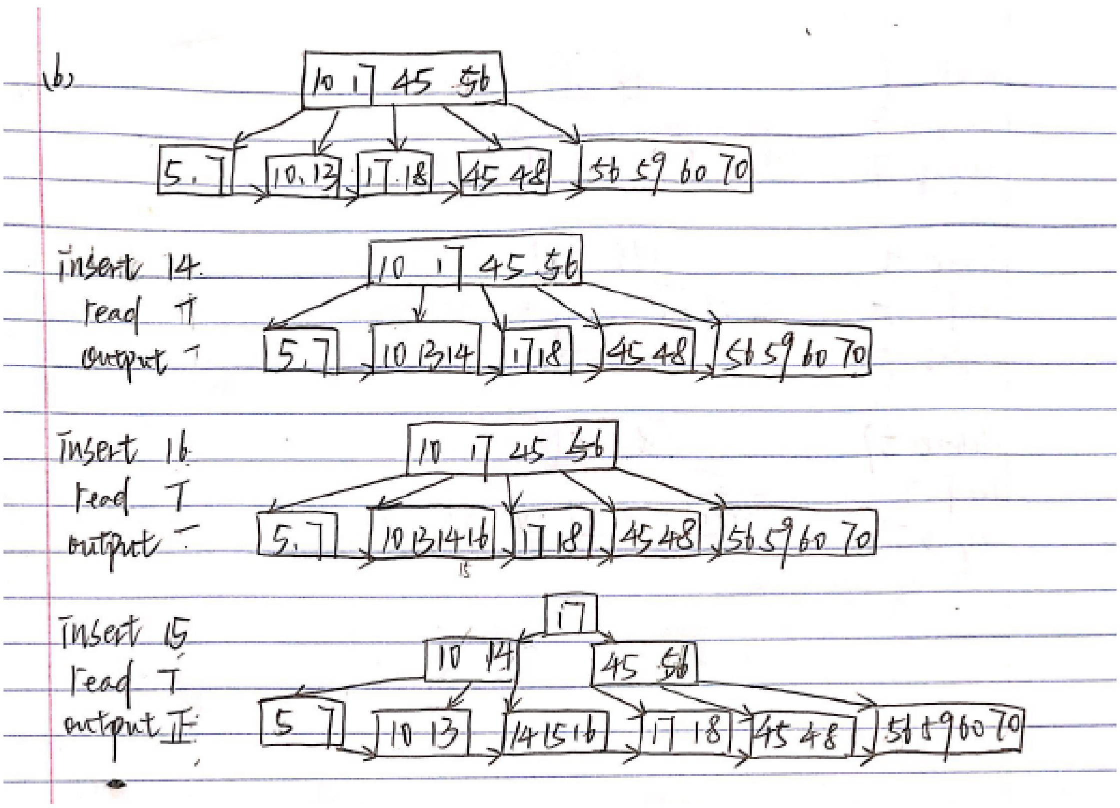
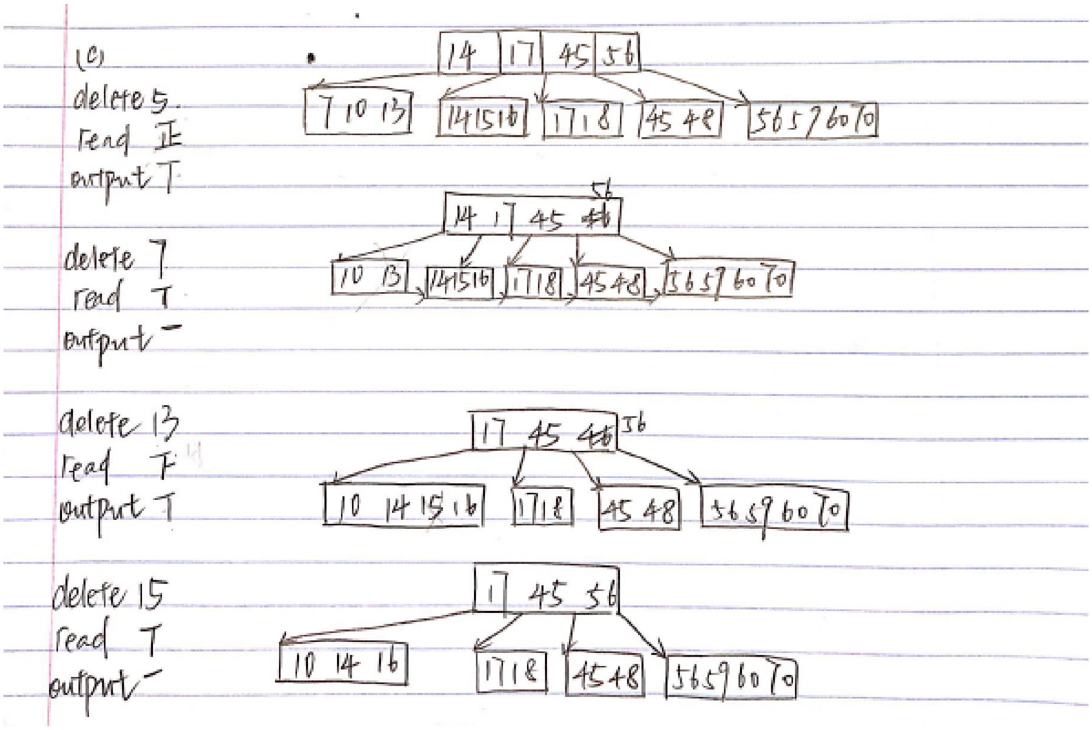
Peiying Lyu   8109407016
HW4

