Solutions

Answer 5.1

1. The data entry with key 9 is inserted on the second leaf page. The resulting tree is shown in figure 10.2.

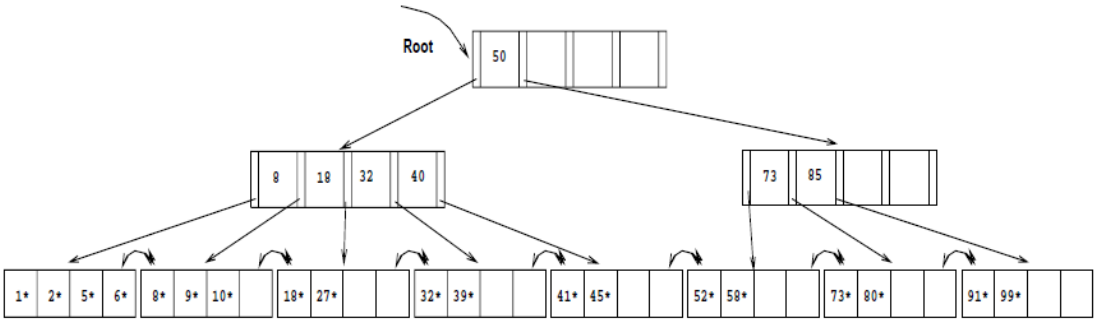

Figure 10.2

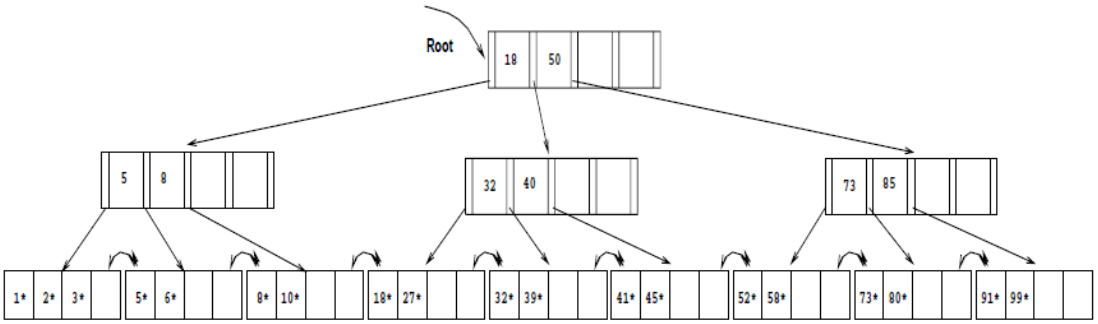

Figure 10.3

2. The data entry with key 3 goes on the first leaf page F. Since F can accommodate at most four data entries (d = 2), F splits. The lowest data entry of the new leaf is given up to the ancestor which also splits. The result can be seen in figure 10.3.

   NOTE: Another valid solution is to move the key 3 to the new leaf page created by the split. In this case, the value 3 will also be copied up to the internal, ancestor node.

   Regarding the insertion cost, one needs to 1) read the root, 2) read the leftmost child C of the root, 3) read the leftmost leaf F, 4) allocate and write a new leaf node F* due to lack of space, 5) update F, 6) read and update the right sibling of the newly created leaf F*, 7) allocate and write a new internal node due to lack of space, 8) update C, 9) update the root. In total, the insertion process involved 4 reads and 6 writes.

NOTE: This exercise assumes enough memory space to keep all read pages; once a page is read from disk for the first time, subsequent accesses to the page incur no additional I/O.

3. The data entry with key 8 is deleted, resulting in a leaf page N with less than two data entries. The left sibling L is checked for redistribution. Since L has more than two data entries, the remaining keys are redistributed between L and N, resulting in the tree in figure 10.4.
4. As is part 3, the data entry with key 8 is deleted from the leaf page N. N's right sibling R is checked for redistribution, but R has the minimum number of keys. Therefore the two siblings merge. The key in the ancestor which distinguished between the newly merged leaves is deleted. The resulting tree is shown in figure 10.5.

