Indexing

Goals:

- Store large files.
- Support multiple search keys.
- Support efficient insert, delete and range queries.

Entry sequenced file: Order records by time of insertion.

Use sequential search.

Index file: Organized, stores pointers to actual records.

Primary key: A unique identifier for records. May be inconvenient for search.

Secondary key: an alternate search key, often not unique for each record. Often used for search key.

Linear Indexing

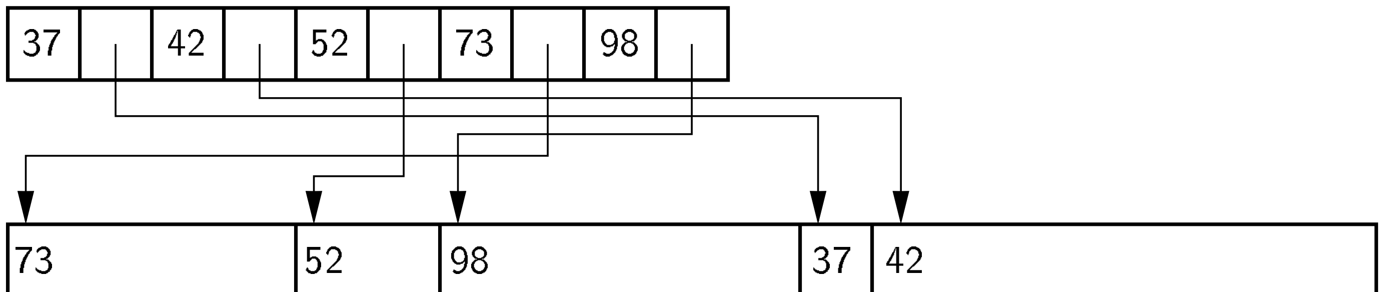
Linear Index: an index file organized as a simple sequence of key/record pointer pairs where the key values are in sorted order.

If the index is too large to fit in main memory, a second level index may be used.

Linear indexing is good for searching variable length records.

Linear indexing is poor for insert/delete.

