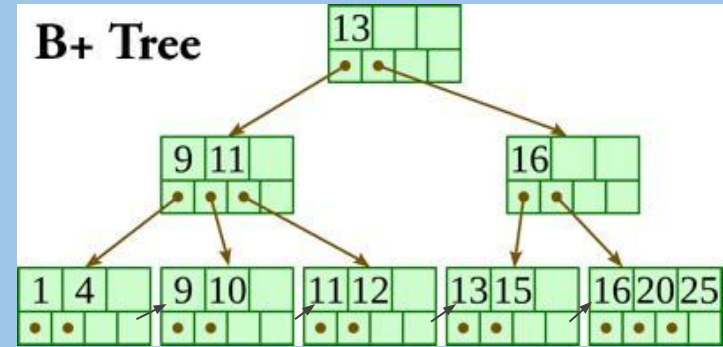# B+ tree:

B tree and beyond

By - Thomson Kneeland

Implementation of B+ Tree

Thomson Kneeland - coding implementation, author

Khushbu Rana - report and slides, author


```
42      //method to check if Page contains key
43      public boolean containsKey(int x){
44          boolean keyInArray=false;//variable to monitor whether key is in array
45          for (int num: keys){
46              if (num==x){
47                  keyInArray= true;
48              }
49          }
50          return keyInArray;
51      }
```

How do we organize databases?


Given a large number of files:

- How to organize the files?
- How to search and retrieve a file?
- How to easily store new files?
- Do all the above efficiently in terms of time and space?

We have a limited CPU memory and databases of millions of records...

B tree

- B tree is one of the data structures that are well organized in insertion and search operation
- Also known as Balanced tree
- The procedure of Insertion and Deletion operation are easy
- Has root, internal nodes, and leaf nodes
- Given minimum degree "t", each node has between t and 2t-1 keys; when full at 2t, it splits into a new node
- Has data pointers in internal nodes to max of 2t children
- All leaves are at the same level/depth, no pointers

Time complexity

Space - O(n)

Search - O(log n)

Insert - O(log n)
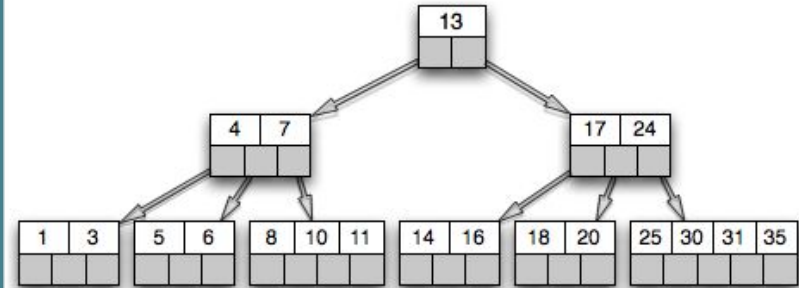
Delete - O(log n)

n = # of keys

History of B Trees

Rudolf Bayer

Bayer and McCreight at Boeing in the 1960s implemented the B Tree in their paper *Organization and Maintenance of Large Ordered Indexes*

Origin of name? "B" for Boeing, "B" for balanced (McReight, 2013)

Bayer also invented the Red-Black Tree

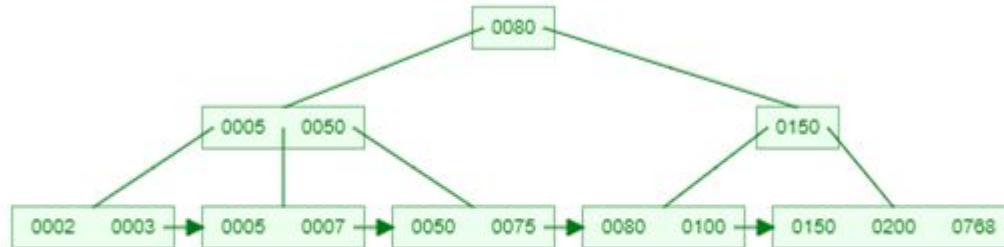One of the most important data structures in modern computer science.

