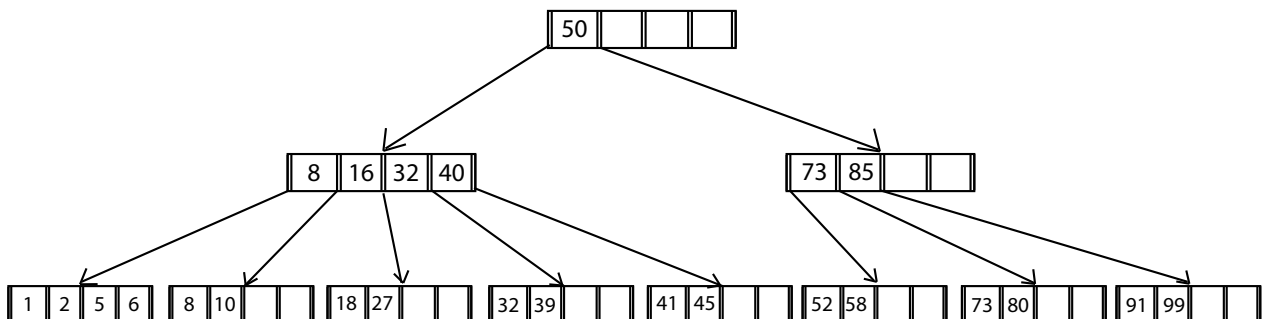


## Problem 1

Perform each of the following operations. For each problem, start with the B+ tree below:

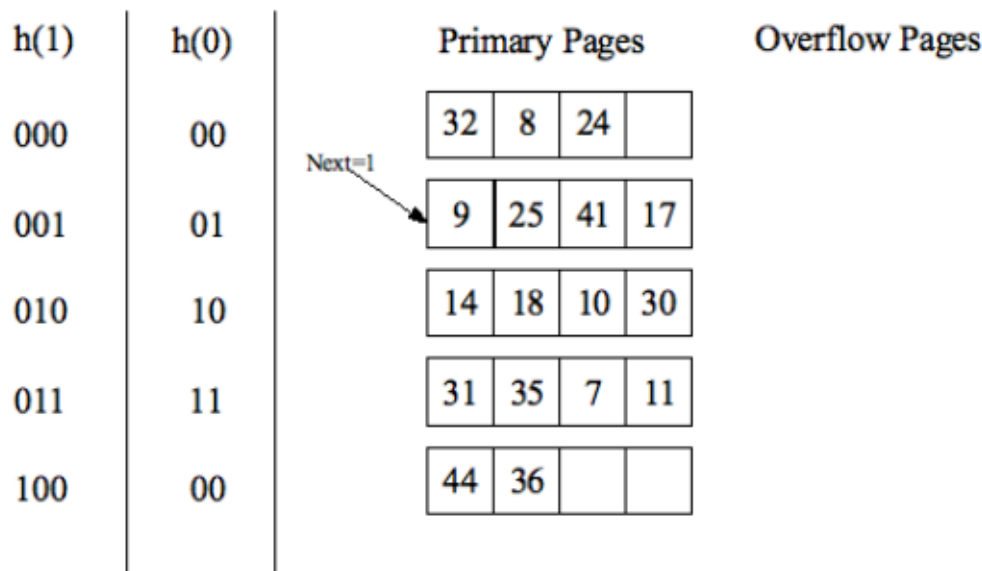
1. Show the tree that would result from inserting 9 into the tree.
2. Show the tree that would result from deleting 8 from the tree, assuming that the left sibling is checked for possible redistribution.
3. Show the tree that would result from inserting 3 into the tree.
4. Show the tree that would result from inserting 46 then deleting 52.
5. Show the tree that would result after the following sequence of deletions: 32, 39, 41, 45, 73.



## Problem 2

Consider the Linear Hashing index shown in below. Assume that we split whenever an overflow page is created. Answer the following questions about this index:

1. What can you say about the last entry that was inserted into the index?
2. Show the index after inserting an entry with hash value 4.
3. Show the original index after inserting an entry with hash value 15.
4. Show the original index after deleting entries with hash values 36 and 44.



### Problem 3

Lord Stan calls in the development team for his new database product, sbzDB, for a check on the real performance. sbzDB implements either extendible hashing or a B tree index and his engineers have tested it on the same datasets. We expect your solution will require a few sentences.

When answering, assume

- Both indexes are secondary indexes. So, leaf nodes (buckets) contain only keys/pointers and "index size" does not include disk space consumed by primary pages to store the tuple data.
- The root node (directory) must fit in memory.

	$10^4$	$10^5$	$10^6$	$10^7$	$10^8$	$10^9$
(a) B tree query time (ms)	100	100	150	150	200	200
E. hashing query time (ms)	50	50	50	50	50	N/A

For each index, the engineers performed a query on the indexed key that returns a single tuple, for example "SELECT \* FROM x WHERE id=100". Explain each of the following: (1) why B tree has a greater access time overall, (2) what happened to the access time of B tree at  $10^6$

and  $10^8$  tuples, (3) why the access time of extendible hashing is constant, and (4) what happened to extendible hashing index at  $10^9 \sim$  tuples.

- (b) Let  $n$  be the number of tuples,  $k$  be the maximum number of key-pointer pairs a B tree node can contain. Assume the B tree is completely full in every node. Express the following mathematical expressions in terms of  $n$  and  $k$ :
1. B tree depth (use floor or ceiling)
  2. Total number of nodes, including root/intermediate/leaf nodes (Do not leave  $\sum$  in the answer)
  3. Assuming  $k \gg 1$ , what can we say about the size of the B tree index in terms of  $n$ ? (Assume every node has the same size on disk)
- (c) In the same way, let  $n$  be the number of tuples,  $k$  be the maximum number of key-pointer pairs a bucket can contain in an Extendible Hashing index. Assume every bucket is full and we have a perfect hash function which has no skew. Express the following mathematical expressions in terms of  $n$  and  $k$ :
1. Number of buckets
  2. Global depth of directory (use floor or ceiling function)
  3. Memory consumption to keep the dictionary on memory (Assume one dictionary entry consumes  $b$  bytes of RAM)

## Problem 4

- (a) What is a lower bound on the percentage space utilization in intermediate (not leaf nor root) nodes of a B tree? In other words, what is the minimum amount of the allocated space that might be in use? (Assume a B tree in a valid state. Some DBMS allows invalid state for write-intensive situation, but it's not allowed here. )
- (b) Compare a B tree implementation to a sorted list data structure, which stores the pairs of a key and a pointer to primary page sorted by key and nothing else. Mathematically derive how much more space it takes for a B tree than a sorted list, supposing a B tree that is completely full to some depth. You may assume that keys and pointers the same size. Again, these are secondary indexes. Disk space consumed by primary data pages is ignored. Approximation or bounding is acceptable.
- (c) Note that a sorted list data structure also has  $\log(n)$  search time using binary search. However, almost all DBMS employ B tree instead of sorted list despite the wasted space. Discuss the reason, referring to the following points. (1) Query (SELECT) cost (2) Maintenance (INSERT/DELETE/UPDATE) cost