Disk accesses dominate instructions done in RAM.

Author's estimate:

\[ \text{disk spins at 7,200 RPM} \]
\[ \text{60 sec / 7200 rotations } \rightarrow \text{.0083 sec / 1 rotation} \]
\[ .0083 \text{ sec } = 8.3 \text{ milliseconds} \]
\[ \rightarrow \text{ 1 rot/sec} \]

On average, the disk does ½ rotation to get to the data being searched for. But then the disk head also has to move to the right spot.

So it turns out that it takes about 8.3 ms / disk access; i.e., 120 disk accesses/sec.

In a database system, you're usually not the only one accessing the data.

Say you're one of 20 users, so you have 1/20th of the resources.

So now you can do only 6 disk accesses/sec.

If you're running on a 500 MIPS computer, you can do \( \frac{500}{20} = 2.5 \) million in-RAM instructions per second if you're one of 20 users.
It's over 4,000,000 times more costly, in time, to do disk accesses than to do in-RAM instructions.

The number of disk accesses required to look something up in a database is a function of the depth of the tree in which the data is stored.

The data is stored in a tree structure where all the non-leaf nodes are indexes into the data and the leaf nodes hold the data.

It's better to have a fat, shallow tree than a skinny deep one. Every time you go another level down the tree, you're doing a disk access - reading in a whole block from the disk. Each node in the tree is a disk block with multiple pieces of data.

A B-tree is an efficient way to store data in a database.
A B tree of order M

1. Data in leaves.
2. Non-leaf nodes have up to \( M-1 \) keys to guide search; key \( i \) represents the smallest key in subtree \( i+1 \).
3. The root is either a leaf or has between 2 and \( M \) children.
4. All non-leaf nodes except the root have between \( \lceil M/2 \rceil \) and \( M \) children.
5. All leaves are at the same depth and have between \( \lceil M/2 \rceil \) and \( L \) data items.

How do you decide the size of \( M \) and \( L \)?

Say one disk block holds 8192 bytes.

Each item in the database in an employee record

<table>
<thead>
<tr>
<th>ID number used as Key</th>
<th>32 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>address</td>
<td></td>
</tr>
<tr>
<td>dept</td>
<td></td>
</tr>
<tr>
<td>DoB</td>
<td></td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>256 bytes</td>
</tr>
</tbody>
</table>
In a $M$-ary B tree, how big would a non-leaf node have to be? Each one looks like this

```
|Key| Disk ptr| Key| Disk ptr| Key| Disk ptr|
```

4 bytes 32 bytes

Why is a disk ptr 4 bytes? 4 bytes is 32 bits. You can encode $2^{32}$ different numbers -- i.e., $2^{32}$ different disk addresses -- with 4 bytes. That's addresses to $2^{32}$ blocks, each of which is 8,192 bytes large. ~ 3.5 tera bytes

34,596,8 GB

There are $M-1$ keys in a non-leaf node and $M$ disk pointers.

$$32(M-1) + 4M = 32M - 32 + 4M = 36M - 32 \text{ bytes}$$

How big can $M$ be such that

$$36M - 32 \leq 8,192$$

$$36M \leq 8,224$$

$$M \leq 228.44$$

round down to 228

$M = 228$

Now how big should $L$ be

$$\left\lfloor \frac{8,192}{256} \right\rfloor = 32$$

$L = 32$
Each leaf node has between 16 and 32 data items.

Each non-leaf node has at least \( \lceil M/2 \rceil \) children, which is 114 children.

Say there are 10,000,000 data items. If the leaf nodes are all just \( \frac{1}{2} \) full, you'd need
\[
\left( \frac{10,000,000}{32} \right) \times 2 = 625,000 \text{ leaf nodes}
\]

The worst case number of accesses

\[
\log_{M/2} N
\]
Insert 15

this has to be split in half

But now the parent of these nodes,

would have 6 children. That's not allowed because this is an M-ary B tree. We have to split the parent.

Do this from the bottom up, maintaining this property

for every key with left ptr on the left and right ptr on the
right, all keys in leaf nodes reachable from
left ptr are less than key and all keys in leaf nodes reachable by right ptr are
greater than or equal to key.

This is how you know what keys should be
put in the parent node.

Split the parent into 2 nodes with 2 keys in left
node and 2 keys in right:

This changes the next parent node up the tree:
Here's a situation where the root node has to be split.

<table>
<thead>
<tr>
<th>46</th>
<th>51</th>
<th>54</th>
<th>57</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>42</td>
<td>44</td>
<td>46</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>50</td>
<td>52</td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>55</td>
<td>56</td>
<td>58</td>
<td>59</td>
</tr>
</tbody>
</table>

Insert 45