

CPSC 311-501 Homework 3

• **Ex. 12.1-2**

- What is the difference between the binary-search-tree property and the min-heap property?

Let x be a node with two children in a binary tree B . If B is a binary search tree, then x is greater than or equal to its left child, and less than or equal to its right child. If B is a min-heap, then x is less than or equal to its two children.

- Can the min-heap property be used to print out the keys of an n -node tree in sorted order in $O(n)$ time?

No. To print out the keys in sorted order, we first need to sort the min-heap. Heapsort takes $\Omega(n \lg n)$ time in the worst case.

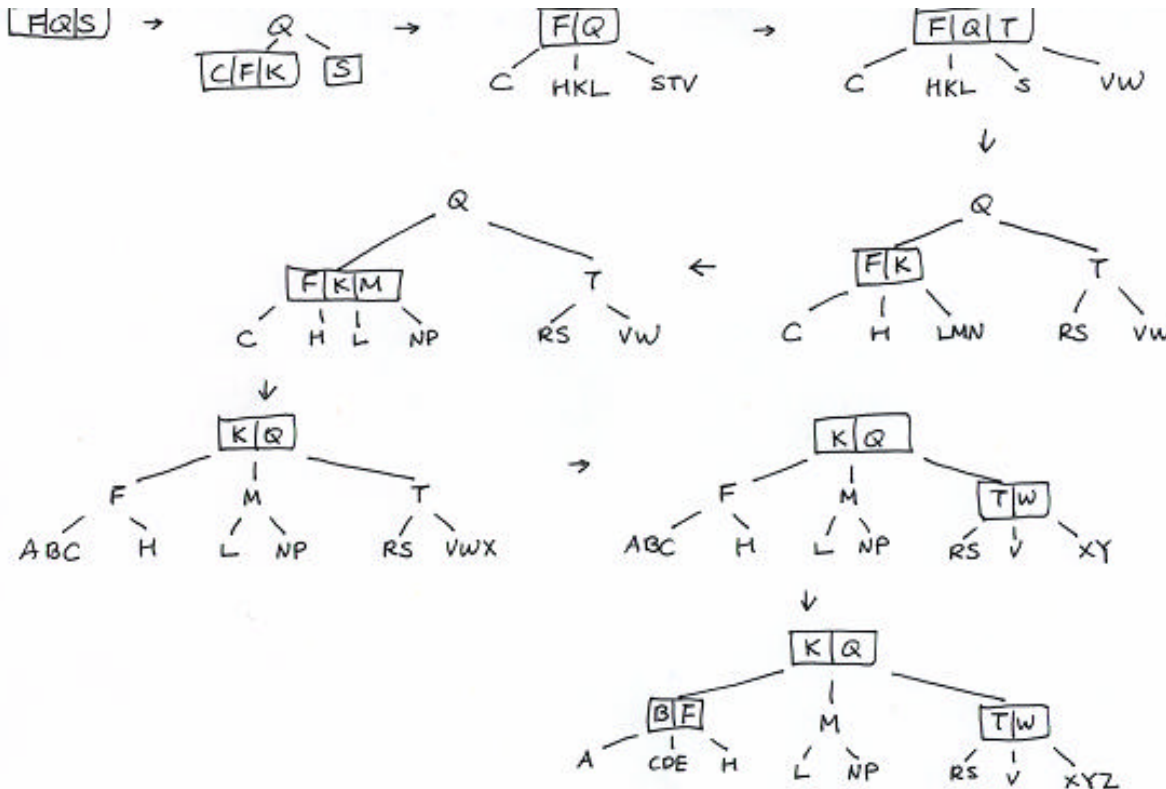
• **Ex. 12.1-5**

An inorder traversal of a binary search tree constructed from an arbitrary list of n elements prints out its keys in sorted order in $\Theta(n)$ time. Thus, any comparison-based algorithm for constructing a binary search tree from a list of n elements should take $\Omega(n \lg n)$ time in the worst case. Assume otherwise. Then, we would have a comparison-based sorting algorithm that requires less than $\Omega(n \lg n)$ time. This contradicts Theorem 8.1.

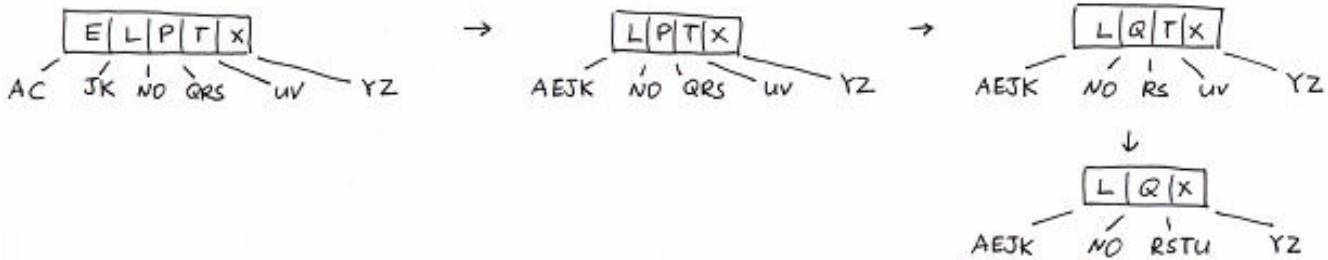
• **Ex. 12.3-3**

The running time for this algorithm is $(n \cdot (\text{the running time of TREE-INSERT})) + \text{the running time of INORDER-TREE-WALK} = n O(h) + \Theta(n)$ where h is the height of the binary search tree. The worst-case running time is $n \cdot O(n) + \Theta(n) = O(n^2)$. The best-case running time is $n \cdot O(\lg n) + \Theta(n) = O(n \lg n)$.