Still used in many databases, file systems, and information retrieval applications
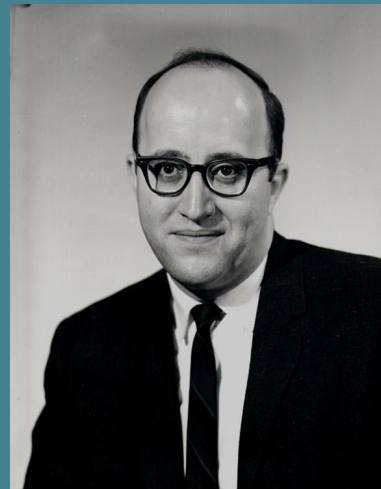
B+ Trees

- B+ Tree was the next major development in efficiency in the 1970's

- No single "inventor", but was used at IBM as early as 1973

- B Trees and B+ Trees both excellent for retrieval of contiguous blocks of data

B+ tree

- "Descendant "of B Tree; same attributes

- Stores all data on leaf nodes

- Internal/parent nodes are "decision makers" to point towards children; database index

- Order of a B+ Tree = the number of children nodes in each parent node

- Since all keys are stored in the leaf nodes, leaf nodes are ordered as linked-list with pointer

- This makes B+ Tree good for "range queries"
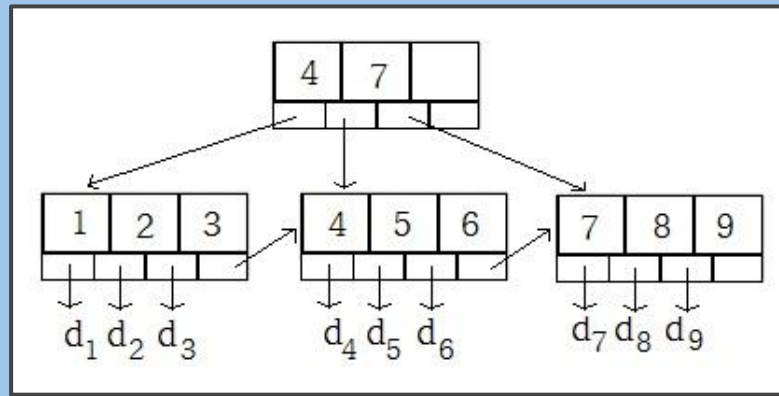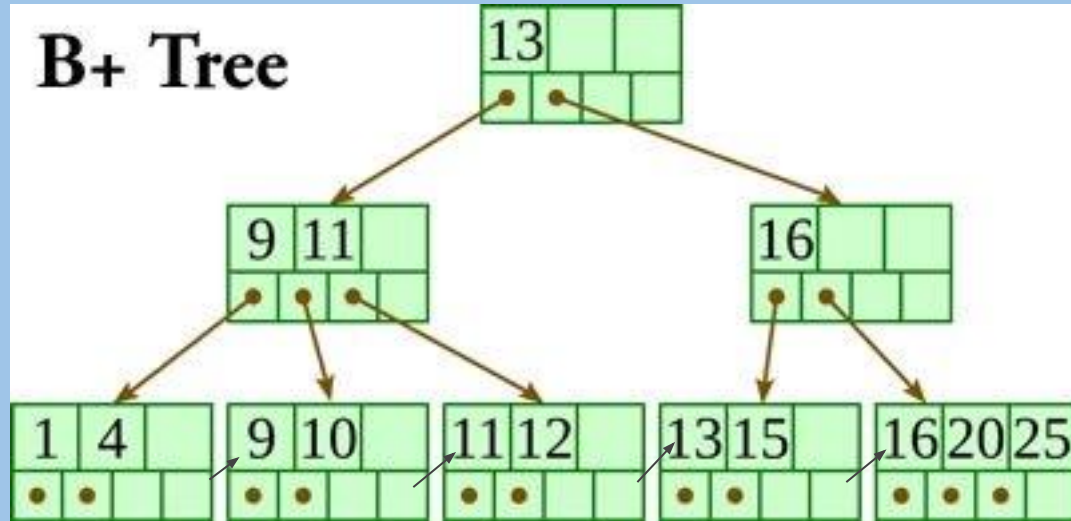
Time complexity

Space - O(n)

Search - O($\log_b n$)

Insert - O($\log_b n$)

Delete - O($\log_b n$)

(b = order of tree, n = # nodes)

Parts of a B+ Tree with order "4"

B+ Tree methods

Interface methods for user:

