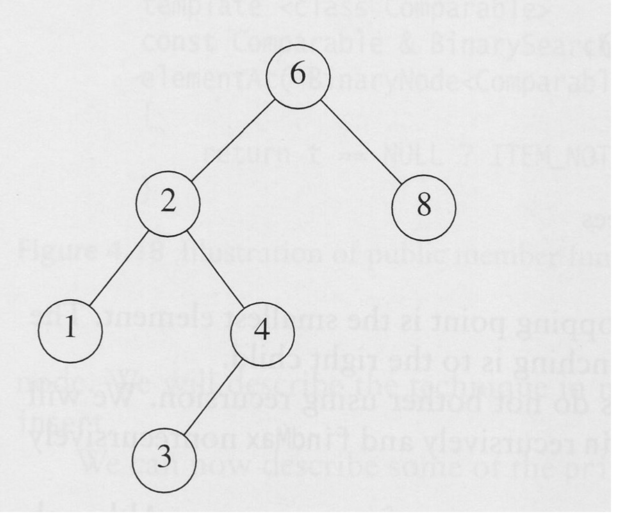
Tree Traversals

- Print the content of a BST in sorted order => inorder traversal

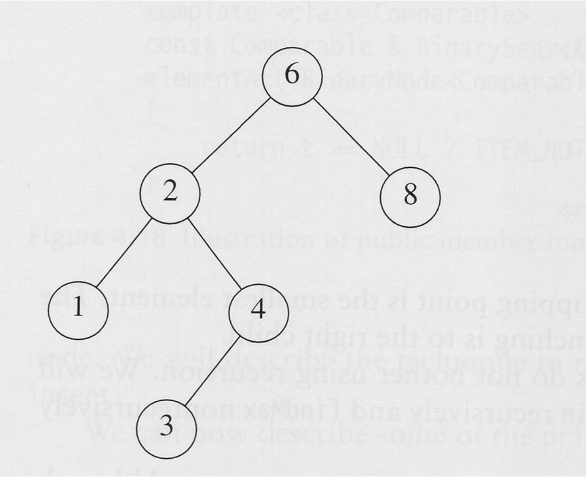
```
void printTree( ostream & out = cout ) const
{
        if( isEmpty( ) )
          cout << "Empty tree" << endl;
        else
          printTree( root, out );
}

/* Internal method */
void printTree( BinaryNode *t, ostream & out ) const
{
        if( t  != NULL )
        {
          printTree( t ->left, out );
          cout << t -> element << endl;
          printTree( t -> right, out );
        }                                    T(N) = ?
}
```



Tree Traversals

- Compute the height of a tree
  => postorder traversal




```
/* Internal method */
int height( BinaryNode *t)
{
      if( t  == NULL )
       return -1;
      else
       return 1 + max( height(t -> left), height( t ->right ))
}
```

T(N) = ?

B - Trees

- We assumed that the entire data structure fits in the main memory.
- What if some of the data resides on disk ?

  => Big-Oh does not work
- Big-Oh assumes that all operations take equal amount of time.
- Example:

  ~100,000 MIPS CPU (Intel Core i7, 4C, 3.4 GHz)

  => 100 billion instructions per second

  Disk: 7200 rpm => one revolution per 1/120 sec. => one revolution takes 8.3 ms => average time = 8.3 ms

  => 120 disk accesses/sec.

