HW4

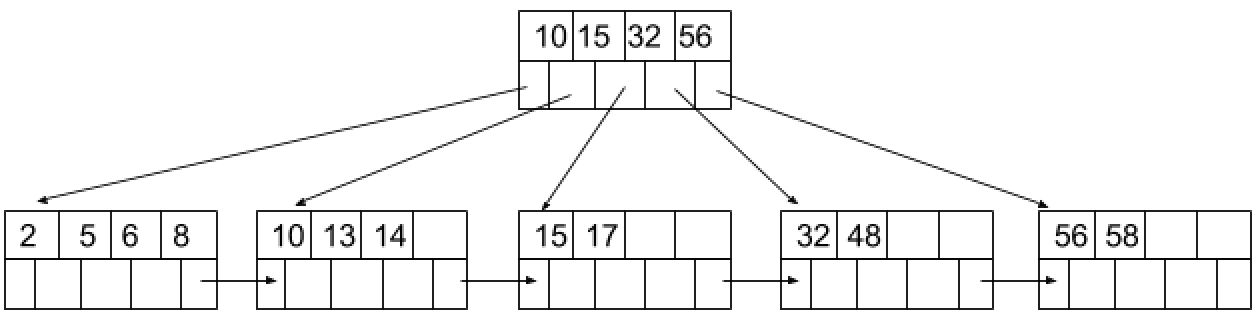
Myungjin Lee
5128876730

1a)
Process :

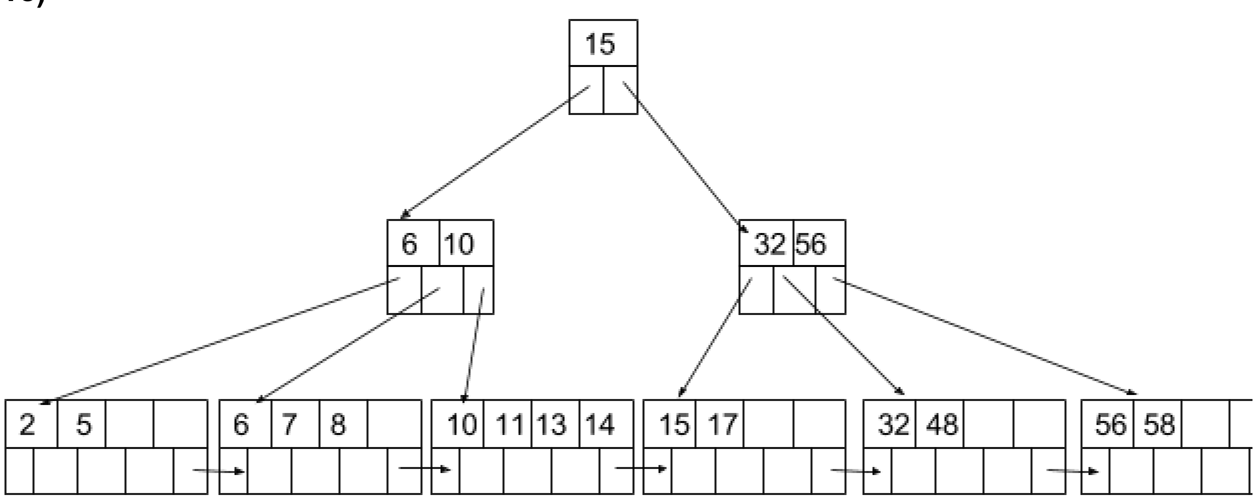
1. Find the first leaf where age >= 10 and age! = 15 in root (in this case)
2. Go to the block which has keys of age >=10 and traverse next block through leaf node.
3. When the keys are satisfied with the range age >=10 then read the corresponding internal node except age=15.

The number of blocks : 5

1b)



1c)



2-a) Nested-loop join with R as the outer relation

Step1 : Read R once (cost : B(R))

Memory : (M-2) blocks of R as a hash table(since 1 for input buffer for S and 1 for output buffer)

Step2 : The size of each run is (M-2), so that the number of runs is $\lceil B(R)/(M-2) \rceil$.