- **Insert**
- **Delete**
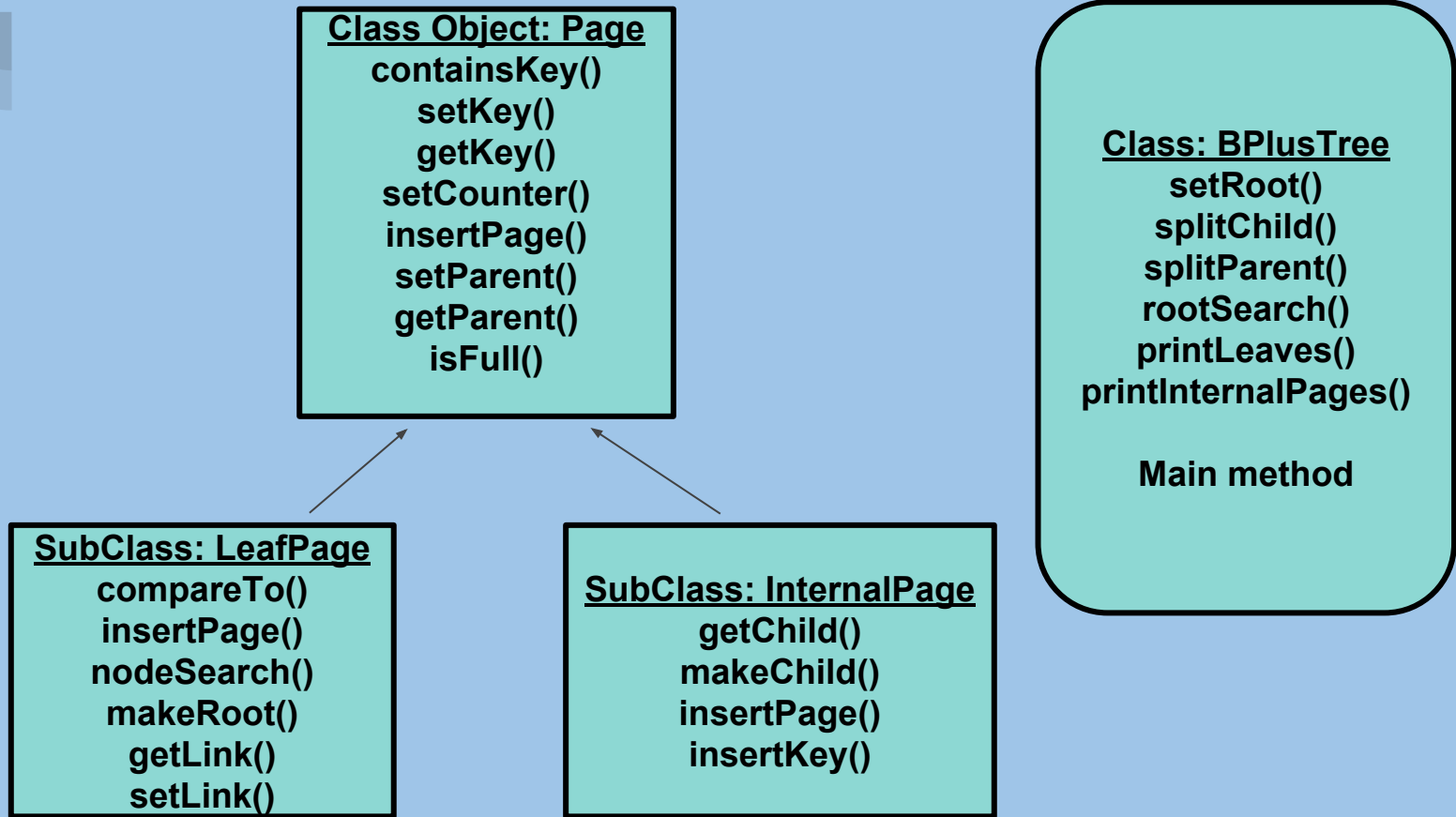- **Search/Retrieve**

Potential Internal methods for organization:

- **Split**
- **MakeNodeRoot**
- **GetParent**
- **GetChild**

Advanced for databases:

- **Join, etc**

Overall Code Schema

Class Object: Page
containsKey()
setKey()
getKey()
setCounter()
insertPage()
setParent()
getParent()
isFull()

Class: BPlusTree
setRoot()
splitChild()
splitParent()
rootSearch()
printLeaves()
printInternalPages()