Linear Index



Database Records

1	2003	5894	10528

Second Level Index

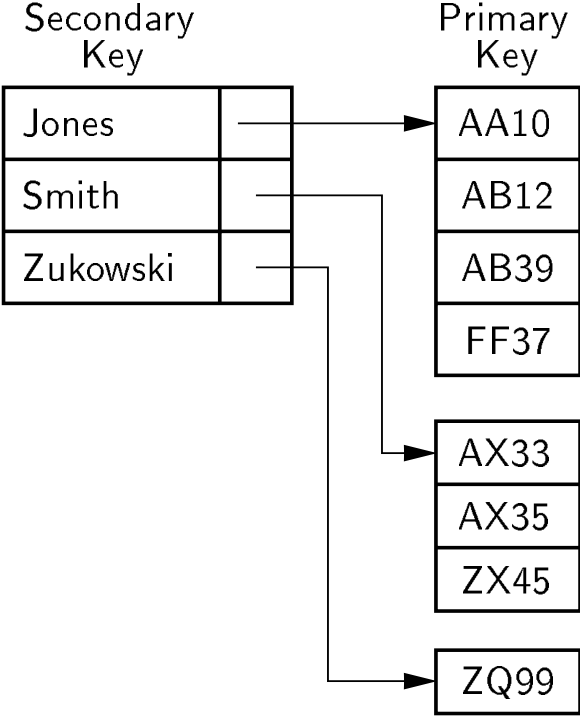
1 2001	2003 5688	5894 9942	10528 10984

Linear Index: Disk Pages

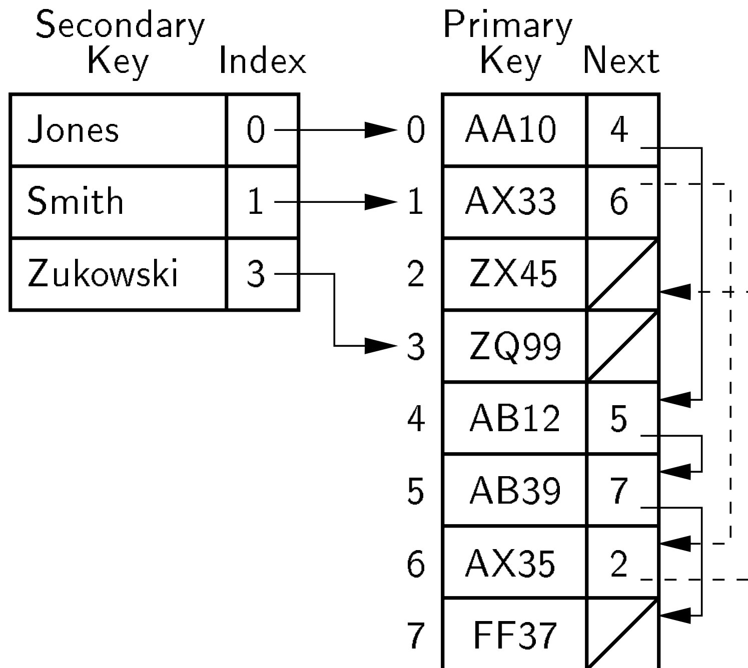
Inverted List

Inverted list is another term for a secondary index. A secondary key is associated with a primary key, which in turn locates the record.

Jones	AA10	AB12	AB39	FF37
Smith	AX33	AX35	ZX45	
Zukowski	ZQ99			



Inverted List (Continued)



Tree Indexing

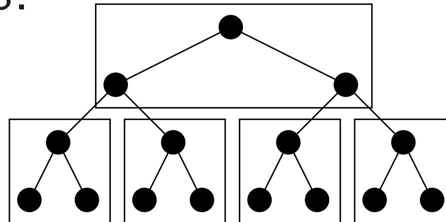
Linear index is poor for insertion/deletion.

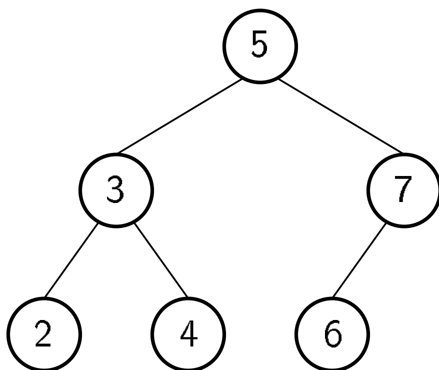
Tree index can efficiently support all desired operations:

- Insert/delete
- Multiple search keys
- Key range search

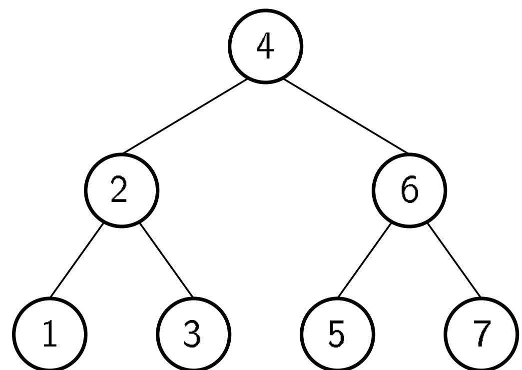
Storing a tree index on disk causes additional problems:

1. Tree must be balanced.
2. Each path from root to a leaf should cover few disk pages.





(a)



(b)

