Question 4.2 from CTCI: Minimal Tree

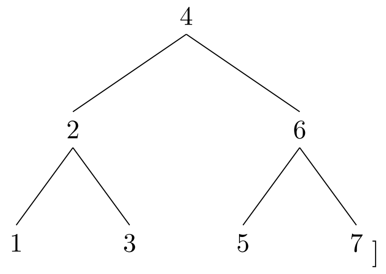
September 19, 2017

Question

Given a sorted (increasing order) array with unique integer elements, write an algorithm to create a binary search tree with minimal height.

Algorithms and Explanation

Binary search trees work with sorted arrays in the sense that the middle element is the root. Then the middle of each cluster of numbers before and after the middle of the array would be the left and right children, respectively, of the tree. For example, say you had a sorted array 1,2,3,4,5,6, and 7. 4 would be the root, and then we'd be left with two separate subarrays: 1,2,3 and 5,6,7. The middle of those two arrays are 2 and 6, respectively. This process is repeated until there are no more subarrays left. This algorithm is better explained through this binary tree example:



Hints

1. We have a sorted array given to us. Based on the algorithm described above, inserting this into a binary tree is based on the first middle element and the middle of the subarrays.
2. How should we go about organizing the array and recursively separating it into subarrays? Should it be recursive?

3. What are some edge cases for this? What kinds of arrays wouldn't work for this? You'll have to consider these before running through the rest of the program.

Code

```
/* ANSWER: RECURSIVE */
public TreeNode createMinimalBST (int array[]){
   return createMinimal BST(array, 0, array.length-1);
}

public TreeNode createMinimalBST(int a[], int start, int finish){
   if(end<start){
      return null; //this isn't a sorted array and won't work as a
           binary search tree
    }

    int mid = (start+end)/2; //index of the first middle number

    TreeNode n = new TreeNode(a[mid]); //this is the root of the tree

    /*this recursive step repeats the process of finding a middle within
         each subarray and
    breaking them into subsubarrays and so on depending on the size of
         the array */

    n.left = createMinimalBST(a, start, mid-1); //left subarray runs
         from a[start] to a[mid-1]
    n.right = createMinimalBST(a, mid+1, a.length); //right subarray
         runs form a[mid+1] to a[end]
    return n;
}
```

Big O Analysis

For the recursive solution, the total time cost is $O(nlogn)$ for n array elements. This is because the amount of times the array gets split in half into subarrays computes to $n\log n$.

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell.