CS 577: Introduction to Algorithms Discussion Solutions for Week 3

Solutions

1. We know that the largest complete subtree is either rooted at the root or some other node. If it is not rooted at the root we can simply recurse on the subtrees of the root. If it is rooted at the root, we need to know how tall it is. Finally, we need to know which one of these options is largest, so we need to compare the heights of two options. This leads to the following recursion method findDepth(node). Have a pair of global variable to track the max depth and corresponding root. Then if tree consists of a single root, return root and 0. Make root and 0 to be the current result. Then recursively call this method on its left and right child. Since any complete tree must have the left and right children have the same height, and be complete so $min(findDepth(left)$, $min findDepth(right) + 1$ will be the max depth at this node. Compare it with the current result we have and update if necessary. In the end, return the result. This is a O(n) algorithm given that we are at most searching all children once.

2. Similar to the normal inversion count question, we use merge sort. We start by splitting the array into half and call merge sort on each of them. To get the number of inversions we add the number of inversions in the left sub array, right sub array and the inversions during merge step. In the merge step, let i be the index in the left sub-array and j for the right sub-array. At any step during merging, if $A[i]$is greater than twice of $A[j]$, then there are $mid - i$ inversions, where mid is the half way point of the array, and since the left and right sub-arrays are sorted, all the following elements after i of the left sub-array will definitely be inversions. When the recursive merging is done, we will get our result that is the sum of these inversion counts.

3. In this problem, we will use Divide and Conquer rule to eliminate one-fourth of the grid each time. The idea is to split the matrix into 4 equal parts and then recursively do the following

   (a) Find the middle element.

   (b) If middle element is same as key then return.

   (c) If middle element is lesser than key then

      i. Search submatrix on lower side of middle element

      ii. Search submatrix on right hand side of middle element

   (d) middle element is greater than the key then

      i. search vertical submatrix on left side of middle element

      ii. search submatrix on right hand side

   This gives $T(n) = 3T(n/2) + O(1)$, which solves to (n^2)

4. FindMin(i,j)

   (a) If$|j - i| \leq 1$

   (b) return0;

   (c) else Set $k := \lfloor \frac{i+j}{2} \rfloor$

   (d) if $(A[k] \leq A[k+1]||(A[k] < A[k-1]))$

   (e) return k

   (f) else if $(A[k] \leq A[j])$

   (g) then FindMin(i,k-1)

   (h) else FindMin (k+1,j)