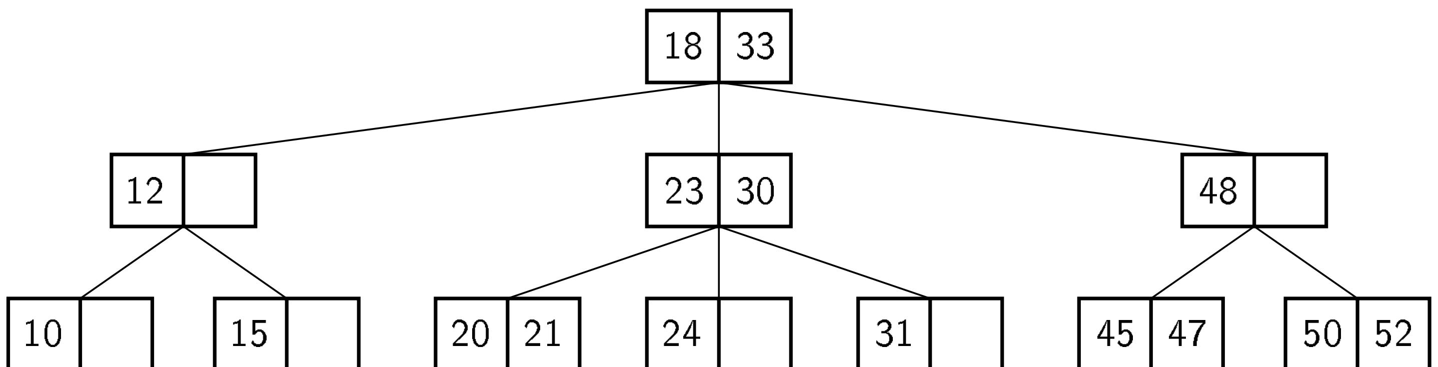
2-3 Tree

A 2-3 Tree has the following properties:

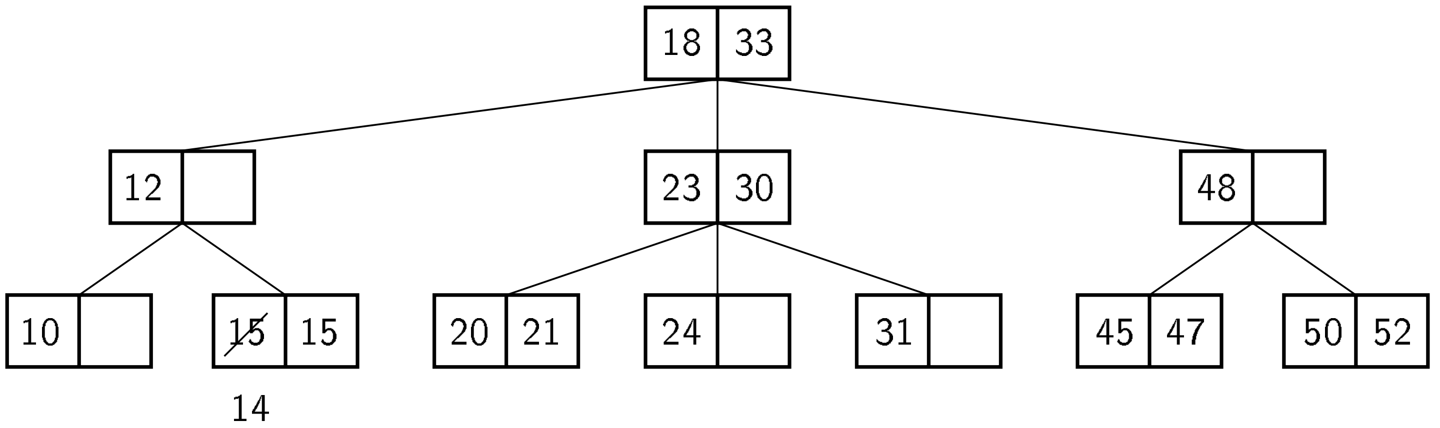
1. A node contains one or two keys.
2. Every internal node has either two children (if it contains one key) or three children (if it contains two keys).
3. All leaves are at the same level in the tree, so the tree is always height balanced.

