<table>
<thead>
<tr>
<th>Operation</th>
<th>Unsorted List (Array/Linked)</th>
<th>Sorted Array</th>
<th>Sorted Linked List</th>
<th>Hash Table</th>
<th>Balanced BST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains</td>
<td>(O(n))</td>
<td>(O(\log n))</td>
<td>(O(n))</td>
<td>(O(1)) expected (0(n)) worst</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>Put</td>
<td>(O(n))</td>
<td>(O(\log n)) if key present</td>
<td>(O(n))</td>
<td>(O(1)) expected (0(n)) worst</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>Remove</td>
<td>(O(n))</td>
<td>(0(n))</td>
<td>(O(n))</td>
<td>(O(1)) expected (0(n)) worst</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>Iterate</td>
<td>(O(n))</td>
<td>(0(n))</td>
<td>(O(n))</td>
<td>(0(n))</td>
<td>(0(n))</td>
</tr>
<tr>
<td>Sorted Iterate</td>
<td>(O(n \log n))</td>
<td>(0(n))</td>
<td>(0(n))</td>
<td>(O(n \log n))</td>
<td>(0(n))</td>
</tr>
</tbody>
</table>

*assuming \(x \leq c\) and hashes of keys uniform

Keys: disjoint intervals of integers
(10, 20) jrg 94
(30, 60) kg 301
(61, 100) jrg 94
(105, 170) kg 301

Values: whatever

Get: given int \(x\), find value for interval containing \(x\)

\(\text{get}(30) = kg 301\) need keys in sorted order
Binary Search Trees

Perfectly balanced tree height $O(\log n)$

$\sum_{k=0}^{n-1} \binom{n}{k} = 2^{n+1} - 1$

$2^{h+1} - 1 = n$

$h = \log_2(n+1) - 1$

$h$ is $O(\log n)$
**BST Properties**

- **Node**: data (key/value), left/right pointers, parent pointer.
- **Children of DXB**
- **Leaves**

**Properties**: Everything in left subtree of node \( n \) has key < \( n \)'s key. Everything in right subtree of node \( n \) has key > \( n \)'s key.
bool smap_contains_key(smap *m, const char *key)
{
    smap_node *curr = m->root;
    while (curr != NULL && strcmp(key, curr->key) != 0)
    {
        if (strcmp(key, curr->key) < 0)
        {
            curr = curr->left;
        }
        else
        {
            curr = curr->right;
        }
    }
    return (curr != NULL);  // why did we stop?
}
bool smap_contains_key(smap *m, const char *key)
{
    smap_node *curr = m->root;
    while (curr != NULL && strcmp(key, curr->key) != 0)
    {
        if (strcmp(key, curr->key) < 0)
        {
            curr = curr->left;
        }
        else
        {
            curr = curr->right;
        }
    }
    return (curr != NULL);
}
Unshapely Trees

rule ball chain hamster cheese dunk donut

worst case height = n

put/contains/get  O(h)  one iteration per level  O(1) per item

Worst case  O(n)

adding in sorted (or reverse) goes worst case

counted (or reverse) goes worst case

6 - 0 = 6
AVL Tree

- AVL: always balance -1, 0, 1
- worst case h is O(log n)
- so put/contains/get O(log n)

balance: difference between heights of subtrees (left - right)

add(s, 4VR)

root:

DXB

S

B06

PTT

PTY

PTX

PXR

YVR

 dye (s, 4VR)
Rotations

(+3 other cases)

6 pts change per rotation (so O(1) per rotation)