Exercise 9.6 Consider again the disk specifications from Exercise 9.5, and suppose that a block size of 1024 bytes is chosen. Suppose that a file containing 100,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

1. How many records fit onto a block?
2. How many blocks are required to store the entire file? If the file is arranged sequentially on the disk, how many surfaces are needed?
3. How many records of 100 bytes each can be stored using this disk?
4. If pages are stored sequentially on disk, with page 1 on block 1 of track 1, what page is stored on block 1 of track 1 on the next disk surface? How would your answer change if the disk were capable of reading and writing from all heads in parallel?
5. What time is required to read a file containing 100,000 records of 100 bytes each sequentially? Again, how would your answer change if the disk were capable of reading/writing from all heads in parallel (and the data was arranged optimally)?
6. What is the time required to read a file containing 100,000 records of 100 bytes each in a random order? To read a record, the block containing the record has to be fetched from disk. Assume that each block request incurs the average seek time and rotational delay.

Answer 9.6 1. $1024/100 = 10$. We can have at most 10 records in a block.

2. There are 100,000 records all together, and each block holds 10 records. Thus, we need 10,000 blocks to store the file. One track has 25 blocks, one cylinder has 250 blocks. we need 10,000 blocks to store this file. So we will use more than one cylinders, that is, need 10 surfaces to store this file.
3. The capacity of the disk is 500,000K, which has 500,000 blocks. Each block has 10 records. Therefore, the disk can store no more than 5,000,000 records.
4. There are 25K bytes, or we can say, 25 blocks in each track. It is block 26 on block 1 of track 1 on the next disk surface.

   If the disk were capable of reading/writing from all heads in parallel, we can put the first 10 pages on the block 1 of track 1 of all 10 surfaces. Therefore, it is block 2 on block 1 of track 1 on the next disk surface.
5. A file containing 100,000 records of 100 bytes needs 40 cylinders or 400 tracks in this disk. The transfer time of one track of data is 0.011 seconds. Then it takes $400 \times 0.011 = 4.4 seconds$ to transfer 400 tracks.

   This access seeks the track 40 times. The seek time is $40 \times 0.01 = 0.4 seconds$. Therefore, total access time is $4.4 + 0.4 = 4.8 seconds$.

If the disk were capable of reading/writing from all heads in parallel, the disk can read 10 tracks at a time. The transfer time is 10 times less, which is 0.44 seconds. Thus total access time is $0.44 + 0.4 = 0.84 seconds$

6. For any block of data, $averageaccesstime = seektime + rotationaldelay + transfertime$.

$$seektime = 10msec$$

$$rotationaldelay = 6msec$$

$$transfertime = \frac{1K}{2,250K/sec} = 0.44msec$$

The average access time for a block of data would be 16.44 msec. For a file containing 100,000 records of 100 bytes, the total access time would be 164.4 seconds.

Exercise 9.7 Explain what the buffer manager must do to process a read request for a page. What happens if the requested page is in the pool but not pinned?

Answer 9.7 When a page is requested the buffer manager does the following:

1. The buffer pool is checked to see if it contains the requested page. If the page is in the pool, skip to step 2. If the page is not in the pool, it is brought in as follows:
   (a) A frame is chosen for replacement, using the replacement policy.
   (b) If the frame chosen for replacement is dirty, it is *flushed* (the page it contains is written out to disk).
   (c) The requested page is read into the frame chosen for replacement.
2. The requested page is *pinned* (the *pin_count* of the chosen frame is incremented) and its address is returned to the requester.

Note that if the page is not pinned, it could be removed from buffer pool even if it is actually needed in main memory. *Pinning* a page prevents it from being removed from the pool.

Exercise 9.8 When does a buffer manager write a page to disk?

Answer 9.8 If a page in the buffer pool is chosen to be replaced and this page is dirty, the buffer manager must write the page to the disk. This is also called flushing the page to the disk.

Sometimes the buffer manager can also force a page to disk for recovery-related purposes (intuitively, to ensure that the log records corresponding to a modified page are written to disk before the modified page itself is written to disk).

