Map Implementation Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Unsorted List (Array/Linked)</th>
<th>Sorted List</th>
<th>Sorted Linked List</th>
<th>Hash Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>contains / get</strong></td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(1)$ expected $O(n)$ worst case $O(\log n)$</td>
</tr>
<tr>
<td><strong>put / add</strong></td>
<td>$O(n)$</td>
<td>$O(\log n)$ if key present</td>
<td>$O(n)$</td>
<td>$O(1)$ expected $O(n)$ worst case $O(\log n)$</td>
</tr>
<tr>
<td><strong>remove</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$ expected $O(n)$ worst case $O(\log n)$</td>
</tr>
<tr>
<td><strong>iterate through all (key, value)s</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Sort:
- $O(n \log n)$
- $O(n)$
- $O(n)$
- $O(n \log n)$
- $O(n)$

**Keys**: disjoint intervals of integers
- $(10, 20)$
- $(30, 60)$
- $(61, 100)$
- $(105, 120)$

**Values**: whatever

**get**: given int $x$, find value associated w/ interval containing $x$

$$get(54) = k_3 301$$
Binary Search Trees

Oct 31, 2019 Page 2
Properties: for every node, values of keys in left subtree all < key in n
for every n, values in right subtree > value in n
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);
}

Searching in a BST
bool smap_put(smap *m, const char *key, int value)
{
    smap_node *curr = m->root;
    smap_node *prev = NULL;
    while (curr != NULL && strcmp(key, curr->key) != 0)
    {
        prev = curr;
        if (strcmp(key, curr->key) < 0)
        {
            curr = curr->left;
        }
        else
        {
            curr = curr->right;
        }
    }
    if (curr == NULL)
    {
        // make a new node
        // link new node with existing nodes
        if (strcmp(key, prev->key) > 0)
        {
            prev->right = new_node;
            new_node->parent = prev;
        }
    }
}