  One disk access => ~1 billion instructions

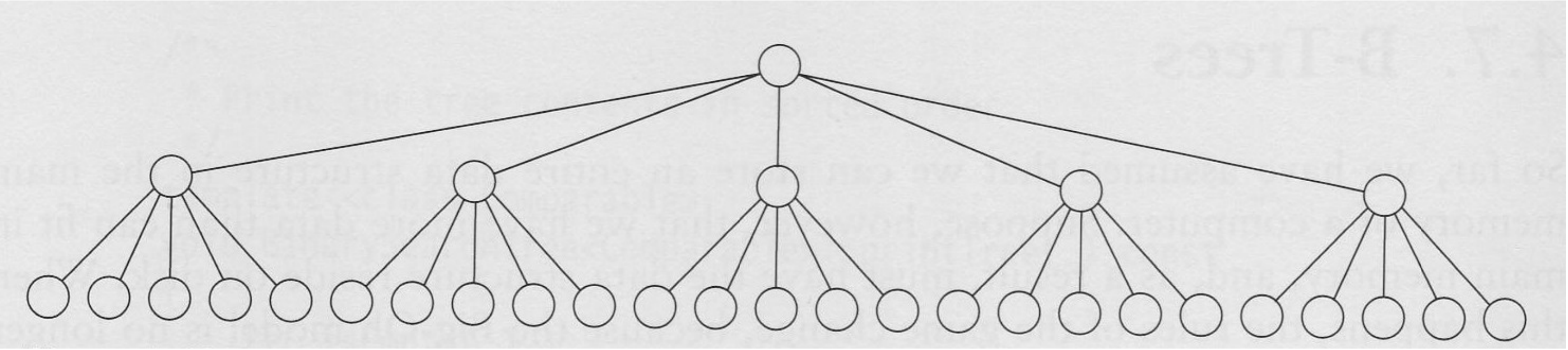
  The number of accesses will dominate the running time.

Typical Search Tree on Disk

- Florida driving records = 10,000,000 items
- Each key (name) = 32 B, record =256 B
- Approx 2.3 GB, assume that it does not fit in memory.
- There are 20 users in the system => 1/20 of resources => in 1 sec: 5 billion instructions or 6 disk accesses.
- Unbalanced BST: 10,000,000 disk accesses => 1.6 million sec. (about 19 days!)
- Average BST: 1.38 log N = 32 disk accesses => 5 sec.