Main method

SubClass: LeafPage
compareTo()
insertPage()
nodeSearch()
makeRoot()
getLink()
setLink()

SubClass: InternalPage
getChild()
makeChild()
insertPage()
insertKey()

```java
//Insert in final leaf node when found
public void InsertPage(int k){
    System.out.println("InsertPage "+k+": InternalPage");
    Arrays.sort(this.children,0,this.numChildren);
    //special instance : only one key in InternalPage
    if(counter==1){
        if(k<this.keys[0]){
            this.getChild(0).InsertPage(k);
        }else if (k>=this.keys[0]){
            this.getChild(1).InsertPage(k);
        }
    }else if (k<this.keys[0]){
    //if search path follows 1st keyif (k<this.keys[0]){
        this.getChild(0).InsertPage(k);
        System.out.println("path = first");
    }else if (k>=this.keys[counter-1]){//if search path follows last key
        this.getChild(counter-1).InsertPage(k);
        System.out.println("path = last");
    }else{
        for (int i=0;i<this.getCounter()-1;i++){//check middle terms of path
            if ((k>=this.keys[i]) && (k<this.keys[i+1])){
                System.out.println("path = middle");
                this.getChild(i).InsertPage(k);
                System.out.println(" Internal path2 =middle");
            }
        }
    }
}
//Arrays.sort(this.children,0,this.numChildren);
```

```java
20
21      //constructor
22      public BPlusTree(int t, int k){
23          order = t; //order of tree
24          isRootLeaf=true;//variable for determining which root to use
25          LeafPage x = new LeafPage(order, k);//create initial root node with value k
26          leaves.add(x);//add root to Linked List
27          this.setRoot(x);//make this node root
28          split = new int[order];//initialize array used for splitting
29          printLeaves();//print to console
30      }
31
32      //method for setting root
33      public void setRoot(LeafPage x){
34          leafRoot=x;
35      }
36
37      //method for setting root
38      public void setRoot(InternalPage x){
39          internalRoot=x;
40      }
41
```

INPUT EXAMPLE:

```
//       q.rootSearch(201);//WORKS


        //Test order 20, 1000 values
        BPlusTree q2 = new BPlusTree(20,1000);//good
        for (int i=1; i<1000;i++){
            q2.treeInsert(i);
            q2.printLeaves();
            q2.printInternalPages();
        }
    }
}
```

Order = 20, insert 1000 keys/files

OUTPUT:

Counter - # of keys:

Keys:

Children or parent pointers:

```
INTERNAL PAGES
*******************************
counter is: 18
pages key is 11
pages key is 21
pages key is 30
pages key is 41
pages key is 50
pages key is 61
pages key is 70
pages key is 81
pages key is 90
pages key is 101
pages key is 110
pages key is 121
pages key is 130
pages key is 141
pages key is 150
pages key is 161
pages key is 170
pages key is 181
pages key is 0
bt.LeafPage@6d06d69c
bt.LeafPage@70dea4e
bt.LeafPage@5c647e05
bt.LeafPage@33909752
bt.LeafPage@55f96302
bt.LeafPage@3d4eac69
bt.LeafPage@42a57993
bt.LeafPage@75b84c92
bt.LeafPage@6bc7c054
bt.LeafPage@232204a1
bt.LeafPage@4aa298b7
bt.LeafPage@7d4991ad
bt.LeafPage@28d93b30
```

```
path = last
LEAF PAGES
*******************************
counter is: 10
leaves key is 181
leaves key is 182
leaves key is 183
leaves key is 184
leaves key is 185
leaves key is 186
leaves key is 187
leaves key is 188
leaves key is 189
leaves key is 1000
leaves key is 0
leaves key is 0
leaves key is 0
leaves key is 0
leaves key is 0
leaves key is 0
leaves key is 0
leaves key is 0
leaves key is 0
bt.InternalPage@15db9742
*******************************
counter is: 10
```

Retrieve Key/Search for file


```
429
430    q.rootSearch(50);//works
431    q.rootSearch(200);//WORKS
432    q.rootSearch(201);//WORKS
433
```

```
Search for 50 complete:
Page is bt.LeafPage@7852e922, key index is: 0

Search for 200 complete:
Page is bt.LeafPage@7852e922, key index is: 2

Search for 201 complete:
key not found
```

Database with 200 keys, no 201!