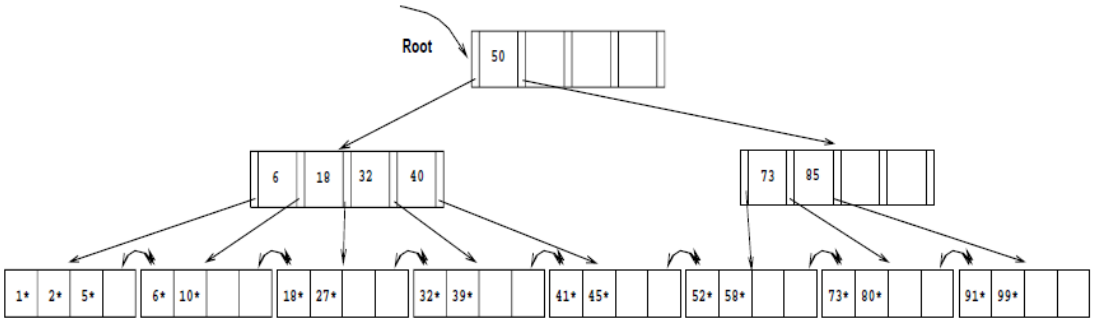

Figure 10.4

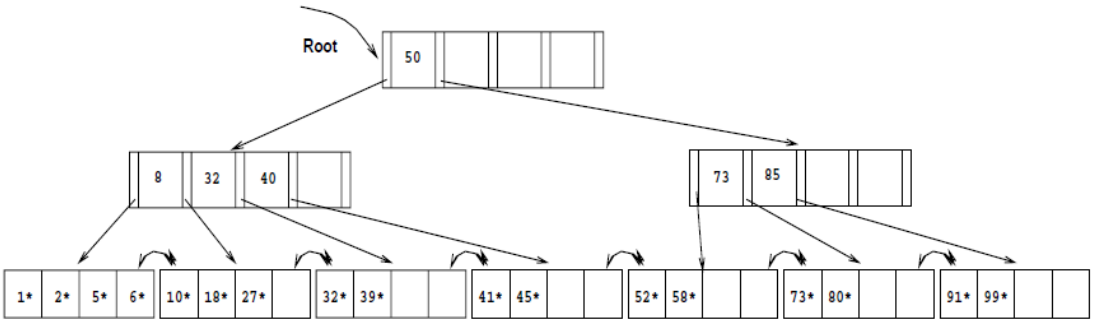

Figure 10.5

5. The data entry with key 46 can be inserted without any structural changes in the tree. But the removal of the data entry with key 52 causes its leaf page L to merge with a sibling (we chose the right sibling). This results in the removal of a key in the ancestor A of L and thereby lowering the number of keys on A below the minimum number of keys. Since the left sibling B of A has more than the minimum number of keys, redistribution between A and B takes place. The final tree is depicted in figure 10.6.

6. Deleting the data entry with key 91 causes a scenario similar to part 5. The result can be seen in figure 10.7.
7. The data entry with key 59 can be inserted without any structural changes in the tree. No sibling of the leaf page with the data entry with key 91 is affected by the insert. Therefore, deleting the data entry with key 91 changes the tree in a way similar to part 6. The result is depicted in figure 10.8.
8. Considering checking the right sibling for possible merging first, the successive deletion of the data entries with keys 32, 39, 41, 45 and 73 results in the tree shown in figure 10.9.