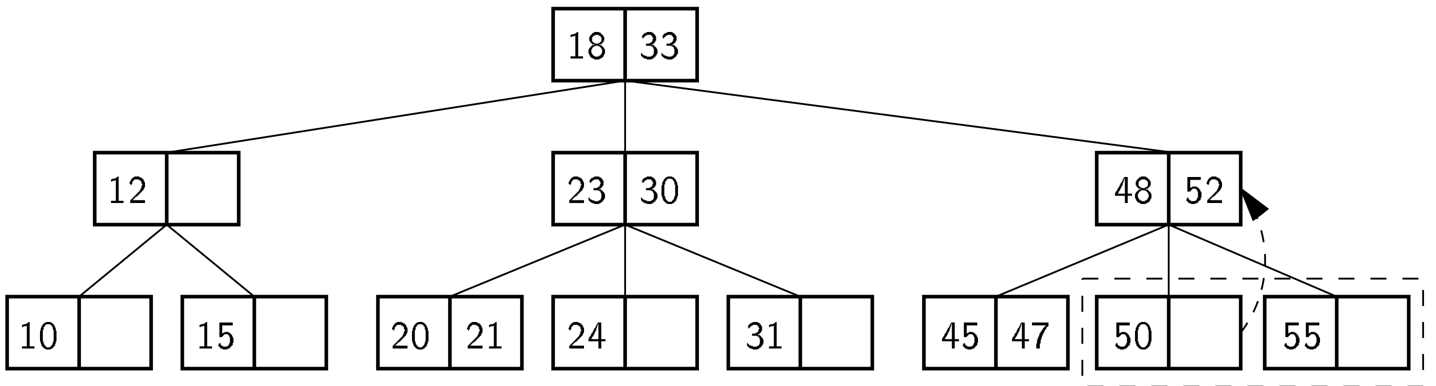
The 2-3 Tree also has a search tree property analogous to the BST.

The advantage of the 2-3 Treeover the BST is that it can be updated at low cost.

