* The height of a BTree determines the maximum number of **disk/network/space seeks** possible when searching for data.
* The height of a BTree is **logm (n)** where **m** is the order of the BTree.
* Therefore **seeks <= logm(n)**

To prove the above property, we need to find a relationship between the number of keys **n** and the height of the BTree **h**. In other words: how is the height **h** bounded by the keys **n?**

**Minimum** number of keys in a BTree of height **h** and order **m**:

Level by level analysis:

Let t= ceil(m/2) 

|  |  |  |  |
| --- | --- | --- | --- |
| Level | Nodes | Keys | Children |
| Root | 1 | 1 | 2 |
| 1 | 2 | 2\*(t-1) | 2\*t |
| 2 | 2\*t | 2\*t\*(t-1) | 2\*t^2 |
| 3 | 2\*t^2 | 2\*t^2\*(t-1) | 2\*t^3 |
| ... | ... | …. | …. |
| h | 2\*t^(h-1) | 2\*t^(h-1)\*(t-1) | 0 (leaves) |

**Min total nodes** =

**Min total keys** =



Thus: *(for any BTree of height h and order m)*

Solving for h:

Since we can say:

Thus we have:

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

Minimum keys:

Maximum keys: //Practice problem

### **Hashing Introduction**

As the high school locker number-to-student name, it is a one-to-one mapping:



|  |  |
| --- | --- |
| **Locker #** | **Name** |
| 103 | Craig |
| 92 | Wade |
| 330 | ... |
| 46 |  |
| 124 |  |

**Keyspace** = all possible locker numbers

There are **3 components** of Hash Tables:

1. The hash function: f(h) -> Integers

* Choose a good hash function is tricky
* Do not use self created hash function

1. The compression: Integer -> array
2. What happened in chaos

Ideally, a Perfect Hash Function is a **one-to-one** function, where for each **data / input**, the hash function produces a **unique output**.

Suppose we want to map the horizontal linefollowing (faculty, course) pairs into a hash table.

Since no two faculty first names start with the same letter; we can define a perfect hash function

**f:= (faculty, course) → first letter of first name ASCII index**



A **good** hash function should be:

* 1. Computation time take O(1)
* 2. Deterministic
* 3. Satisfy the SUHA = simple uniform hashing assumption where p(f(k1) == f(k2)) = 1/n