Each time of $\lceil B(R)/(M-2) \rceil$ reads S (cost : $B(S)\lceil B(R)/(M-2) \rceil$)

=> $\lceil 500/98 \rceil$ = 6 runs for R, each time for 6 runs reads S

Total cost: $B(R) + B(S)\lceil B(R)/(M-2) \rceil$

$= 500 + 10{,}000 * \lceil 500/98 \rceil = 60{,}500$

2-b) Nested-loop join with S as the outer relation

Step1 : Read S once (cost : B(S))

Memory : (M-2) blocks of S as a hash table(since 1 for input buffer for R and 1 for output buffer)

Step2 : The size of each run is (M-2), so that the number of runs are $\lceil B(S)/(M-2) \rceil$.

Each time of B(S)/(M-2) reads R (cost : $B(R)\lceil B(S)/(M-2) \rceil$)

=> $\lceil 10{,}000/98 \rceil$ = 103 runs for S, each time for 103 runs reads R

Total cost: $B(S) + B(R)\lceil B(S)/(M-2) \rceil$

$= 10{,}000 + 500 * \lceil 10{,}000/98 \rceil = 61{,}500$

2-c) Sort-merge join

Step1 : Split R into runs of size M and split S into runs of size M. (cost : 2B(R) + 2B(S))

=> 500/100 = 5 runs of size 100 for R, 10,000/100 = 100 runs of size 100 for S

Step2 : Merge M-1 runs into a new run (cost : 2B(R) + 2B(S))

=> $\lceil 105/99 \rceil$ = 2 runs

Step3 : Join 2 runs from R and S (cost : B(R) + B(S))

Total cost: 2B(R) + 2B(S) + 2B(R) + 2B(S) + B(R) + B(S) = 5B(R) + 5B(S)

= 5*500 + 5*10,000 =52,500

2-d) Simple sort-based join

Sort R:

Step1 : Split R into runs of size M so that we get B(R)/M lists : 500/100 = 5 lists of size 100

Step2 : Merge 5 runs : 1 sorted list of size 500 (cost : 2B(R)*(# of pass))

Sort S:
Step1 : Split S into runs of size M so that we get B(S)/M lists
=> 10,000/100 = 100 lists of size 100
Step2 : Merge M-1 runs into a new run (since 1 is for output buffer)
=> 1 sorted list of size 9,900 and 1 sorted list of size 100
Step3 : Merge two lists (1 size of 9,900 and 1 size of 100) (cost : 2B(S)*(# of pass))
=> 1 sorted list size of 10,000

Finally, read both relations in sorted order, match tuples (join R and S) (cost : B(R) + B(S))

Total cost: 2B(R)*(# of pass) + 2B(S)*(# of pass) + B(R) + B(S)
= 2*500*2 + 2*10,000*3 + 500 + 10,000 =72,500

2-e) Partitioned-hash join

Step1 : Hash R and S into M-1 buckets. (since 1 buffer is for input) (cost : 2B(R) + 2B(S))
(# buckets : M-1, size of buckets : B(R)/(M-1) for R and B(S)/(M-1) for S)
=> 99 buckets of size 500/99 for R and 99 buckets of size 10,000/99 for S
Step2 : Load the buckets of Ri with corresponding Si, one block at a time, output joining tuples with using another hash function
(cost : B(R) + B(S))

Total cost: 2B(R) + 2B(S) + B(R) + B(S) = 3B(R) + 3B(S)
= 3*500 + 3*10,000 = 31,500

2-f) Index join

Step1 : Iterate over R, for each tuple, fetch corresponding tuples from S
(# joined tuples : B(S)/V(S,a) from S)

Total cost: B(R) + B(S)T(R)/V(S,a)
= 500 + 10,000*5,000/200 = 250,500

The most efficient algorithm : Partitioned-hash join