Application of B+ tree

- Think of the B+ tree as an index that points to files

- Keeps "index file" in working memory which allows for easy retrieval of data from long term memory storage

- A file search for a key starts at the root and iterates down to the leaf nodes. The key in the leaf node points to the location of the file and retrieves it

- B+ Tree used in SQL Server, Oracle 8, IBM DB2 and other relational database systems
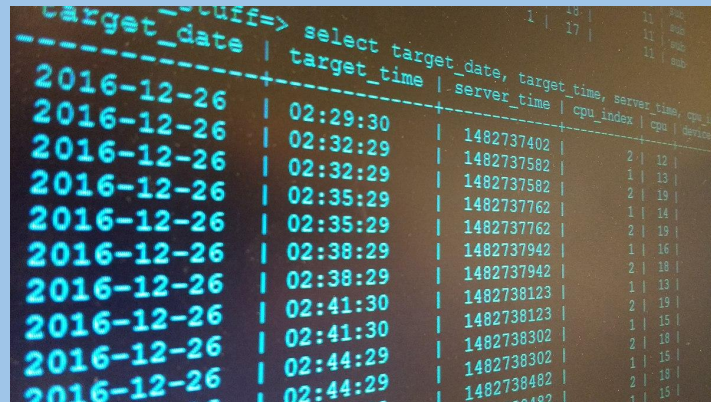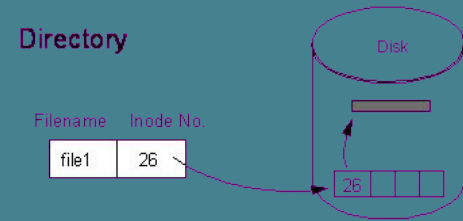
B+ Tree Demonstration
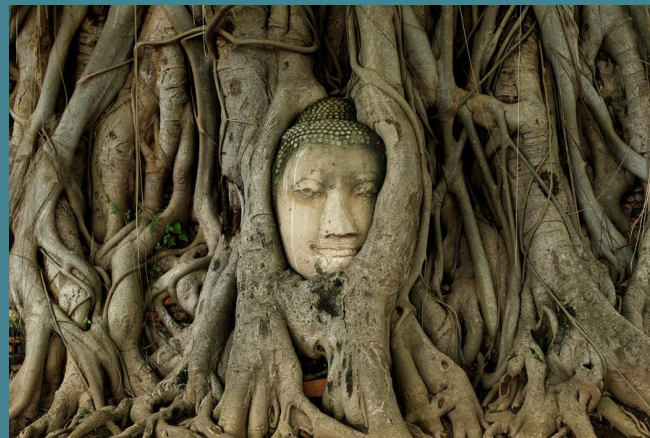
Differences between B tree and B+ tree

- B+ tree stores all keys in its leaf nodes, B tree does not

- This means information retrieval on a B Tree moves in index order "sideways" on leaves of the same level ("in-order traversal")

- The B+Tree searches from root to leaves and does not need to iterate through nodes on the same level; quicker search time

Advantages of B tree over B+ tree?

- "In-order traversal" keeps keys in sorted order
- No redundant nodes results in smaller "index file" than B+ Tree; less space
- B Tree is older version of B+Tree; not that advantageous
- The difficulty with both trees in general is "concurrency": being accessed by more than one program or network and maintaining integrity and efficiency over time as it grows
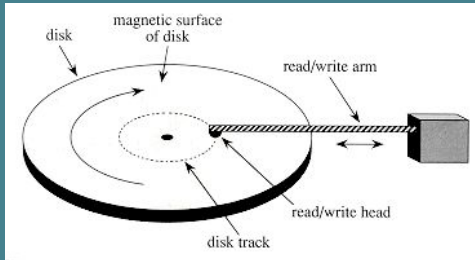
Advantages of B+tree over B tree

- B+ trees do not waste space, since all data stored in leaf nodes

- Keeps data easily ordered, thus reducing search time; structural simplicity!

