Homework 4 Solutions

1. Solve the following recurrences

   (a) $T(n) = 3T(n/2) + n; T(1) = 1$

   (b) $T(n) = T(n/8) + n; T(1) = 1$. *Ans.*

$$\begin{aligned} T(n) &= T(n/8) + n \\ &= T(n/8^2) + n + n/8 \\ &= T(n/8^3) + n + n/8 + n/8^2 \\ &= \cdots \\ &= \cdots \\ &= T(n/8^k) + n + n/8 + n/8^2 + \cdots + n/8^{k-1} \end{aligned}$$

   The recursion stops when $n/8^k$ equals one. This happens when $k = \log_8 n$. Thus

$$T(n) \leq T(1) + n + n/8 + n/8^2 + \cdots$$

   The sum of the infinite series

$$1 + 1/8 + 1/8^2 + \cdots$$

   is $\frac{1}{1-7/8}$ which equals 8. Thus $T(n) \leq 8n + 1$, thus $T(n) \in O(n)$.

2. Give a recursive algorithm that gets a sorted array (of integers) as input and constructs a Balanced BST. Your algorithm should not use balancing ideas such as AVL balancing Trees. Your algorithm must be a recursive algorithm. Write the recurrence relation for the run-time and solve the recurrence. You will receive zero credit, if your algorithm is not recursive. Your grade depends on the run-time of the algorithm.

   Ans.

   Idea: We place median element as the root of the tree And place all elements smaller than median in the left subtree and all element greater than median in the right subtree. We create the sub-trees recursively. Here is the formal description of the algorithm.

   Create Root with a left pointer and right pointer. Consider the following recursive method. This method attempts to create a BST of $A[left], A[left+1] \cdots A[right]$ with $Node$ as root.,

```
CreateTree(A, left, right, Node)
   Node.value = A[(left+right)/2];//Median Element
   CreateTree(A, left, (left+right)/2-1, Node.left);
   CreateTree(A, (left+right)/2+1, right, Node.right);
```

Now the method to construct a tree that gets a sorted array A as input makes a call to $CreateTree(A, 1, n, root)$. Note that for every *node* of the tree the number of elements in the right subtree equals the number of elements in the left sub tree. Thus if the tree has n elements, then the left and right subtree's have at most $n/2$ elements. If we use $h(n)$ to denote the size of the tree, then $h(n) \leq 1 + \max\{h(n/2), h(n/2)\}$. Thus $h(n) = 1 + h(n/2)$.

$$\begin{aligned} h(n) &\leq h(n/2) + 1 \\ &\leq h(n/4) + 2 \\ &\leq h(n/8) + 3 \\ &\leq \cdots \\ &\leq \cdots \\ &\leq h(n/2^k) + k \end{aligned}$$

Note that $h(1) = 1$. We have $n/2^k = 1$, when $k = \log_2 n$. Thus

$$h(n) = h(n/2^{\log_2 n}) + \log_2 n = O(\log n)$$

Let $t(n)$ be the run time of the above algorithm on an array of input length. Then we have the following recurrence.

$$t(n) = 2t(n/2) + 1$$

$$\begin{aligned} t(n) &= 2t(n/2) + 1 \\ &= 4t(n/4) + 2 + 1 \\ &= 8t(n/4) + 4 + 2 + 1 \\ &= \cdot \\ &= \cdot \\ &= 2^k t(n/2^k) + 2^{k-1} + \cdots + 4 + 2 + 1 \\ &= 1 + 2 + 3 + \cdots n \\ &= O(n) \end{aligned}$$

3. Given two 2-dimensional points $A = \langle x_1, y_1 \rangle$ and $B = \langle x_2, y_2 \rangle$, we say that $A < B$ if $x_1 < x_2$ and $y_1 < y_2$. Given a set of $S = \{A_1, A_2, \cdots, A_n\}$ of 2-dimensional points, the goal is to output a *maximal* set $S' \subseteq S$ such that for every pair of points C and D in S', neither $C < D$ nor $D < C$. Give a divide and conquer algorithm to solve this problem. Write the recurrence relation for the run-time and solve the recurrence. Your grade depends on the run-time. Your will receive zero credit, if you do not use divide-and-conquer. A subset S' of S maximal if the following holds: For every $D \in S - S'$, there exists C in S' such that either $C < D$ or $D < C$.

   Ans. Consider the following algorithm.

   (a) **Procedure MaximalSet(S)**

   (b) Sort S by x-axis.

   (c) Call FindMaximal(S)

(a) **FindMaximal(S)**

(b) If $|S|$ is 1, return S.

(c) Find Median A of S (based on x-co-ordinate)

(d) Let S_1 be all points whose x-coordinate is atmost x-coordinate of A

(e) Let $S_2 = S - S1$.

(f) $T = FindMaximal(S_1)$

(g) $U = FindMaximal(S_2)$.

(h) Let $B = \langle x, y \rangle$ be a point in T whose y-cooridnate is the smallest.

(i) Let U' be set of all points in U whose y-coordinate is smaller than y.

(j) Return $T \cup U'$.

Run-Time Analysis of `FindMaximal(S)`. Step (b) takes $O(1)$ time as the array is sorted. Steps $d, e, h, i, andj$ take $O(n)$ time. Thus the recurrence is

$$T(n) = 2T(n/2) + O(n)$$

Thus $T(n) = O(n \log n)$ (See notes for a way to solve this recurrence).

4. Given a directed graph $G = (V, E)$, let $G^2 = (V', E')$ be defined as follows: $V' = V$; $\langle u, v \rangle \in E'$ if there is a path of length 2 between u and v in G. Suppose that a directed graph G is given as adjacency list. Give an algorithm to compute G^2. Derive the time bound of your algorithm. Your grade depend on the run-time. Here m is number of edges in G and n is number of vertices in G.

   Ans. Consider the following algorithm: We use $AdjList(u)$ to denote the set of nodes in the adjacency list of u in the graph G. We will use A to denote the Adjlist list of G^2.

```
Let $A$ be array of size $n$, where each cell points to a linked list.
For each vertex v in V
         Create an empty list at $A[v]$.
               For each vertex u in adjList(v)
                    For each vertex w in adjList (u) (
                                If w is not v add w to the list at $A[v]$.
```

   it Correctness: Suppose that there is a path of length 2 from x to y. Suppose that the path is x to z and z to y. This implies that $z \in adjList(x)$ and $y \in AdjList(z)$. Thus when the value of v in the outermost loop is x, then the value of u becomes z during some iteration of the second loop; When this happens, the value of w becomes y in the innermost loop. Thus we place an edge from x to y in G^2.

   Runtime: Fix a vertex v. Let $d(v)$ denote the out-degree of v. Thus the total time is

$$\sum_{v \in V} \sum_{u \in Adj(v)} \sum_{w \in Adj(u)} c$$

Note that the size of any $Adj(u)$ for every u is at most n, thus the above expression is atmost

$$\sum_{v \in V} \sum_{u \in Adj(v)} cn$$

which equals

$$\sum_{v \in V} cnd(v)$$

which equals cmn, thus the time is $O(mn)$.

Caveat. The above may produce a multi-graph (I.e., there might be multiple edges between two vertices). The output can be made a simple graph by using a Boolean Array and time can be still kept $O(mn)$ in the worst-case, when $V = \{1, 2, \cdots, n\}$. Think about it.