1. B+tree

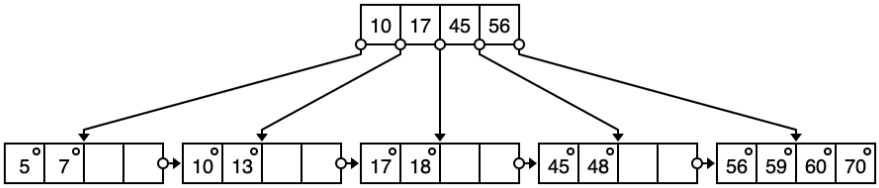

Figure 1: Original B+tree

(a) Read the root, follow the first pointer, read the the leaf with nodes 5 and 7, then read the right leaf with 10 and 13, continue reading the right leaf with 17 and 18, continue reading the right leaf with 45 and 48. Note that the last last leaf starts with value <= 56, which is larger than 50, so we do not need read the last leaf.
Cost: read 5 blocks

(b)

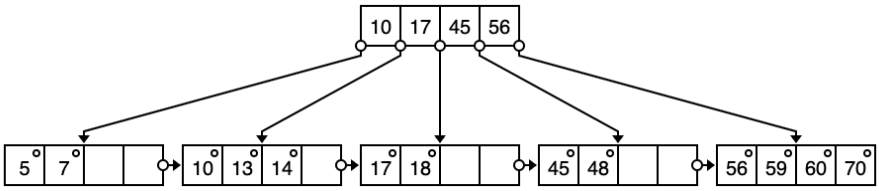

Figure 2: after inserting 14

[**Insert 14**] **Cost:** read 2 blocks, write 1 blocks, total 3 I/O

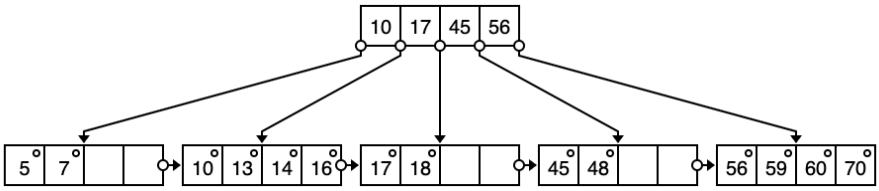

Figure 3: after inserting 16

[**Insert 16**] **Cost:** read 2 blocks, write 1 blocks, total 3 I/O

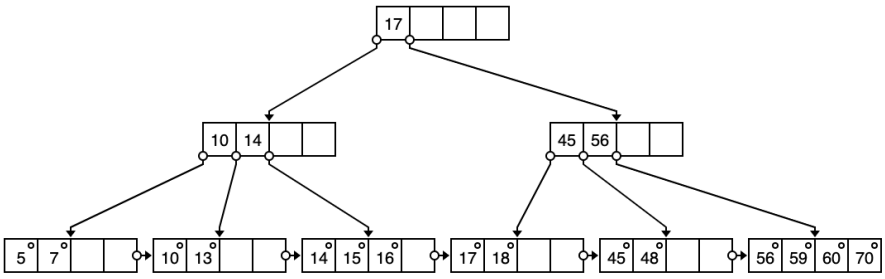

Figure 4: after inserting 15

[Insert 15] Cost: read 2 blocks, write 5 blocks, total 7 I/O
Total cost for all inserts: 13 blocks I/O

(c)