- The reduced height also makes search queries quicker

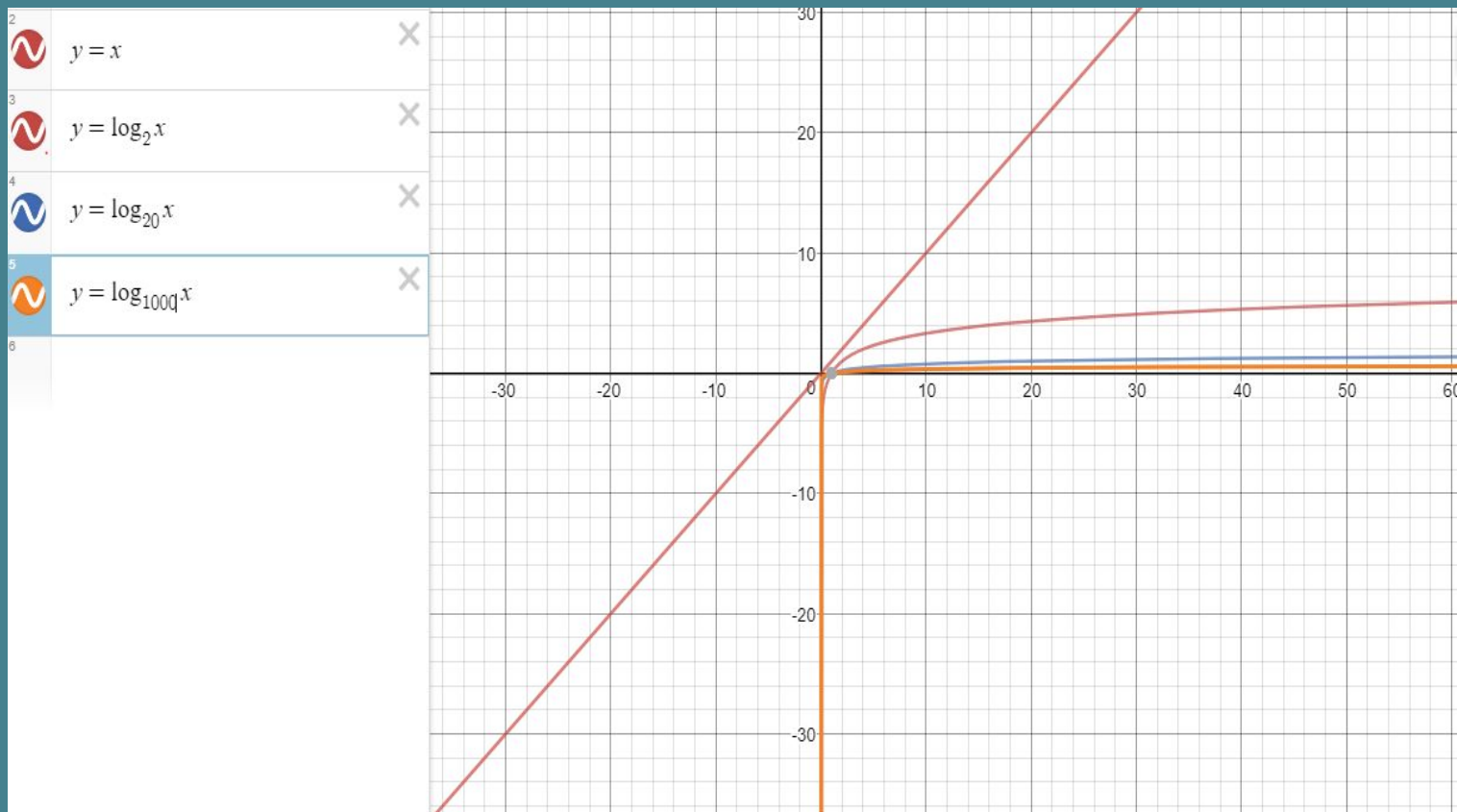- B+ tree more efficient for database systems; minimize disk access

Comparison of Run Times

O(n) - space

B-Tree

B+Tree: Order 20

Order 1000

$y = x$

$y = \log_2 x$

$y = \log_{20} x$

$y = \log_{1000} x$

Variations and Future of B trees

2-3 Tree : B Tree of order 3

2-3-4 tree: B tree of order 4, used for "dictionary" data structure

B* tree: keeps internal nodes densely packed

Research continues for making database structures more efficient:
Examples:

- Fractal Prefetching B+ Trees
- MB+ Tree: Modularized B+ Tree, uses different algorithms for each level of its tree (module)

Need to create a database?

Use a B+Tree!

Thomson Kneeland