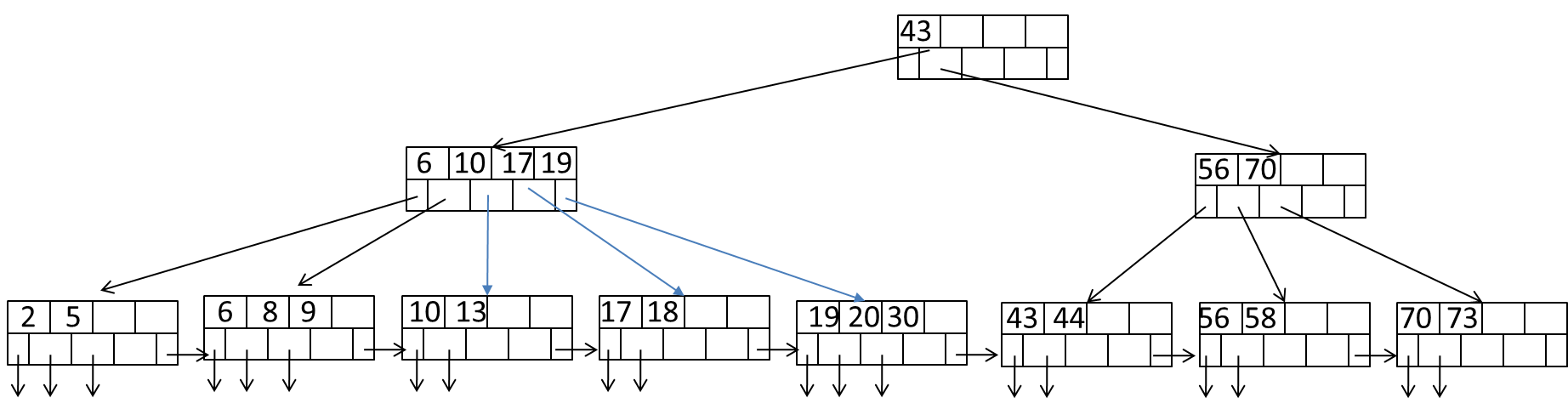
# DSCI 551 – HW4

# (Indexing and Query Execution)

# (Fall 2020)

## 100 points, Due 4/18, Sunday

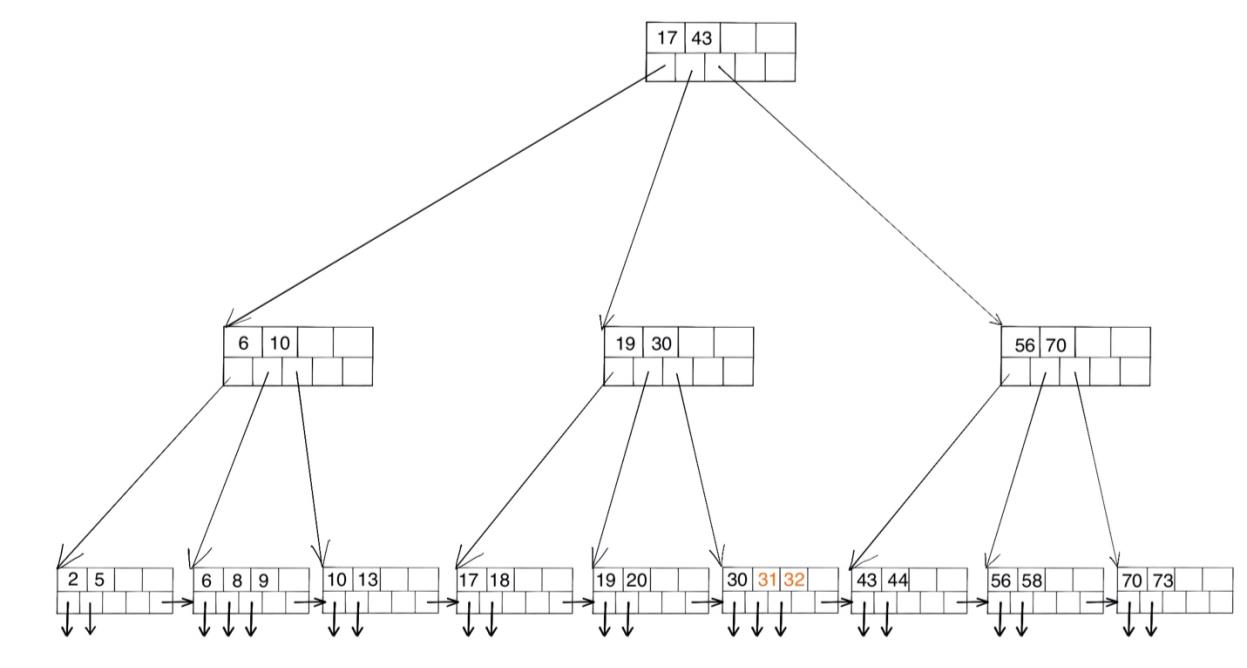
1. [30 points] Consider the following B+tree for the search key “age. Suppose the degree d of the tree = 2, that is, each node (except for root) must have at least two keys and at most 4 keys.



