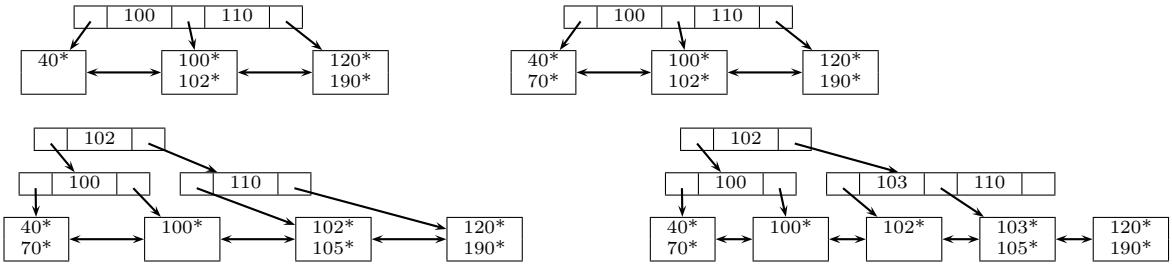
**1A.** N = 20000/10 = 2000
**1B.** M = $\lceil log_4(10,000) \rceil = 7$.
**2A.** The smallest unit of data transfer is a block; i.e., each transfer is in terms of multiple of 8KB block size. Each data record retrieval requires a disk seek and rotational delay: $2(s + r + 8/d)$
**2B.** Both records are read with one seek and one rotational delay: $s + r + 16/d$
**3.** B = 7 or 8. Outer relation is smaller relation (ie R with 20 pages). Allocate 1 page for inner relation, 1 page for output, and B-2 pages for outer relation. Let N denote number of times inner is scanned. $N = \lceil 20/B - 2 \rceil$.
**4A.** L = 100.
**4B.** P = 5000.
**5.** Clearly, if $x*$ is inserted into the bucket with (a*, b*) , this will cause the bucket to be split and the directory to be doubled since the bucket's local depth has the same value as the global depth.

If $x*$ is inserted into the bucket with (c*, d*) , this will cause the bucket to be split and a new bucket (corresponding to directory entry 101) to be created. If the last three bits of c, d, and x are all 001 (or all 101), then the overflow will remain unresolved, and a further split will be required which will double the directory.

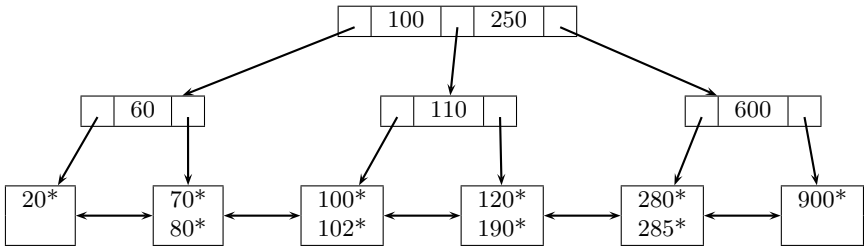
Thus, the last three bits of $x$ could be 000, 001, or 101.
**6.**

- The total number of internal nodes in $T'$ is 3.
- The total number of leaf nodes in $T'$ is 5.
- There is a leaf node in $T'$ with data entries 103* and 105*.