- AVL Tree: average case log N = 25 disk accesses => 4 sec.
- Problem: We cannot go below log N using BST!
- Solution ? => more branching reduces height

M-ary Search Tree

- Allows M-way branching => height = $\log_M N$
- Example: 5-ary tree of 31 nodes
- BST of 31 nodes => 5 levels
- 5-ary tree => 3 levels
- BST: we need one key to decide which child to explore
- M-ary Tree: we need M-1 keys to decide which child to explore



B - Trees

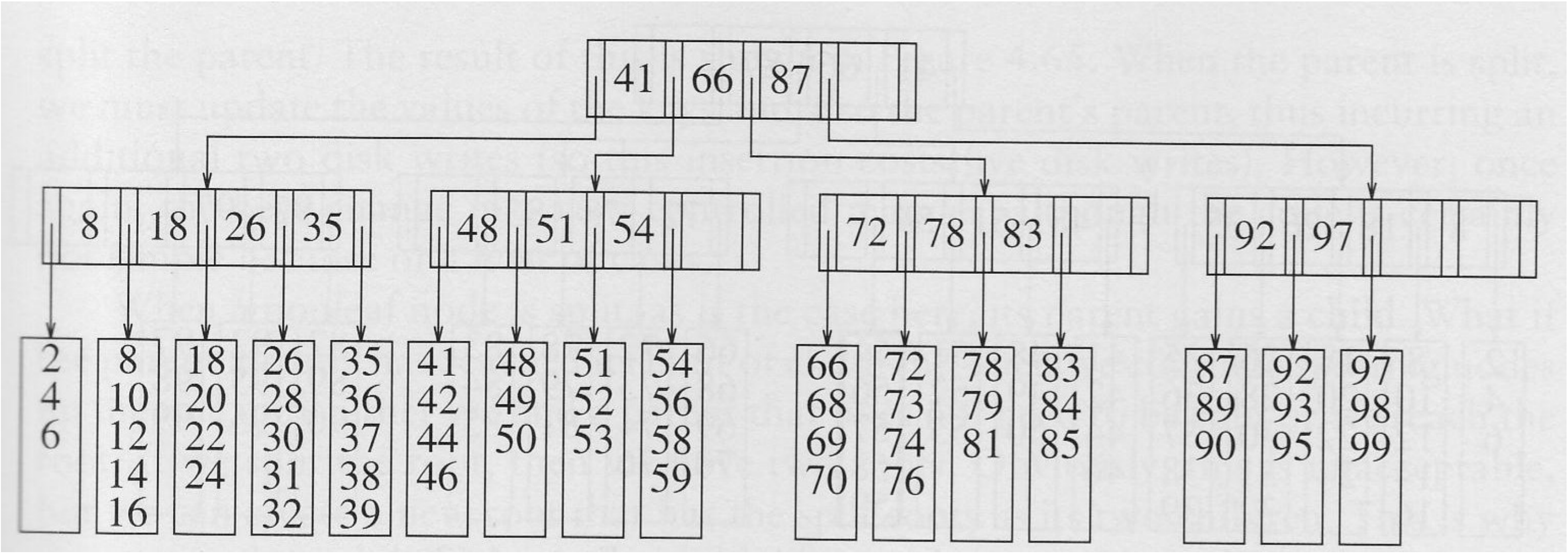
R. Bayer and E. McCreight (1972):