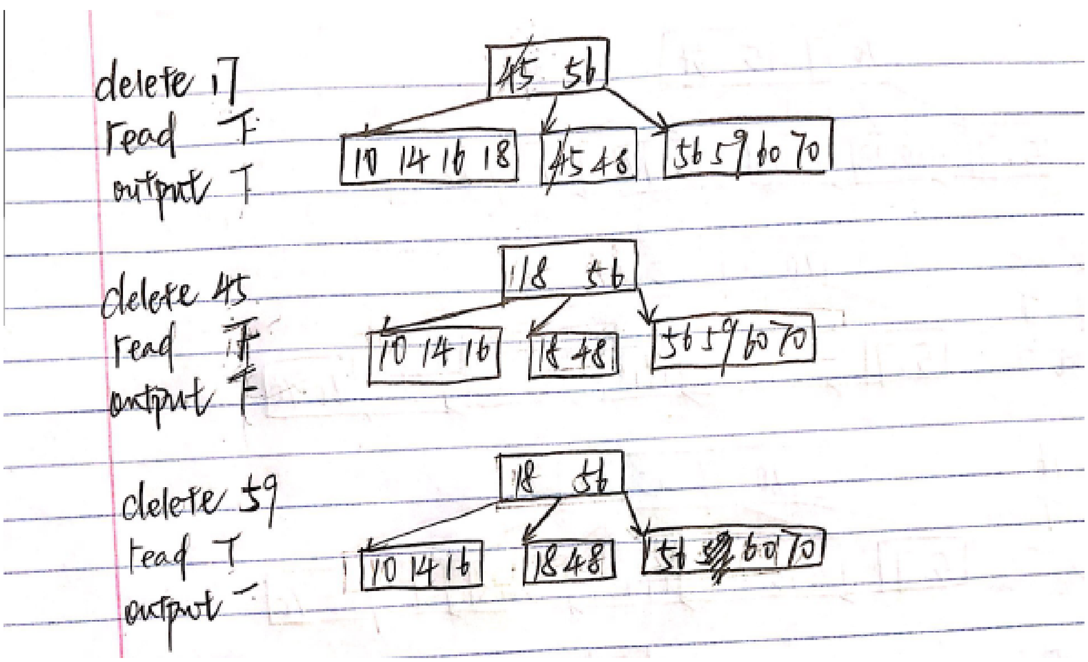
1.