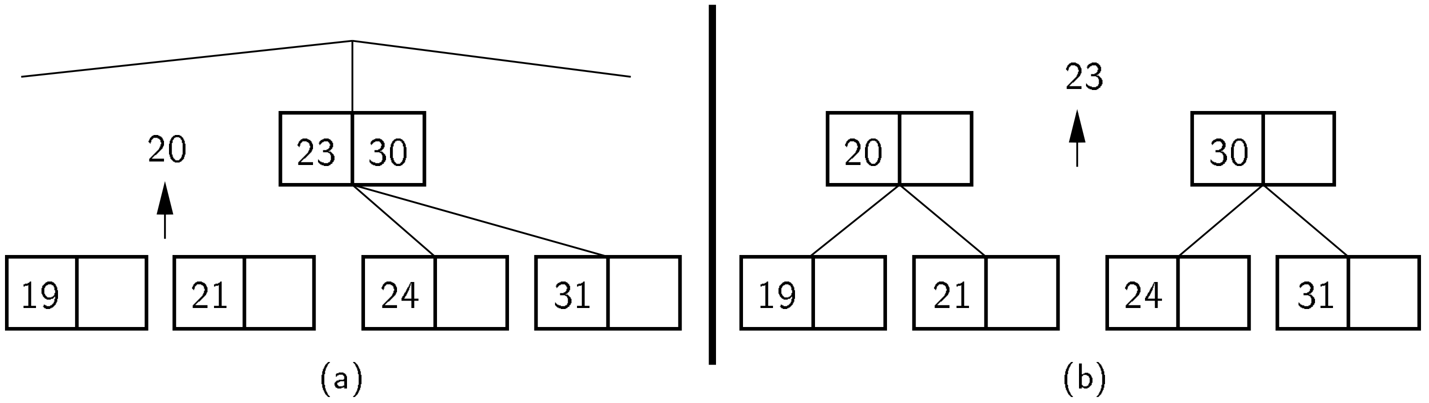


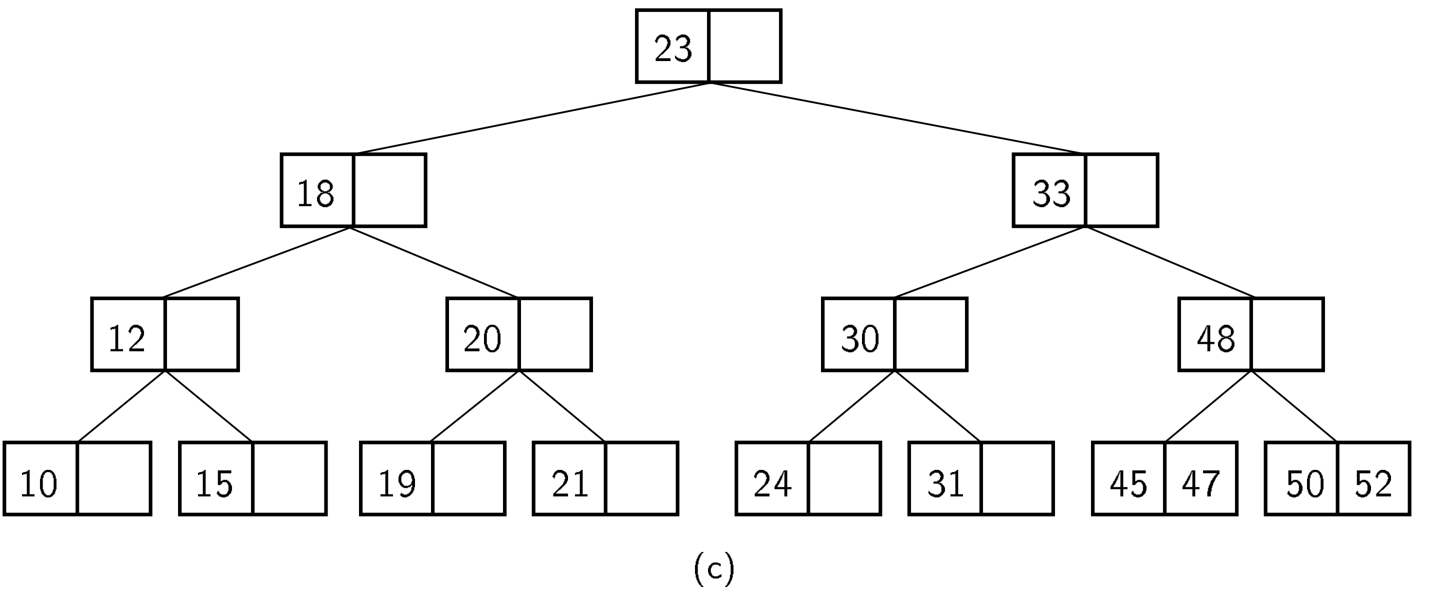
2-3 Tree Insertion





2-3 Tree Splitting





B-Trees

The B-Tree is an extension of the 2-3 Tree.

The B-Tree is now **the** standard file organization for applications requiring insertion, deletion and key range searches.

1. B-Trees are always balanced.
2. B-Trees keep related records on a disk page, which takes advantage of locality of reference.
3. B-Trees guarantee that every node in the tree will be full at least to a certain minimum percentage. This improves space efficiency while reducing the typical number of disk fetches necessary during a search or update operation.

B-Trees (Continued)

A B-Tree of order m has the following properties.

- The root is either a leaf or has at least two children.
- Each node, except for the root and the leaves, has between $\lceil m/2 \rceil$ and m children.
- All leaves are at the same level in the tree, so the tree is always height balanced.