”Organization and Maintenance of Large Ordered Indexes”, Acta Informatica 1 (3): 173–189, 1972

A B – Tree of order M is an M-ary tree with the following properties:

- The data items are stored at leaves.
- The non-leaf nodes store up to M-1 keys to guide the searching; key i represents the smallest key in subtree i+1.
- The root is either a leaf or has between 2 and M children.
- All non-leaf nodes (except the root) have between $\lceil M/2 \rceil$ and M children.
- All leaves are at the same depth and have between $\lceil L/2 \rceil$ and L data items, for some L.

B – Tree Example



- M = 5, L = 5
- Nodes need to be half full => guarantees that the B-tree does not degenerate into a binary tree.

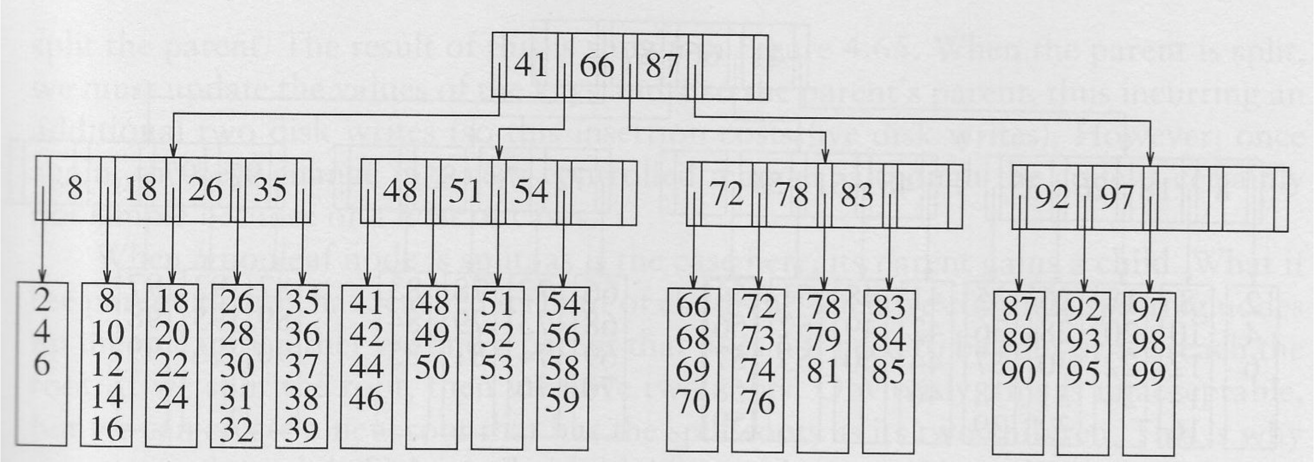
B - Trees

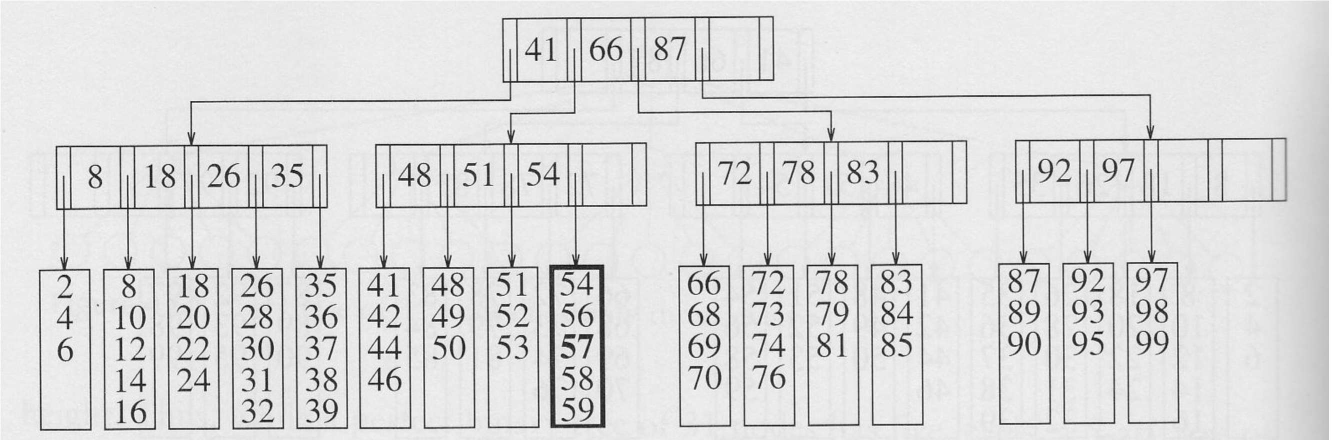
- Each node = disk block
- How to select M and L?
- Example:
  - Disk block = 8192 Bytes
  - key = 32 Bytes => Size of nonleaf node = 32(M-1) + 4M =

    = 36 M - 32 Bytes
  - Pick largest M such that 36M – 32 <= 8192 => M = 228
  - Data record = 256 Bytes => 32 records/block => L = 32.
  - Each leaf has between 16 and 32 records => at most 625,000 leaves.
  - Leaves are at level 4 (worst case) => 3 disk accesses if we cache the root.
- Worst-case number of accesses = $O(\log_{M/2} N)$

