BTree Analysis

The height of the BTree determines maximum number of ____________ possible in search data.

...and the height of the structure is: ________________.

Therefore: The number of seeks is no more than ____________.

...suppose we want to prove this!
In our AVL Analysis, we saw finding an upper bound on the height (given $n$) is the same as finding a lower bound on the nodes (given $h$).

We want to find a relationship for BTrees between the number of keys ($n$) and the height ($h$).
**BTree Analysis**

**Strategy:**
We will first count the number of nodes, level by level.

Then, we will add the minimum number of keys per node \((n)\).

The minimum number of nodes will tell us the largest possible height \((h)\), allowing us to find an upper-bound on height.
BTree Analysis

The minimum number of **nodes** for a BTree of order \( m \) at each level:

- root:

- level 1:

- level 2:

- level 3:

  ...

- level \( h \):
BTree Analysis

The total number of nodes is the sum of all of the levels:
BTree Analysis

The total number of keys:
BTTree Analysis

The **smallest total number of keys** is:

So an inequality about \( n \), the total number of keys:

Solving for \( h \), since \( h \) is the number of seek operations:
A CS 225 Field Trip (Section AL1, 11am ONLY)
Nick Holonyak
- Invented the visible-light LED
- Professor of ECE at UIUC
- Celebrating his 90th birthday!
A CS 225 Field Trip (Section AL1, 11am ONLY)

This Friday (Oct. 26), **CS 225-AL1 (11:00am)** will meet in Lincoln Hall Theater instead of ECEB.

**CS 225-AL2 (2:00pm)** will meet in ECEB as normally scheduled. *(However, feel free to come to AL1!)*
MP4 Animations
BTree Analysis

Given $m=101$, a tree of height $h=4$ has:

Minimum Keys:

Maximum Keys:
Hashing

Goals:
We want to define a **keyspace**, a (mathematical) description of the keys for a set of data.

...use a function to map the **keyspace** into a small set of integers.
## Hashing

<table>
<thead>
<tr>
<th>Locker Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td></td>
</tr>
</tbody>
</table>
Hashing
A Hash Table based Dictionary

Client Code:

```csharp
Dictionary<KeyType, ValueType> d;

// Populate dictionary
```

A **Hash Table** consists of three things:

1. 

2. 

3. 
A Perfect Hash Function

(Angrave, CS 241)
(Beckman, CS 421)
(Cunningham, CS 210)
(Davis, CS 101)
(Evans, CS 126)
(Fagen-Ulmschneider, CS 225)
(Gunter, CS 422)
(Herman, CS 233)
A Perfect Hash Function

Keyspace:
Rolling 5 dice!

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>