1. Describe the process of finding keys for the query condition “age >= 10 and age <= 20”. How many blocks I/O’s are needed for the process?
2. Read block (43)
3. Read block (6, 10, 17,19).
4. Read block (10,13), block (17,18), block (19, 20, 30). Since 30 > 20, stop.

Total I/O’s: 5

1. Draw the B+tree after inserting 31 and 32 into the tree. Only need to show the final tree after the two insertions.



1. Draw the tree after deleting 43 from the original tree.

Diagram

Description automatically generated

1. [40 points] Consider natural-joining tables R(a, b) and S(a,c). Suppose we have the following scenario.
   * 1. R is a clustered relation with 2,000 blocks and 10,000 tuples
     2. S is a clustered relation with 50,000 blocks and 100,000 tuples
     3. S has a clustered index on the join attribute a
     4. V(S, a) = 5 (recall that V(S, a) is the number of distinct values of a in S
     5. 102 pages available in main memory for the join.
     6. 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) 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?

* 1. Nested-loop join with R as the outer relation
  2. Sort-merge join (assume only 100 pages 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.
  3. Partitioned-hash join (assume 101 pages used in partitioning of relations and no hash table used for lookup in joining buckets)
  4. Index join (ignore the cost of index lookup)

3. each 10 points. Description 8 pts, calculation 2 pts. Give partial credit for wrong answers.

1. Nested loop, R outer:
   1. 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)

Input: S, 100 br

size: B(S)\*B(R)/100

Output: (r,s)

Total block I/O: B(R) + B(S)B(R)/100 = 2000+2000\*50000/100 = 1002000 blocks

1. Sort-merge
   1. Sort R by the join attribute. R can be sorted by 1 pass and sorted into 20 runs.

Input: R, size B(R)

Output: sorted R, B(R)

* 1. Sort S by the join attribute. S can be sorted by 1 pass and sorted into 500 runs and in pass 2 they are sorted to 5 runs.

Input: S, size 2B(S)

Output: sorted S, 2B(S)

* 1. Use 20 blocks of memory to put first block of 20 runs of sorted R and use one block for read one block of S each time. Check if r and s can join. Read more bs of S and br of R

Input: sorted R, B(R); sorted S, B(S)

Output: (r,s)

Total I/O: 3B(R) + 5B(S) = 3\*2000+5\*50000 = 256000 blocks

1. Partitioned-hash join
   1. Hash S into 100 buckets, send all buckets to disk (2000 < 100^2)

Input: S, B(S)

Output: hashed S, B(S)

* 1. Hash R into 100 buckets in 1 pass, send all buckets to disk. Hash R into 5 buckets in 2nd pass.

Input: R, 2B(R)

Output: hashed R, 2B(R)

* 1. Read in a partition of S, say Si. Load matching Ri one block at a time, output joining tuples

Input: R, B(R); S, B(S)

Output: (r,s)

Total I/O: 3B(R) + 5B(S) = 256000 blocks

1. Index join
   1. Iterate over R, for each tuple, fetch corresponding tuple(s) from S using index of S

Total I/O: B(R) + T(R) B(S)/V(S,a) = 100002000 blocks

From results above, we can see both sort-merge and partitioned-hash join are the most efficient algorithm.