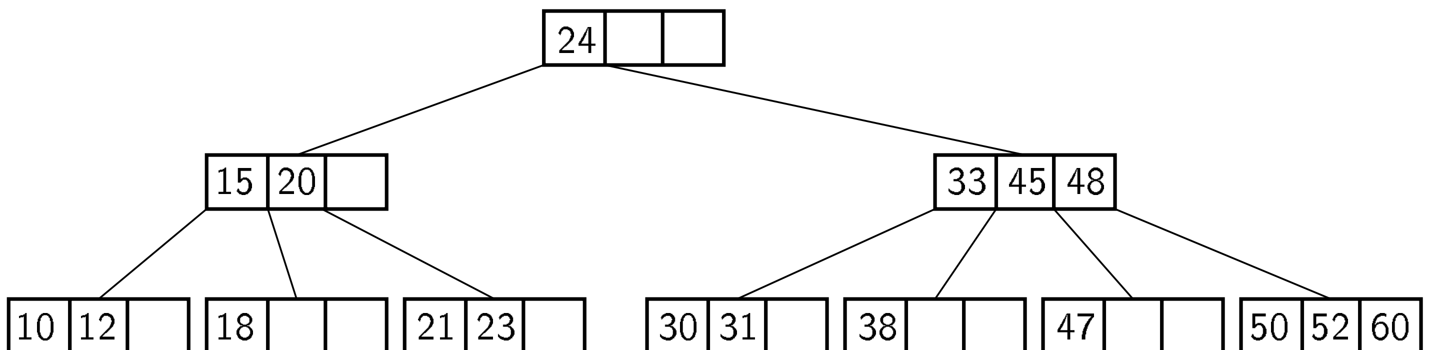
A B-Tree node is usually selected to match the size of a disk block.

A B-Tree node could have hundreds of children.

B-Tree Example

Search in a B-Tree is a generalization of search in a 2-3 Tree.

1. Perform a binary search on the keys in the current node. If the search key is found, then return the record. If the current node is a leaf node and the key is not found, then report an unsuccessful search.
2. Otherwise, follow the proper branch and repeat the process.



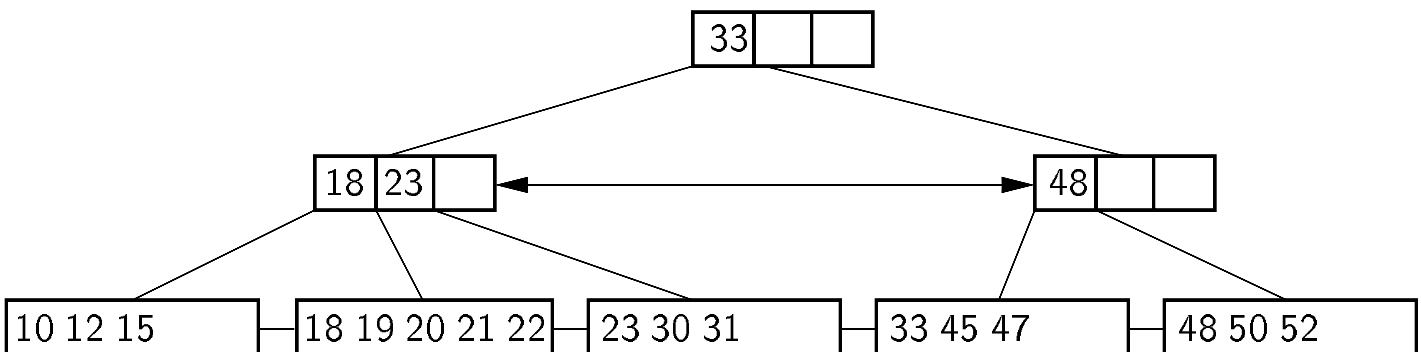
B$^+$-Trees

The most commonly implemented form of the B-Tree is the B$^+$-Tree.

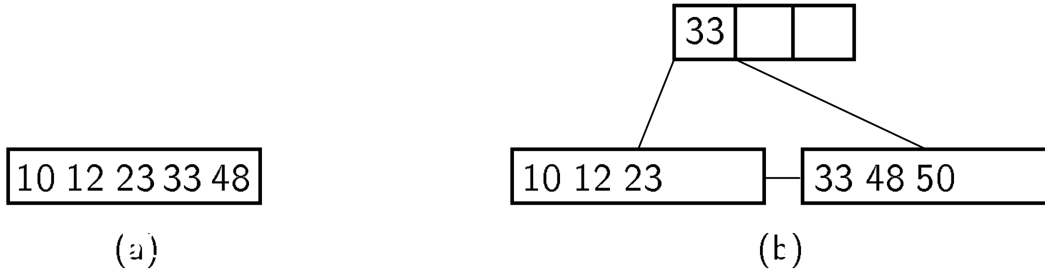
Internal nodes of the B$^+$-Tree do not store records – only key values to guide the search.