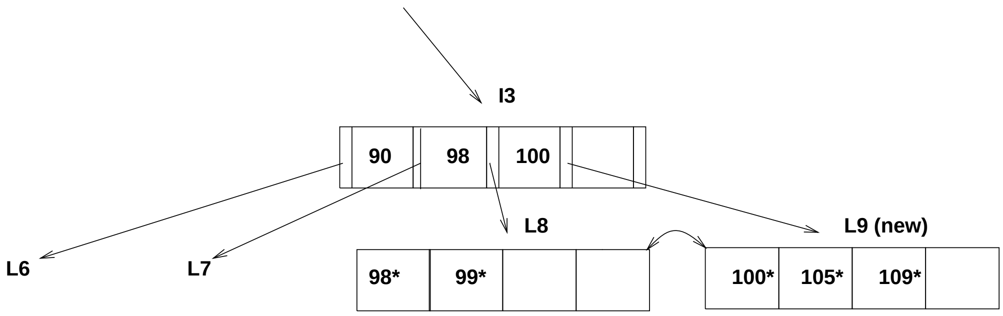

Figure 10.11

2. Show the B+ tree that would result from inserting a record with search key 109 into the tree.

3. Show the B+ tree that would result from deleting the record with search key 81 from the original tree.

4. Name a search key value such that inserting it into the (original) tree would cause an increase in the height of the tree.

5. Note that subtrees A, B, and C are not fully specified. Nonetheless, what can you infer about the contents and the shape of these trees?

6. How would your answers to the preceding questions change if this were an ISAM index?

7. Suppose that this is an ISAM index. What is the minimum number of insertions needed to create a chain of three overflow pages?

Answer 10.2 The answer to each question is given below.

1. I1, I2, and everything in the range [L2..L8].

2. See Figure 10.11. Notice that node L8 is split into two nodes.

3. Assuming that there is redistribution from the right sibling, the solution can be seen in Figure 10.12.

4. There are many search keys X such that inserting X would increase the height of the tree. Any search key in the range [65..79] would suffice. A key in this range would go in L5 if there were room for it, but since L5 is full already and since it can't redistribute any data entries over to L4 (L4 is full also), it must split; this in turn causes I2 to split, which causes I1 to split, and assuming I1 is the root node, a new root is created and the tree becomes taller.

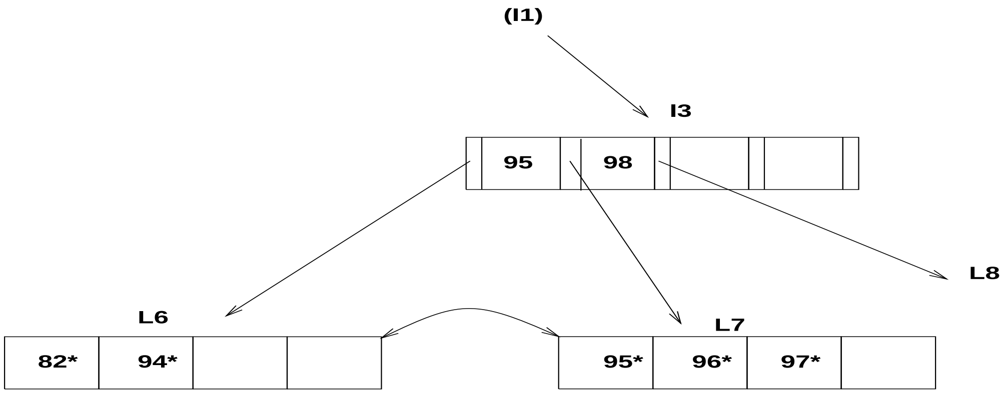

Figure 10.12

5. We can infer several things about subtrees A, B, and C. First of all, they each must have height one, since their "sibling" trees (those rooted at I2 and I3) have height one. Also, we know the ranges of these trees (assuming duplicates fit on the same leaf): subtree A holds search keys less than 10, B contains keys ≥ 10 and < 20, and C has keys ≥ 20 and < 30. In addition, each intermediate node has at least 2 key values and 3 pointers.