Insertion

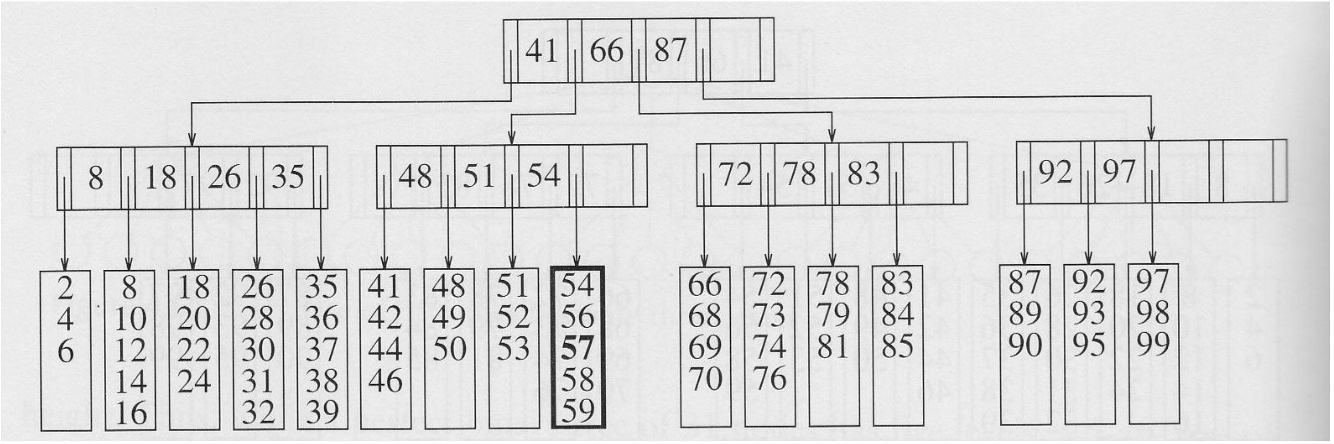
Insert 57

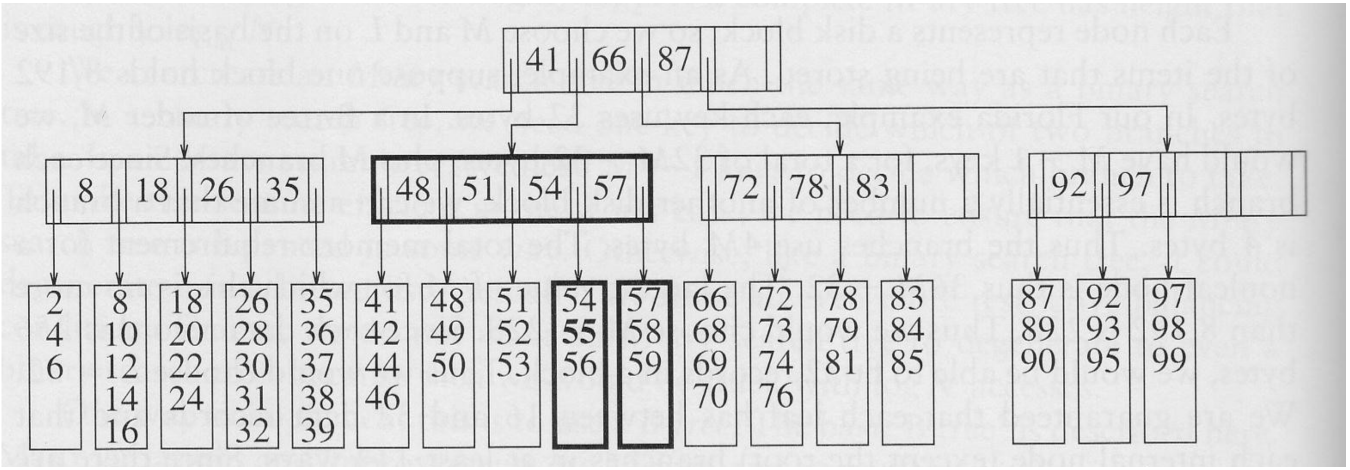




Insertion

Insert 55