Leaf nodes store records or pointers to records.

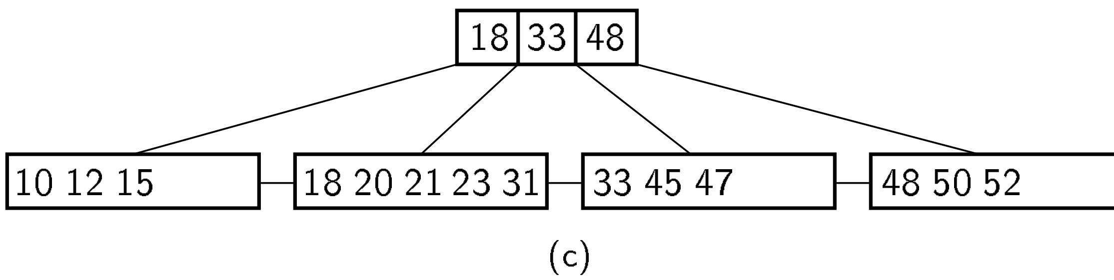
A leaf node may store more or less records than an internal node stores keys.



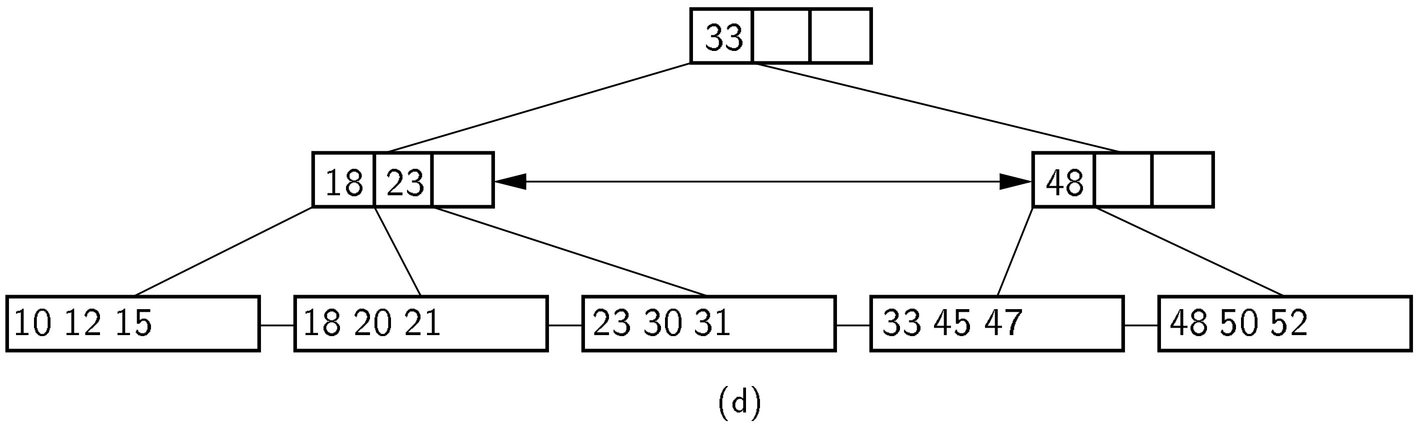
B^+-Tree Insertion



(a) (b)