6. The answers for the questions above would change as follows if we were dealing with ISAM trees instead of B+ trees.

   (a) This is only a search, so the answer is the same. (The tree structure is not modified.)

   (b) Because we can never split a node in ISAM, we must create an overflow page to hold inserted key 109.

   (c) Search key 81 would simply be erased from L6; no redistribution would occur (ISAM has no minimum occupation requirements).

   (d) Being a *static* tree structure, an ISAM tree will never change height in normal operation, so there are no search keys which when inserted will increase the tree's height. (If we inserted an X in [65..79] we would have to create an overflow page for L5.)

   (e) We can infer several things about subtrees A, B, and C. First of all, they each must have height one, since their "sibling" trees (those rooted at I2 and I3) have height one. Here we suppose that we create a balanced ISAM tree. Also, we know the ranges of these trees (assuming duplicates fit on the same leaf): subtree A holds search keys less than 10, B contains keys ≥ 10 and < 20, and C has keys ≥ 20 and < 30. In addition, each of A, B, and C contains five leaf nodes (which may be of arbitrary fullness), and these nodes are the first 15 consecutive pages prior to L1.

7. If this is an ISAM tree, we would have to insert at least nine search keys in order to develop an overflow chain of length three. These keys could be any that would map to L4, L5, L7, or L8, all of which are full and thus would need overflow pages on the next insertion. The first insert to one of these pages would create the first overflow page, the fifth insert would create the second overflow page, and the ninth insert would create the third overflow page (for a total of one leaf and three overflow pages).

Exercise 10.3 Answer the following questions:

1. What is the minimum space utilization for a B+ tree index?
2. What is the minimum space utilization for an ISAM index?
3. If your database system supported both a static and a dynamic tree index (say, ISAM and B+ trees), would you ever consider using the *static* index in preference to the *dynamic* index?

Answer 10.3 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. The minimum space utilization by an ISAM index depends on the design of the index and the data distribution over the lifetime of ISAM index. Since an ISAM index is static, empty spaces in index pages are never filled (in contrast to a B+ tree index, which is a dynamic index). Therefore the space utilization of ISAM index pages is usually close to 100 percent by design. However, there is no guarantee for leave pages' utilization.
3. 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 off line, static indices can be rebuilt and scaled to the current occupancy of the index. Infrequent or scheduled updates are flags for when to consider a static index structure.

Exercise 10.4 Suppose that a page can contain at most four data values and that all data values are integers. Using only B+ trees of order 2, give examples of each of the following:

1. A B+ tree whose height changes from 2 to 3 when the value 25 is inserted. Show your structure before and after the insertion.

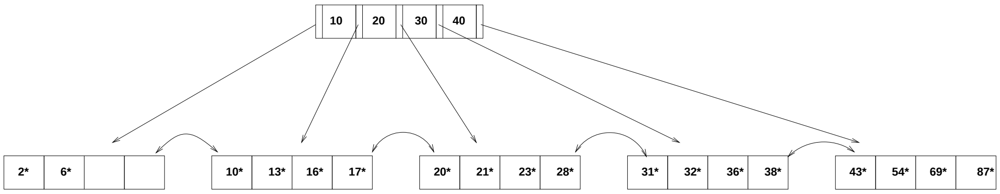

Figure 10.13

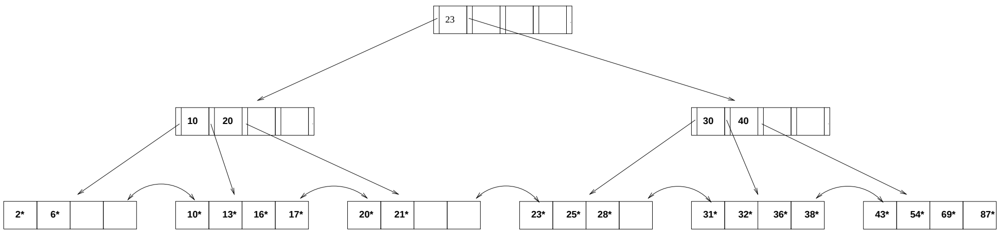

Figure 10.14

2. A B+ tree in which the deletion of the value 25 leads to a redistribution. Show your structure before and after the deletion.

