500 = 4500

Ans: According to the total number of block I/O's needed for each algorithm, both Sort-merge join and Partitioned-hash join are most efficient.