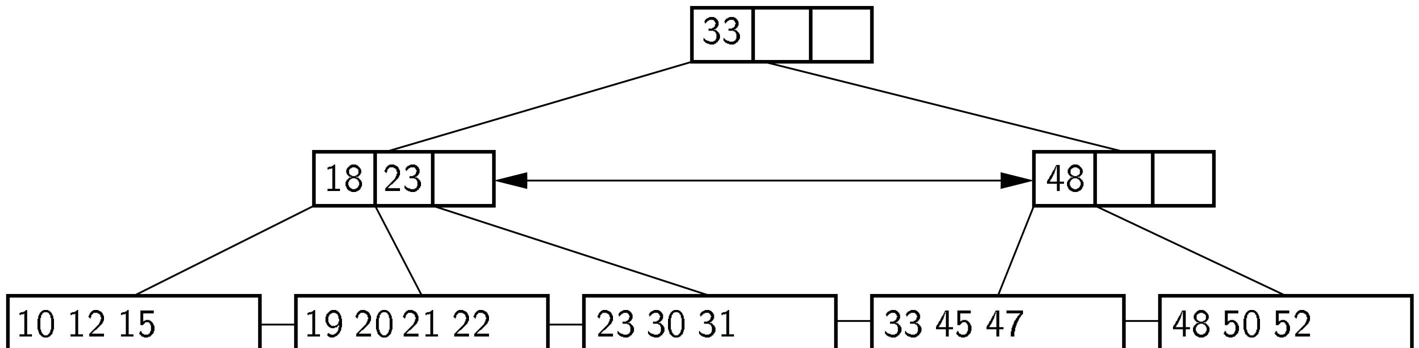


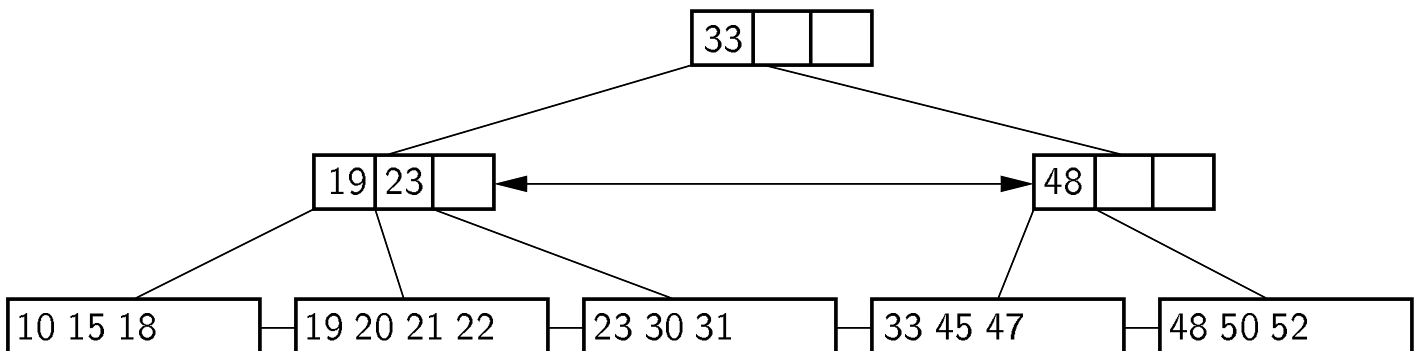
(c)



(d)

B$^+$-Tree Deletion





B-Tree Space Analysis

B^+-Tree nodes are always at least half full.

The B*-Tree splits two pages for three, and combines three pages into two. In this way, nodes are always 2/3 full.

Asymptotic cost of search, insertion and deletion of records from B-Trees, B^+-Trees and B*-Trees is $\Theta(\log n)$. (The base of the log is the (average) branching factor of the tree.)

Example: Consider a B^+-Tree of order 100 with leaf nodes containing 100 records.
1 level B^+-Tree:

2 level B^+-Tree:

3 level B^+-Tree:

4 level B^+-Tree:

Ways to reduce the number of disk fetches:

- Keep the upper levels in memory.
- Manage B^+-Tree pages with a buffer pool.