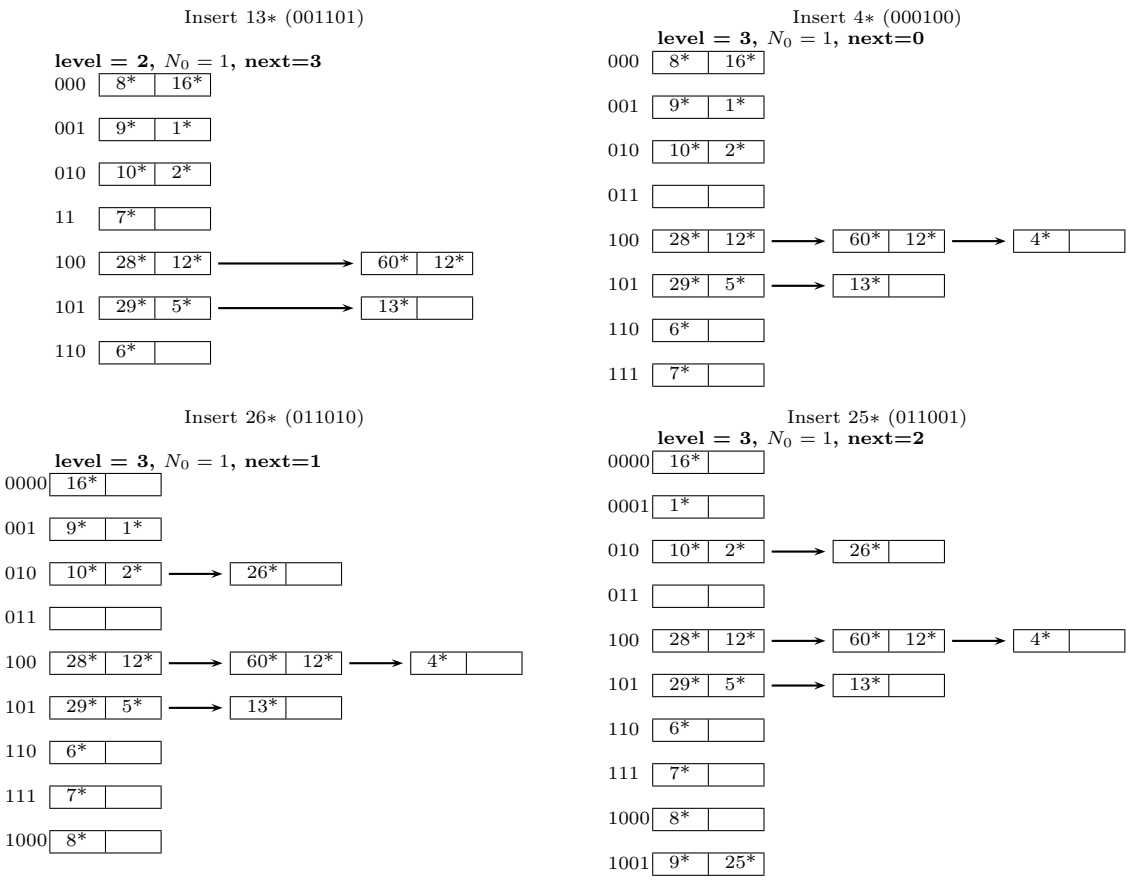

**7.**

- The total number of internal nodes in $T'$ is 4.
- The total number of leaf nodes in $T'$ is 6.
- There is an internal node in $T'$ with exactly one index entry for the key value 110.
- There is an internal node in $T'$ with exactly one index entry for the key value 600.




**8.**

- L = 3.
- N = 2.
- $B_2 = 1$.
- $B_3 = 2$.




**9A.** $B_{min} = 4$. Recall that when a bucket B (with local depth L) overflows, a new bucket B' is created, where both B and B' have the same incremented local depth value of L+1. The idea is to try to use the additional one bit to resolve the overflow problem. If the incremented local depth exceeds the global depth, then the global depth needs to be incremented (i.e., the directory needs to be doubled). Therefore, for the global depth to increase from 0 to 3, there must be at least 3 new buckets added to $I'$. Since $I$ originally has 1 bucket, the minimum number of buckets in $I'$ is 4.
**9B.** $E_{max} = 2$. Recall that if an insertion causes a bucket B to overflow, and the overflow is resolved after the creation of its corresponding bucket B', then both $B$ and $B'$ (both with incremented local depth of L+1) must be non-empty. On the other hand, if all the entries in the overflowed bucket $B$ still have the same last $L + 1$ bits, then the newly created bucket $B'$ will be empty and $B$ will have to be further splitted. Thus, for an empty bucket to be created in $I'$, there must exist an overflow situation that requires a sequence of at least 2 new buckets to be created to resolve the overflow such that all the newly created buckets in the sequence, except for the last one, are empty. Given that the global depth is increased from 0 to 3, at most 3 new buckets can be created to resolve the overflow due to a new insertion. Moreover, there can be at most one insertion into $I$ that requires the creation of 3 new buckets to resolve the overflow; and it is impossible to have more than two insertions into $I$ where each of them requires the creation of 2 new buckets to resolve the overflow. Therefore, the maximum number of empty buckets in $I'$ is 2.
**9C.** $B_{max} = 5$. Since the global depth has a value of 3, a bucket with a local depth of 1 in $I'$ has 4 directory entries pointing to it. To maximize the number of data buckets in $I'$, the remaining 4 directory entries in $I'$ must each be pointing to a different bucket. So the maximum number of data buckets is 5.
**10.** v = 0
**11.** False. We show that the claim is false by illustrating with the following example of $I$. Consider $I$ with the following properties: (1) $x*$ will be inserted into a bucket $B_x$ in $I$, (2) $y*$ will be inserted into a bucket $B_y$ in $I$, (3) both $B_x$ and $B_y$ are full, and (4) **next** is currently pointing to bucket $B_y$ which immediately precedes bucket $B_x$.

First, consider the insertions to create $I_{xy}$. The insertion of $x*$ into $I$ will overflow $B_x$. Since $B_y$ is the next bucket to be split, $x*$ is stored in an overflow page in $B_x$, and a bucket split is triggered for $B_y$.

Assume that the split of $B_y$ results in $B_y$ becoming not full after the split. **next** is updated to point to $B_x$. When $y*$ is inserted into $I_x$, the insertion does not overflow $B_y$.

Next, consider the insertions to create $I_{yx}$. The insertion of $y*$ into $I$ will overflow $B_y$. Since $B_y$ is the next bucket to be split, this causes $B_y$ to be split, and **next** is updated to point to $B_x$. When $x*$ is inserted into $I_y$, $x*$ is inserted into $B_x$ which overflows. Since $B_x$ is the next bucket to be split, this causes $B_x$ to be split. Assume that the split of $B_x$ resolves the overflow without creating any overflow page. Thus, $x*$ is not stored in an overflow page in $I_{yx}$.