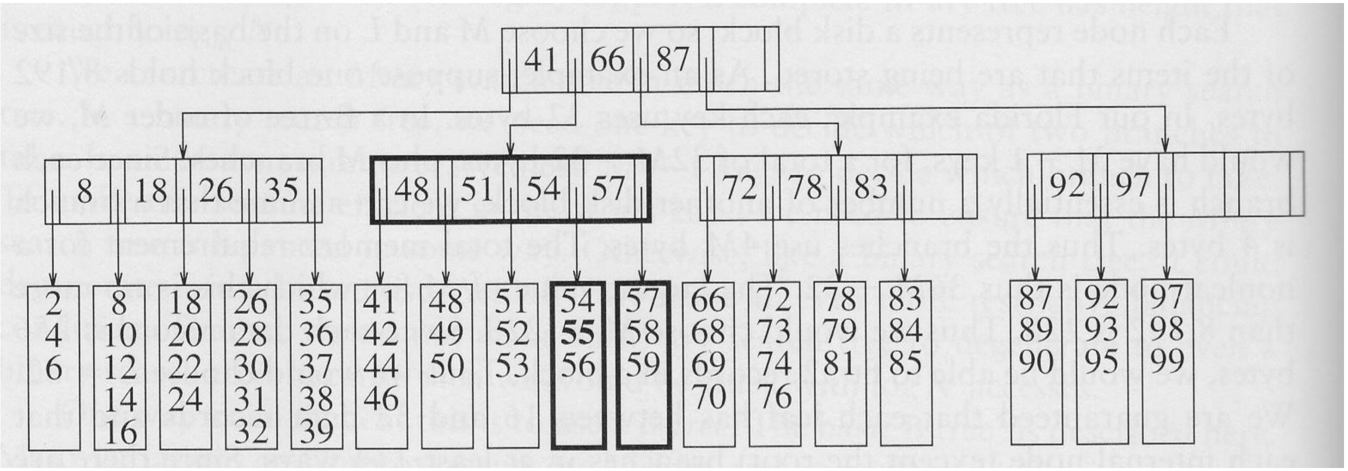


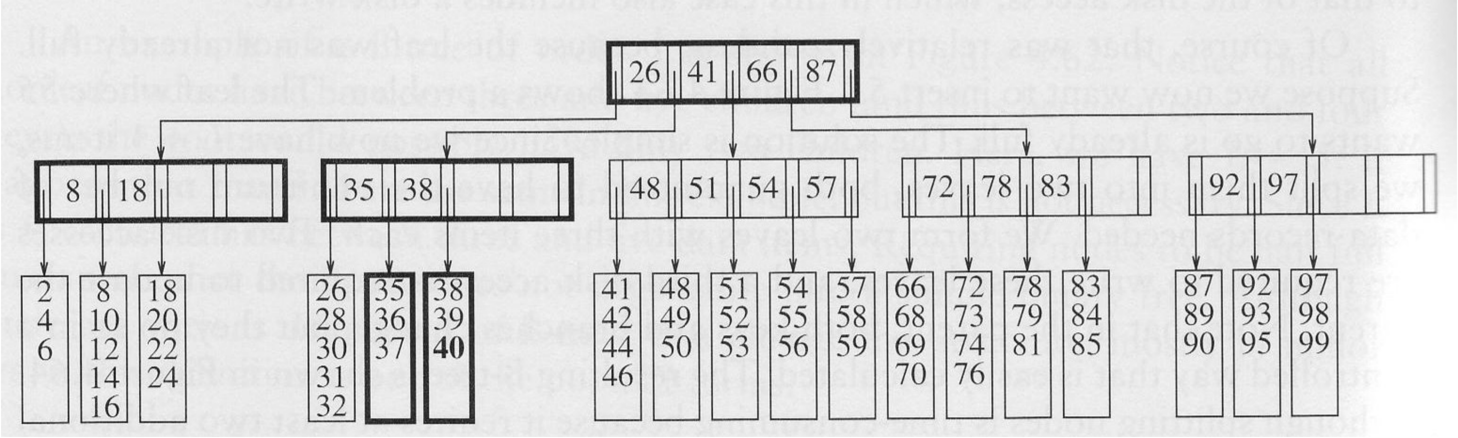
3 additional accesses for updating

Splitting is rare.

