## Disk-Bound Searching

Big-Oh analysis assumes all operations are equal,
but this is not true when disk I/O is involved.

Most databases are too large to fit in RAM, so they have to be stored on hard disk. (Google and Altavista are exceptions.)

Disk access is much slower than RAM access.

## The Cost of Access

A 500-MIPS processor executes 500 million instructions per second.

A hard disk spins at 7200 RPM, so it can perform about 120 disk accesses per second (possibly even fewer, taking account of the time taken to move the disk head).

Thus, we are willing to perform up to 4 million operations in order to avoid a disk access.

## BST Analysis

Consider a database of 10,000,000 records,
with a record length of 256 bytes and a key length of 32 bytes.

| | | |
|---|---|---|
| Worst case: | $N = 10,000,000$ | disk accesses |
| Average case: | $1.38 \log N = 32$ | disk accesses |

## $d$-ary trees

Q: How can we reduce the number of disk accesses for a tree?

A: Allow each node to have more than two children!

The tree becomes shorter and fatter.

In a binary tree, we store one data item at each leaf and one key at each non-leaf node, to tell us which of two branches to follow.

In a $d$-ary tree, we store up to $d$ children (addresses) and
$d{-}1$ data items (to tell us which of $d$ branches to follow).

## How to choose $d$?

Typically a whole *block* of data is read from the disk at one time.

We choose $d$ so that each node fits neatly into one block.

Suppose each block is 8192 bytes, each data item (including key) is 256 bytes, and the address of a block occupies 4 bytes.

Then the size of each node is

$$256\,(d-1) + 4\,d$$

We choose the largest $d$ for which this number is less than 8192, i.e. $d = 31$.

## B-Trees

B-Trees are *balanced* $d$-ary trees.

They must satisfy these conditions:

1. The root is either a leaf or has between 2 and $d$ children.
2. All other non-leaf nodes have between $\lceil d/2 \rceil$ and $d$ children.
3. (non-root) leaf nodes have between $\lceil d/2 \rceil - 1$ and $d-1$ items.

(Note: There is a more efficient version of B-Trees – which we will not discuss in the course – where all data items are stored in the leaves and only keys and addresses are stored in the non-leaves.)

## Inserting into a B-Tree

1. Find the appropriate node, and insert the new data item.
2. If that node becomes full (more than $d-1$ items), split it into two nodes with $\lfloor d/2 \rfloor$ and $\lceil d/2 \rceil - 1$ items.
3. If the parent node becomes full (more than $d$ children), split it into two nodes with $\lfloor d/2 \rfloor + 1$ and $\lceil d/2 \rceil$ children. If the grandparent becomes full, the split propagates up to the great-grandparent, etc.
4. If the root node becomes full, a new root node is created with two children (the height of the tree increases by one).

(Some enhancements to the algorithm are possible, involving “adoption” of children by neighboring nodes.)

## Deleting from a B-Tree

1. If the item is in a non-leaf node, swap it into a leaf node.
2. If the leaf node falls below $\lceil d/2 \rceil - 1$ items, try to adopt an item from a neighboring leaf – unless the neighboring leaf also has only $\lceil d/2 \rceil - 1$ items, in which case we combine the two into one leaf.
3. If the combining of the two leaves causes the parent node to have fewer than $\lceil d/2 \rceil$ children, follow the same strategy of adopting or combining with a neighbor. If necessary, this process continues to the grandparent and great-grandparent, etc.
4. If the root ends up with just one child, remove the root making its child the new root (the height of the tree decreases by one).

On average, only 2 out of every $d$ insertions or deletions cause a split or merge. Splits and merges further up the tree are very rare.

## Summary

1. Search trees need to be balanced, unless the data are known to be reasonably random, or the amount of data is small.
2. AVL trees are the simplest kind of balanced BST.
3. Red-Black and AA-trees achieve faster performance than AVL.
4. B-Trees are preferred when the amount of data is too large to store in main memory.