3. A B+ tree in which the deletion of the value 25 causes a merge of two nodes but without altering the height of the tree.

4. An ISAM structure with four buckets, none of which has an overflow page. Further, every bucket has space for exactly one more entry. Show your structure before and after inserting two additional values, chosen so that an overflow page is created.

Answer 10.4 For these answers, two illustrations are given, one showing the tree before the specified change and one showing it after.

1. See Figures 10.13 and 10.14.

2. See Figures 10.15 and 10.16.

3. See Figures 10.17 and 10.18.

4. See Figures 10.19 and 10.20 (inserted 27 and 29).

Exercise 10.5 Consider the B+ tree shown in Figure 10.21.

1. Identify a list of five data entries such that:

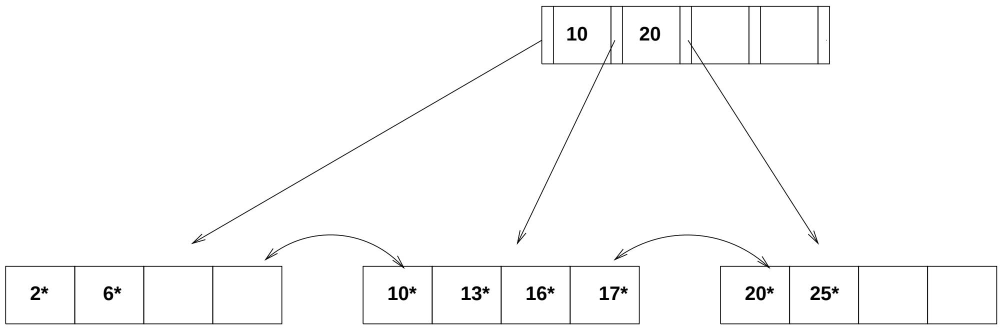

Figure 10.15

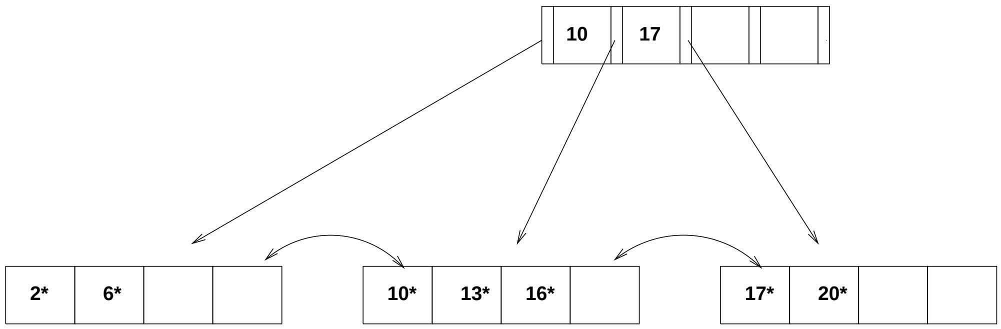

Figure 10.16

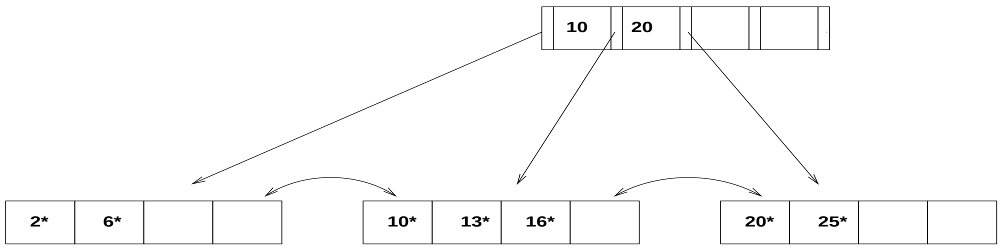

Figure 10.17

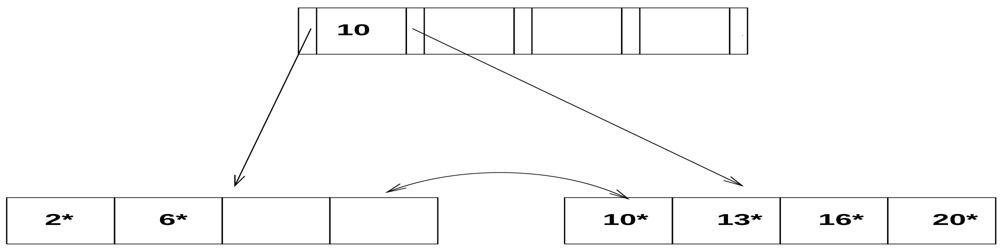

Figure 10.18

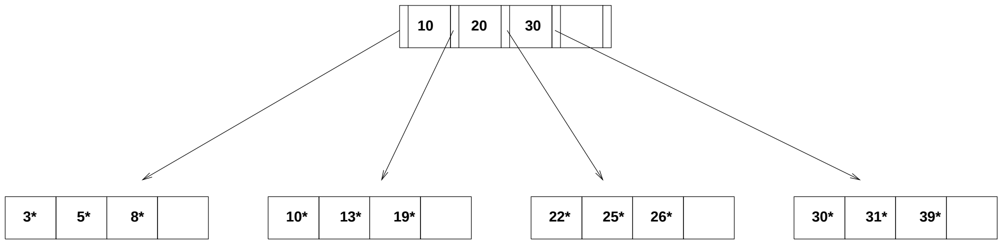

Figure 10.19

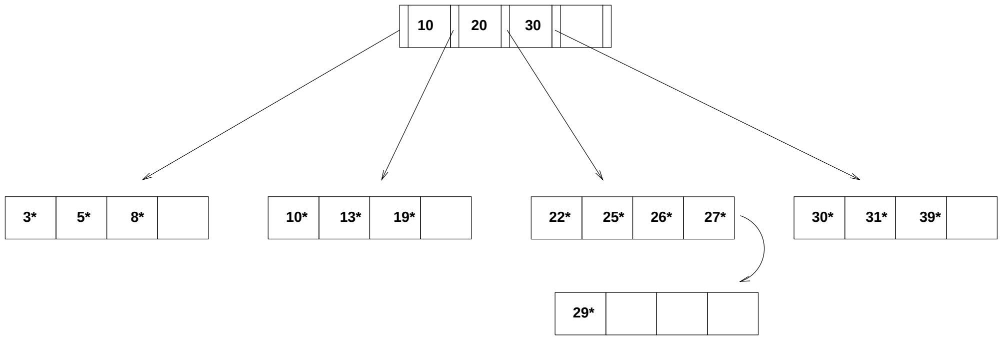

Figure 10.20

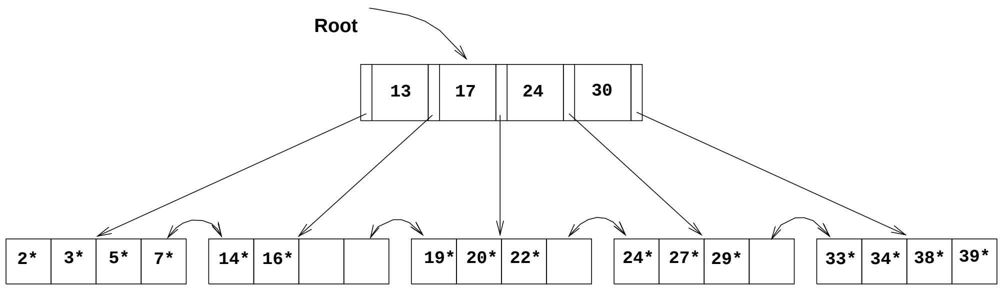

Figure 10.21 Tree for Exercise 10.5