• **Ex. 18.2-1**



• **Ex. 18.3-1**



• **Prob. 18-2, parts (a) and (b)**

▪ (a)

When creating a B-tree (B-TREE-CREATE(T) in p. 442), after we allocate a new root node x , we set $\text{height}[x]$ to be zero.

When splitting a node (B-TREE-SPLIT-CHILD(x, i, y) in p. 444), after we split y into two nodes, y and z , we set $\text{height}[z]$ to be $\text{height}[y]$.

When inserting a key (B-TREE-INSERT(T, k) in p. 445), after we allocate a new root node, s , we set $\text{height}[s]$ to be $\text{height}[r]+1$.

Our implementation does not change the search and delete operations, and does not affect their asymptotic running times. The running time of B-SPLIT-CHILD is $\Theta(t) + O(1) = \Theta(t)$, and the running time of B-TREE-INSERT is still $O(t \log n)$.

▪ (b)

(i) If $h' < h''$, then find the leftmost node of T'' , x , such that $\text{height}[x] = h'$. Let $p(x)$ be the parent node of x . While descending to x , whenever a full node is encountered, split the node. Make k be the leftmost key of $p(x)$. Make T' be the first child of $p(x)$.

(ii) If $h' = h''$, then allocate a new root node, x . Insert k into x . Make T' be x 's first child and make T'' be x 's second child.

(iii) If $h' > h''$, then find the rightmost node of T' , x , such that $\text{height}[x] = h''$. Let $p(x)$ be the parent node of x . While descending to x , whenever a full node is encountered, split the node. Make k the rightmost key of $p(x)$. Make T'' be the last child of $p(x)$.

Case (i) takes $O(h'' - h')$ time to descend to x , and $O(1)$ time to insert k into $p(x)$ and to make T' be $p(x)$'s child. Thus, it takes $O(1 + |h' - h''|)$ time. Similarly, case (iii) takes $O(1 + |h' - h''|)$ time. Case (ii) takes $O(1) = O(1 + |h' - h''|)$ time.

• **Ex. 11.2-3**

The running times for successful searches, unsuccessful searches, and deletions are not affected. The worst-case running time for insertion becomes proportional to the length of the list. Assuming simple uniform hashing, insertion takes $O(1 + I)$ time where $I = n/m$.

• **Prob. 11-3, part (a).**

The probe sequence is:

$$h(k, 0) = h(k)$$

$$h(k, 1) = (h(k) + 1) \bmod m$$

$$h(k, 2) = (h(k) + 1 + 2) \bmod m$$

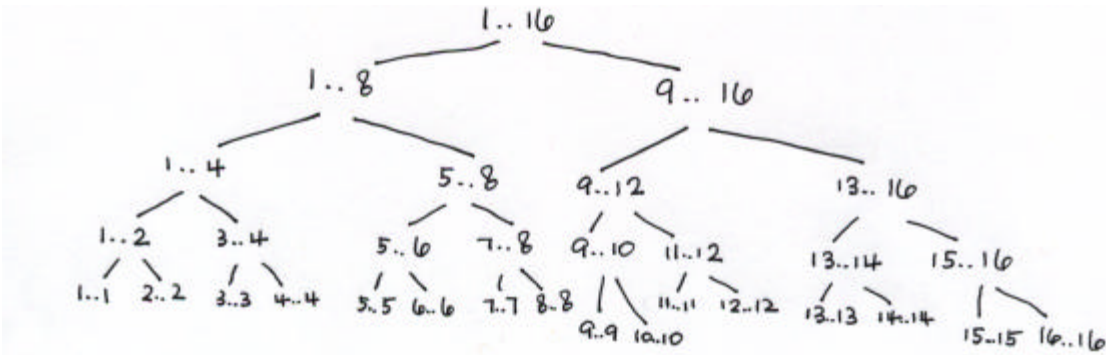
$$h(k, 3) = (h(k) + 1 + 2 + 3) \bmod m$$

...

$$h(k, i) = (h(k) + 1 + 2 + 3 + \dots + i) \bmod m = (h(k) + i(i+1)/2) \bmod m = (h(k) + i/2 + i^2/2) \bmod m.$$

Thus, $c_1 = c_2 = 1/2$.

- Ex. 15.3-2



Since there are no overlapping subproblems, maintaining a table with subproblem solutions is ineffective in speeding up MERGE-SORT.