

HW5 Solution Sketches

March 4, 1999

12.4-1 Here is what the final table looks like:

slot	0	1	2	3	4	5	6	7	8	9	10
linear probing	22	88	-	-	4	15	28	17	59	31	10
quadratic probing	22	-	88	17	4	-	28	59	15	31	10
double hashing	22	-	59	17	4	15	28	88	-	31	10

Grading: two points for each method. You lose 1 if you get up two entries incorrect. 0 of more than two entries incorrect.

13.1-2 The BST property says nodes are ordered inorder. That is, all the nodes in the left subtree of x are smaller than x , all the nodes in the right subtree of x are larger than x . The heap property says that the children of a parent are smaller than the parent, but puts no constraints on the relative order of the children.

You can't use a heap to print out nodes in sorted order in time $O(n)$. We know it only takes time $O(n)$ to build the heap (see Section 7.3), so if we could sort in time $O(n)$ given a heap, we could sort in time $O(n)$, which can't be done.

[Grading: 1 point each for stating the BST property and heap property. 2 points for saying that a heap can be used to sort and justifying it.]

[Grader: Eric Melin]

13.1-5 Given a binary search tree, we can sort in time $O(n)$ in the comparison-based model (without doing any comparisons at all). Since sorting must take $O(n \lg n)$ comparison, building the tree must have taken $O(n \lg n)$ comparisons.

[Grader: Eric Melin]

13.2-3 There are two cases. First suppose that x has a right successor. Then TREE-SUCCESSOR(x) returns the minimum element in the subtree rooted at $right(x)$. Suppose this element is y . Clearly y is the successor of x in the subtree rooted at x . We still have to argue that there is no element z such that $x < z < y$ anywhere else in the tree. Let $p^0[x]$ be x , let $p^1[x]$ be the parent of x , $p^2[x]$ be the grandparent of x , etc. The root of the tree is $p^m[x]$ for some m . We show by induction on k that y is the successor of x in the subtree rooted at $p^k[x]$, for $k = 0, \dots, m$. We've already shown it for $k = 0$. Suppose we've shown it for k ; we show it

for $k + 1$. If $p^k[x]$ is the left child of $p^{k+1}[x]$. Then $p^{k+1}[x]$ and all the nodes in its right subtree must all be bigger than y , so y is still the successor of x in the tree rooted at $p^{k+1}[x]$. If $p^k[x]$ is the right child of $p^{k+1}[x]$, then $p^{k+1}[x]$ and all the nodes in its left subtree are smaller than x , so y must also be the successor of x in the subtree rooted at $p^{k+1}[x]$. This completes the inductive step. It follows that y is the successor of x in the tree rooted at $p^m[x]$, i.e., in the whole tree. Next suppose that x has no right child. Then TREE-SUCCESSOR goes up the tree until it finds the first ancestor $p^k[x]$ such that $p^k[x]$ is the left child of $p^{k+1}[x]$. In that case, we return $p^{k+1}[x]$. (Note that if there is no such ancestor, x is the rightmost node in the tree, and hence the largest node.) We want to show that $p^{k+1}[x]$ is the successor of x in the tree. Since $p^k[x]$ is a left child of $p^{k+1}[x]$, $p^k[x]$ and all the nodes below it (including x) are smaller than $p^{k+1}[x]$, so that $p^{k+1}[x] > x$. Moreover, since x is the rightmost node in the subtree rooted at $p^k[x]$, it is the largest node in that subtree. Thus, $p^{k+1}[x]$ is the successor of x in the subtree rooted at $p^{k+1}[x]$. We now do exactly the same inductive argument as before to show that $p^{k'+1}[x]$ is still the successor of x in the subtree rooted at $p^{k'}[x]$ for all k' up to m .

Grading scheme: 4 points for each of the two cases. First part: -1 if did not prove (by induction) that no other element in the tree can be a successor. Second part: -1 if similar inductive argument not mentioned.

[Grader: Indranil Gupta]

13.2-4 A priori it seems that this algorithm would take time $O(n \lg n)$, since each call to TREE-SUCCESSOR could take time $O(\lg n)$. However, it's not as bad as that. It's easy to see that the sequence of calls to TREE-SUCCESSOR traverses the tree following an inorder tree walk. We now show by strong induction on the size of the tree that when we do a walk on a BST, starting at the root, from there going to the smallest element, then the next smallest element, and so on, then going from the largest element back up to the root, we traverse each edge of the tree twice, once going down and once going up. This depends crucially on the BST property. Since a tree with n nodes has $n - 1$ edges (with each node other than the root, we can associate the edge going up to its parent; this takes care of all the edges in the tree), this shows that the walk takes time $O(n)$.

The base case of the induction is immediate. Now suppose we have a BST with n nodes. Let r be the root of the BST, and let x and y be the left and right children of r , respectively (one of these may be NIL). We do our walk as follows:

- we first go from r to its left child x (if there is one)
- we then walk through the tree T_x rooted at x (since all the nodes in T_x are less than x)
- the next node we visit is r , which means we need to come back to x , then go from x to r
- we then go from r to y and traverse the subtree T_y rooted at y .
- we then go from the largest node in T_y to r , which mean we must pass y .

It is clear from this description that the edges from r to x and from r to y are each traversed twice, once going down and once going up. The induction hypothesis assures us the the walks

through T_x and T_y , starting and ending at x (resp., y) results in going through each edge in T_x and T_y twice. This completes the inductive step of the proof.