(a).
First, going to the root. --- 1 block
Second, going to the second child where value of node is beginning from 10 ---1 block
Third, searching till the value of node is 48, --- 2 blocks.
Then, read the 5th child to estimate which is >50, --- 1 block
Total cost = 5 blocks.



(b).   **Total cost = 13 blocks**





(c). **Total cost = 33 blocks**

2.

(a) nested-loop join

1.For each (M-2)=100 blocks in R <u>(read cost=B(R))</u>

2.iterate through every block in S;
3.iterate through each tuple in R block,
4.iterate through each tuple in S block,
5.if condition matches, return (r, s).
Run this algorithm B(R)/(M-2) times, and each time need to read S: costs = B(S)*B(R)/(M-2).

Total cost = B(R) + B(R)B(S)/(M-2) = 20 000 + (20 000 * 10 000)/100 = 2,020,000

(b) nested-loop join

1.For each (M-2)=100 blocks in S (read cost=B(S)),
2.iterate through every block in R;
3.iterate through each tuple in S block
4.iterate through each tuple in R block,
5.if condition matches, return (s, r).
Run this algorithm B(S)/(M-2) times, and each time need to read R: costs = B(S)*B(R)/(M-2).

Total cost = B(S) + B(R)B(S)/(M-2) = 10 000 +(20 000 *10 000)/100 = 2,010,000

(c) sort-merge join

Step 1: Sort R: load 100 blocks of R at a time, sort them and write back on disk. This generates **200 runs**, each of size 100 blocks. Cost=2B(R)=40000

Step 2: Sort S: load 100 blocks of S at a time, sort them and write back on disk. This generates **100 runs**, each of size 100 blocks. Cost=2B(S)=20,000

Since # of runs of R and S > M, we cannot merge them directly.

Step 3: Sort R: load 100 blocks of R at a time, sort them and write back on disk. This generates **2 runs** Cost=2B(R)=40000

Since # of runs of R and S=102 > M=101, we cannot merge them directly.

Step 4: Sort R: load 100 blocks of s at a time, sort them and write back on disk. This generates **1 runs** Cost=2B(S)=20000

Step 4: Final merge: Cost = B(R)+B(S) = 30 000

Total cost = 150000

(d) simple sort-merge join

Step 1: Sort R: load 100 blocks of R at a time, sort them and write back on disk. This generates **200 runs**, each of size 100 blocks.   Cost=2B(R)=40000

Step 2: Sort S: load 100 blocks of S at a time, sort them and write back on disk. This generates **100 runs**, each of size 100 blocks.   Cost=2B(S)=20,000

Step 3: Load 100 blocks of R at a time, sort and write back to the disk. This generates **2 runs**, each of size 10000 blocks. Cost = 2B(R) = 40000

Step 4: Load 100 blocks of S at a time, sort and write back to the disk. This generate **1 runs**, each of size 10000 blocks   Cost=2B(S)=20,000

Step 5: Load 100 blocks of R at a time, sort and write back to the disk. This generate **1 runs,** each of size 20000 blocks. Cost = 2B(R) = 40000

Step 6: Final merge: Cost = B(R)+B(S) = 30 000

Total cost = 190000

(e) Partitioned-hash join

Step1： Hash R into M – 1 = 101 buckets and send them back to disk: Cost = 2 B(R) = 40,000

Step2： Hash S into M – 1 = 101 buckets and send them back to disk: Cost = 2 B(S) = 20,000

Step3： Read each bucket of smaller relation into memory. For each bucket, read the same bucket of the: larger relation block by block and join the matching tuples. Cost: B(R) + B(S) = 30,000

Total cost = 3B(R) +3B(S) = 90,000

(f) Index-based join

S, R are clustered
1. Load 100 blocks from R
2. For each tuple r(a.b), fetch corresponding tuples from S
3. Join the tuples which are matched and send it to output buffer

Total cost = B(R) + T(R)B(S)/v(s,a) = 200 00 + 200000*10000/100 = 20,020,000

The best one is Partitioned-hash join.