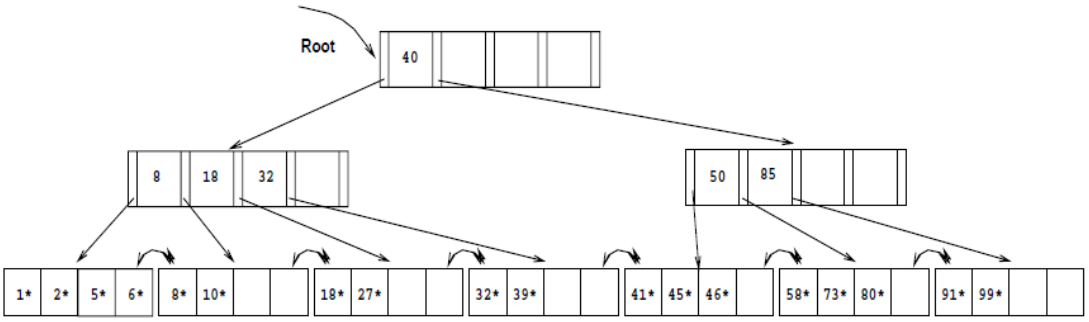

Figure 10.6

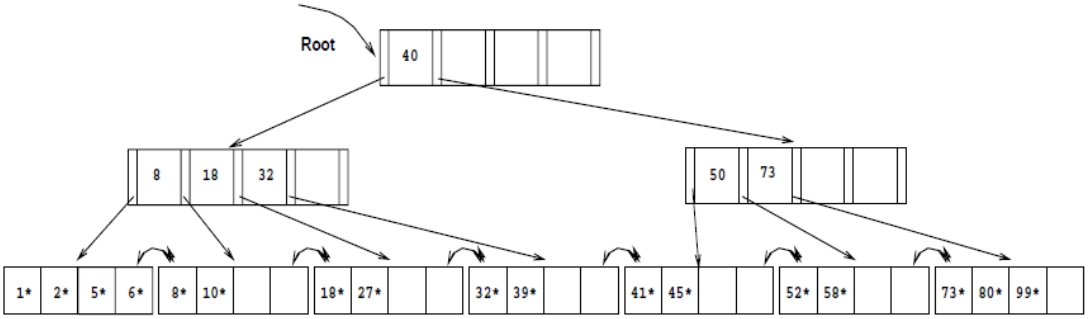

Figure 10.7

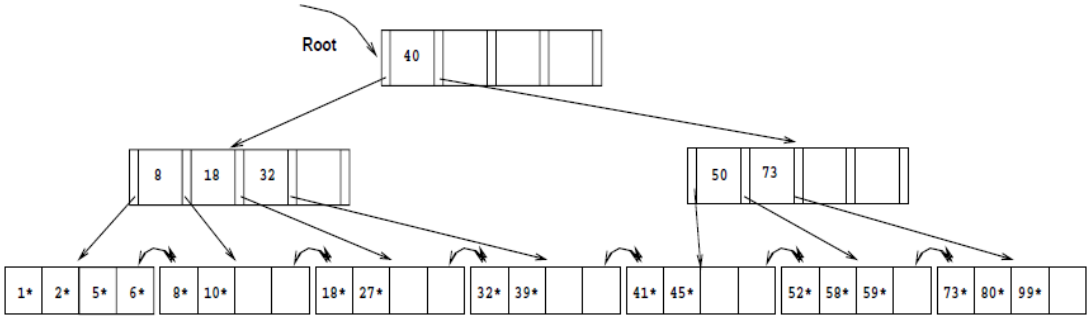

Figure 10.8

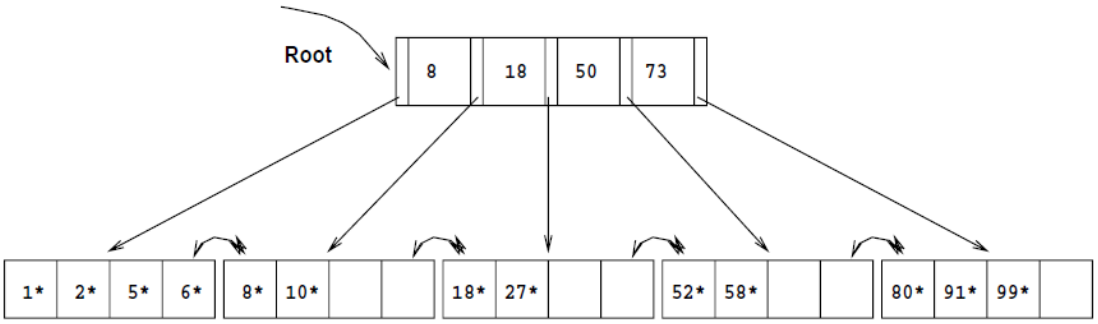

Figure 10.9

Answer 5.2 The answer to each question is given below.

1. By the definition of a B+ tree, each index page, except for the root, has at least d and at most 2d key entries. Therefore—with the exception of the root—the minimum space utilization guaranteed by a B+ tree index is 50 percent.
2. A static index without overflow pages is faster than a dynamic index on inserts and deletes, since index pages are only read and never written. If the set of keys that will be inserted into the tree is known in advance, then it is possible to build a static index which reserves enough space for all possible future inserts. Also, if the system goes periodically offline, static indices can be rebuilt and scaled to the current occupancy of the index. Infrequent or scheduled updates hint for a static index structure.