Grading criteria: pointing out the fact that each edge is traversed twice will get 2 points, to show the walk takes time $O(n)$ with the fact and the BST property will get 1 point. The induction proof of the fact will take 5 points (Base case: 1 point, inductive case: 4 points). Several students used the substitution method directly and got full marks.

Common mistakes:

1. Some students haven't mastered the substitution method yet. Please refer to the comment of 4.1-1 in hw1 solution.
2. Several students claimed that if using the recursive inorder tree walk algorithm, then between printing one node and its successor, the algorithm will traverse a path between these two points. With this assumption, since the algorithm in this question always traverses the shortest path between one node and its successor, so it is not slower than the recursive algorithm. However, this assumption is vague since they regarded the recursive function call and return as the action of traversing. Moreover, no one even tried to prove it.
3. Some students considered the times of traversing a path instead an edge.

[Grader: Lantian Zheng]

14.1-3 Both the longest path and shortest path from x to a leaf have the same number of black nodes, say n . The longest path can have more red nodes than the shortest one, but it can have at most n red nodes (since we can't have two consecutive red nodes on any path, by red-black property 3). Thus, the longest path can have length at most $2n$ (if it has as many red nodes as possible), and the shortest path has length at least n (if it has no red nodes). Thus, the longest path is at most twice as long as the shortest path.

[Grader: Randy Fernando]

14.1-4 Since the height of a tree is at most twice its black height (by the previous exercise) and at least its black height, if a tree has black height k , its height is between k and $2k$. That means that it has at least internal $2^k - 1$ nodes, and at most $2^{2k} - 1$ internal nodes.

[Grader: Jason Howes]

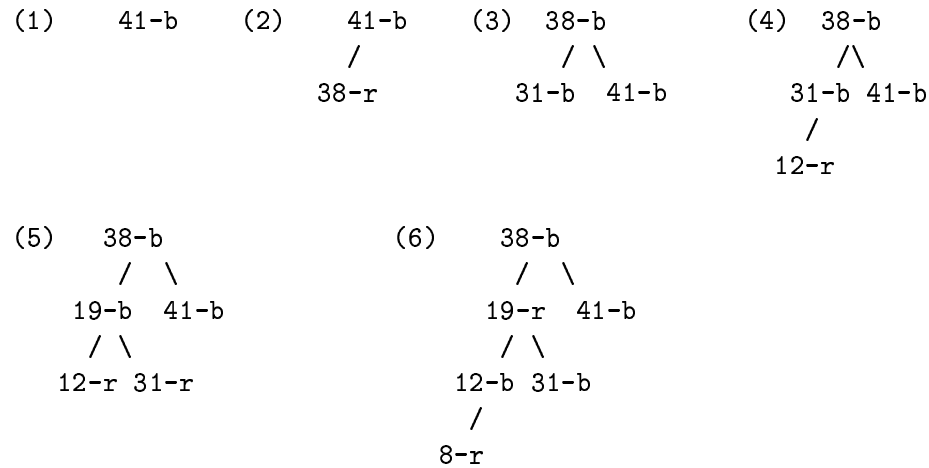
14.1-5 You get the largest possible ratio of black to red internal nodes by having a balanced tree of odd height $2k + 1$, where the root is black, the level 1 nodes are red, the level 2 nodes are black, and so on. At each odd level, there are twice as many nodes as at the preceding even level, so there are twice as many red nodes as black nodes, i.e., the ratio of red:black = 2:1. On the other hand, if we have a balanced binary tree, it satisfies the red-black properties if all the nodes are black, so the ratio of red:black = 0:1. This is clearly the smallest possible ratio.

[Grader: Jason Howes]

14.3-1 Setting the color of x to black would violate RB property 4. It's not obvious how to go about changing things to recover property 4.

[Grader: Ronnie Choy]

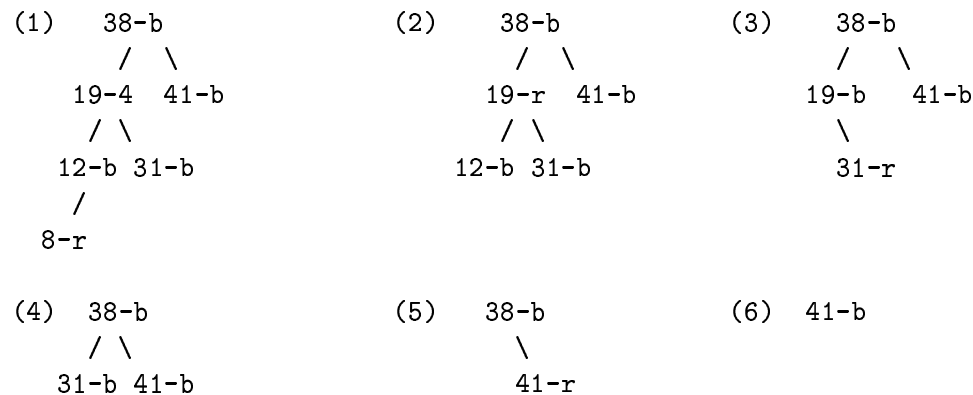
14.3-3 The sequence looks like this (after performing all the manipulations):



[Grading: 1 point for each correct transition. If a transition is incorrect, and the later transitions are consistent with it, then that's OK.]

[Grader: Ronnie Choy]

14.4-2 The sequence looks like this (after performing all the manipulations):



[Grader: Joy Alamgir]