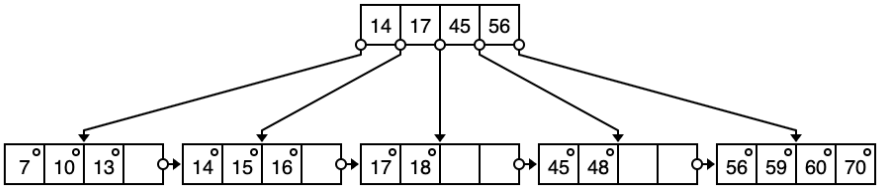

Figure 5: after deleting 5

[Delete 5] Cost: read 5 blocks, write 2 blocks, total 7 I/O

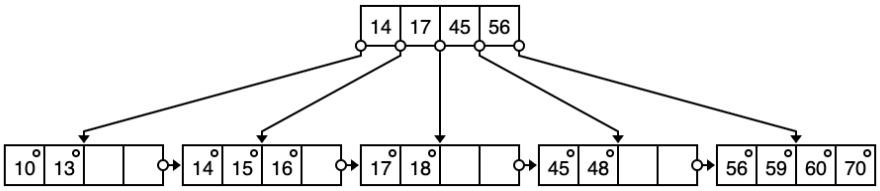

Figure 6: after deleting 7

[Delete 7] Cost: read 2 blocks, write 1 blocks, total 3 I/O

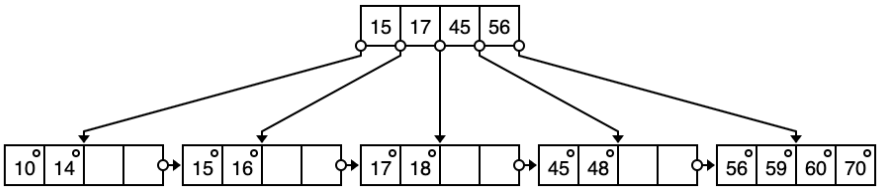

Figure 7: after deleting 13

[Delete 13] Cost: read 3 blocks, write 3 blocks, total 6 I/O

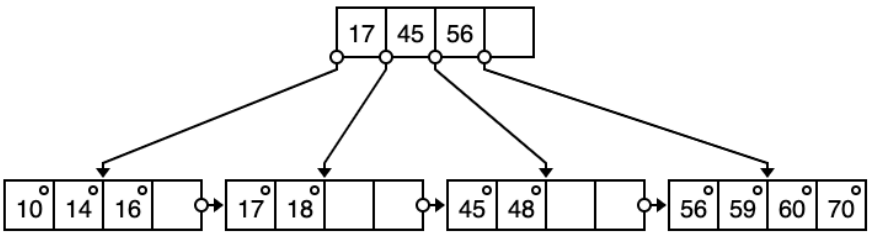

Figure 8: after deleting 15

[Delete 15] Cost: read 4 blocks, write 2 blocks, total 6 I/O

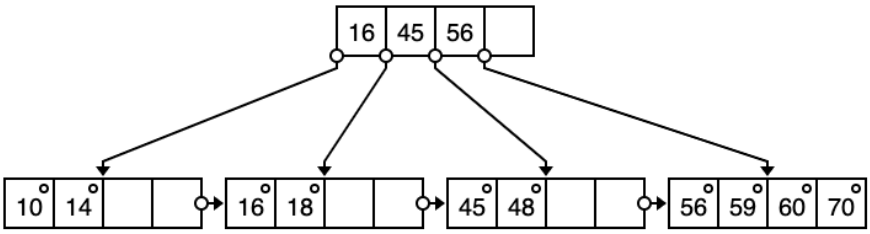

Figure 9: after deleting 17

[**Delete 17**] **Cost:** read 3 blocks, write 3 blocks, total 6 I/O

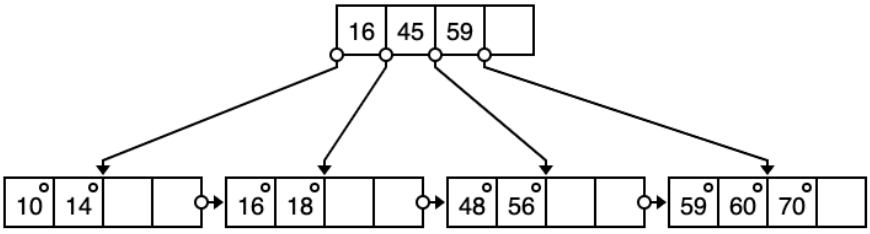

Figure 10: after deleting 45

[**Delete 45**] **Cost:** read 4 blocks, write 3 blocks, total 7 I/O

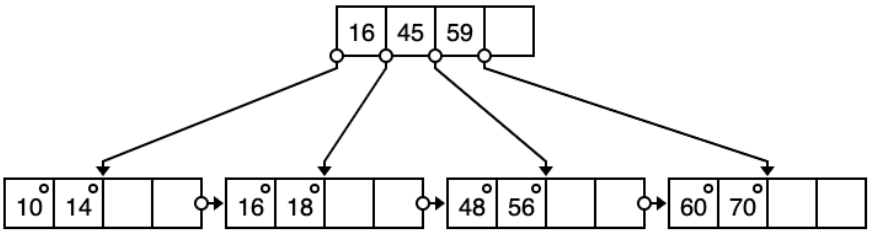

Figure 11: after deleting 59

[**Delete 59**] **Cost:** read 2 blocks, write 1 blocks, total 3 I/O
Total cost for all deletes: 38 blocks I/O

2. Natural-Joining Tables R(a,b) and S(a,c)

(a) For each 100 blocks b_r of R do
For each block b_s of S do
For each tuple r in b_r do
For each tuple s in b_s do
if r and s can be joined than output (r,s)

Cost:
Read R once, cost B(R) = 20,000
Number of outer loops = $\frac{B(R)}{M-2} = \frac{20000}{100} = 200$
For each outer loop read B(S) = 10,000
Total Cost: $= 20000 + 200 * 10000 = 2,020,000$ block I/O

(b) For each 100 blocks b_s of S do
For each block b_r of R do
For each tuple r in b_r do
For each tuple s in b_s do
if r and s can be joined than output (r,s)

Cost:
Read S once, cost B(S) = 10,000
Number of outer loops = $\frac{B(S)}{M-2} = \frac{10000}{100} = 100$
For each outer loop read B(R) = 20,000
Total Cost: $= 10000 + 100 * 20000 = 2,010,000$ block I/O

