

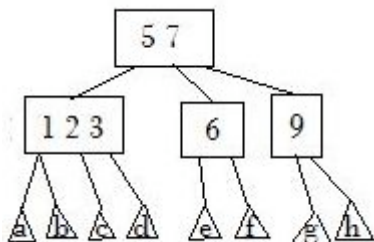
Problem 1. (7 points) A Miscellany

a. (1 point) Suppose you implement a binary search method and it takes 10 microseconds to search a 1,000,000-int array for a number that turns out not to be in the array. How long do you estimate it would take for your implementation to search a 1,000,000,000,000-int array for a number that's not in the array?

b. (2 points) Explain why  $(1/(n-100)) \in O(n)$ . Don't just reiterate the definition of big-Oh; give specific values of  $c$  and  $N$  that support the claim. (No need for a complete proof, though.)

c. (2 points) When we remove an item from a 2-3-4 tree, sometimes a node tries to "steal" a key from a sibling by performing a rotation. However, the `remove` algorithm we learned in class only allows a node to steal from an adjacent sibling. But there's no fundamental reason why a node couldn't steal a key from a nonadjacent sibling, except that we wanted to keep the algorithm as simple as possible.

Show how we could modify the 2-3-4 tree below so that the "9" node has one more key, the "1 2 3" node has one fewer key, every other node has the same number of keys it had before, and the tree is still a valid 2-3-4 tree containing the same keys it had before. The triangles at the bottom are subtrees, which you cannot restructure (but you can change their parents).



d. (2 points) Fill in the blank. The worst-case running time of the following method, expressed as a function of the input parameter  $n$ , is in  $\Theta(\text{_____})$ . Note that the loop does not use "i - -"!

```
public static void weirdo(long n) {
    Tree234 tree = new Tree234(); //Construct an empty 2-3-4 tree.
    for (long i = n; i > 1; i = i / 2) {
        tree.insert(i);
    }
}
```

It's not quite like the running time of any other algorithm you've seen. An explanation of your reasoning might help you get part marks if your answer is wrong.)

Problem 2. (7 points) Binary Trees.