Insertion

Insert 40

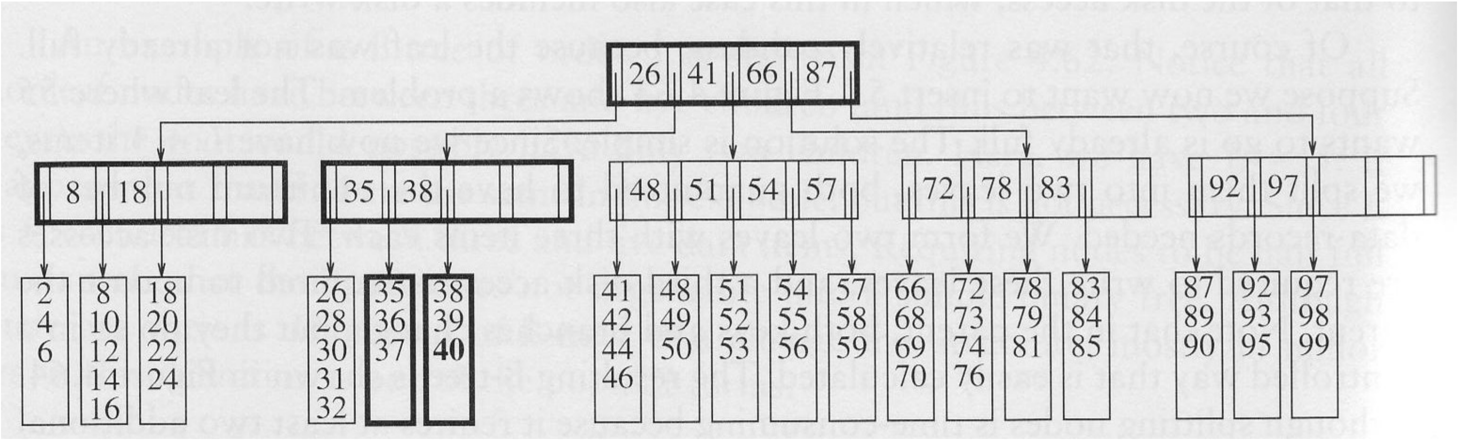


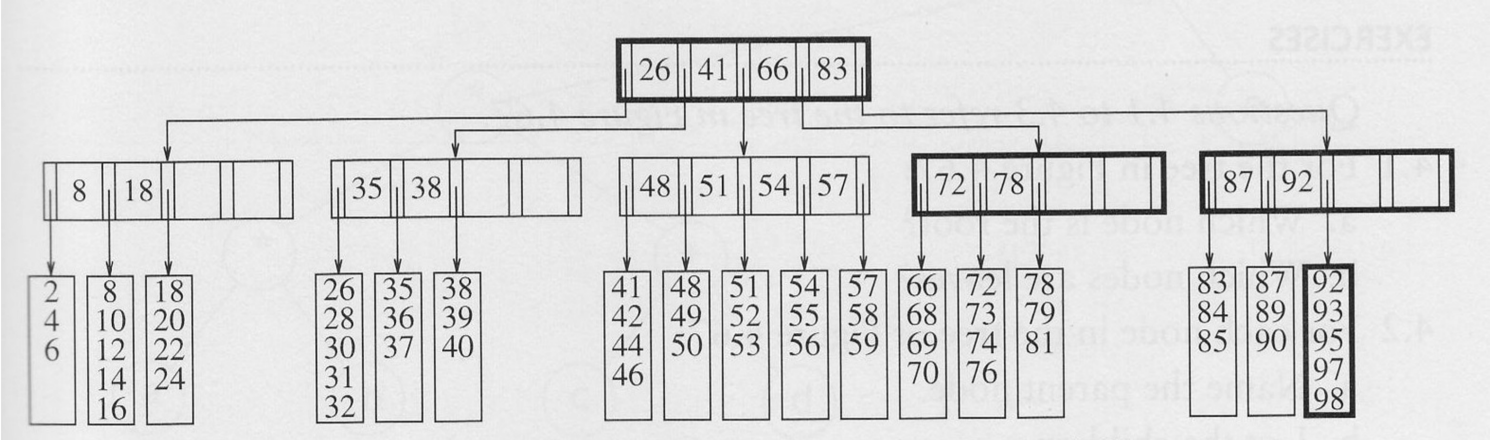


5 additional accesses for updating

Deletion

Delete 99





4 additional accesses for updating

Tree Applications

- Operating systems (file systems)
- Compilers (parse trees)
- Data bases (search trees)
- Sorting algorithm: insert items in a BST and then perform an inorder traversal => $O(N \log N)$