(c)
- sort R: output $\frac{20000}{100} = 200$ runs, with 100 blocks/run, **Cost:** 2B(R)
- sort S: output $\frac{10000}{100} = 100$ runs, with 100 blocks/run, **Cost:** 2B(S)
- Further merge R: output $\frac{200}{100} = 2$ runs, with $100 * 100 = 10000$ blocks/run, **Cost:** 2B(R)
- Further merge S: output $\frac{100}{100} = 1$ runs, with $100 * 100 = 10000$ blocks/run, **Cost:** 2B(S)
- Join 2 runs from R and 1 run from S: output 1 run, with 30000 blocks/run, **Cost:** B(R) + B(S)

Total Cost: 5B(R) + 5B(S) = 150,000 blocks I/O

(d)
- sort R: output $\frac{20000}{100} = 200$ runs, with 100 blocks/run, **Cost:** 2B(R)
- sort S: output $\frac{10000}{100} = 100$ runs, with 100 blocks/run, **Cost:** 2B(S)
- Further merge R: output $\frac{200}{100} = 2$ runs, with $100 * 100 = 10000$ blocks/run, **Cost:** 2B(R)
- Further merge R: output 1 runs, with 20000 blocks/run, **Cost:** 2B(R)
- Further merge S: output $\frac{100}{100} = 1$ runs, with $100 * 100 = 10000$ blocks/run, **Cost:** 2B(S)
- Join 1 run from R and 1 run from S: output 1 run, with 30000 blocks/run, **Cost:** B(R) + B(S)

Total Cost: 7B(R) + 5B(S) = 190,000 blocks I/O

(e)
- Hash R into 102-1=101 buckets, **Cost:** 2B(R)
- Hash S into 102-1=101 buckets, **Cost:** 2B(S)
- Join 101 buckets from R and 101 buckets from S

Total Cost: 3B(R) + 3B(S) = 90,000 blocks I/O

(f) For each 102-2=100 blocks b_r of R do
For each tuple r in b_r do
Lookup corresponding value in S index

Cost:

Read R once, cost B(R) = 20,000
Number of outer loops = T(R) = 200,000
For each outer loop read look up in S index, on average cost = $\frac{B(S)}{V(S,a)} = \frac{10000}{100} = 100$ blocks
Total Cost: $= 20000 + 100 * 200000 = 20,020,000$ blocks I/O

Partitioned-hash join is most efficient.