SOLUTION

ISAM

1) Bulk load the ISAM with values 46, 10, 70, 49, 23, 40, 59, 29, 34, 54, 75, 30

2) Insert 80

3) Insert 24

4) Remove 70

5) How many I/O to find 80?
   3

6) How many I/O to find out 79 is not in index?
   3

7) How many I/O to find 40?
   2

8) How many I/O to find 10?
   2 if unique index, 3 otherwise
**B+Tree**

1) Bulk load the B+Tree with values 46, 10, 70, 23, 40, 59, 29, 34, 54

![B+Tree 1](image)

2) Insert 80

![B+Tree 2](image)

3) Insert 24

![B+Tree 3](image)

4) Remove 70

![B+Tree 4](image)

5) Go back to the original bulk loaded B+Tree (step 1). Remove 70

![B+Tree 5](image)

6) Insert 24

![B+Tree 6](image)

7) Insert 80

![B+Tree 7](image)

Notice: although the index has the same leaves as step 4, its organization is different because of the order of operations.
8) Insert 35

9) Insert 85

10-13) All answers are 3! This is one of the benefits of B+Trees, very predictable!

14) Suppose that you have a file that is already sorted in key order and you want to construct a dense, clustered B+ Tree index on this file using <key, RID> pairs for data entries. A simple way to accomplish this is to create a B+Tree, and then sequentially scan the file, inserting an index entry for each record using the normal B+Tree insertion routine. What performance and storage utilization problems are there with this approach?

Performance and storage utilization problem: Most leaf nodes are half full as a result of inserting sorted key values and splitting nodes at the leaf level in a way that each of the two nodes is half full. So utilization is roughly 50%. Also because the tree is large and to insert each key, the B+Tree is traversed from root to the leaf the performance is sub-optimal.

15) Briefly describe a change to the B+Tree insertion routine that would solve the problems you identified in question 14.

Two solutions:
   a. One is to change the splitting method. When a leaf node is split, do not move half key values to the newly created node. Just move the last key to the new node so that the left node is still full.
   b. Bulk loading - That is to build B+Tree bottom up from sorted key values.

16) \[ 8KB = 8192 \text{ bytes per page} \]
\[ 292 \text{ keys} \times 20 + 293 \text{ pointers} \times 8 = 8184 \text{ bytes} + 8 \text{ bytes to spare (for the pointer to the next page on leaf nodes)} \]
\[ 293 \times 293 \times 293 \times 292 = 7,344,897,004 \text{ entries} \]