**Quiz 2: 30 March, 2019**

**ESO207: Data Structures and Algorithms**

Time 45 minutes Max Marks 25

**Instructions**: *Please write clearly and clearly mention the question number.*

**Question 1.** [Marks 10].

Let $T$ be a binary search tree storing integers. Let *root* be a pointer pointing to its root. If $p$ is a pointer pointing to a node in this tree, then the integer stored in this node is given by $p.value$. In addition, $p.left, p.right$ and $p.parent$ denote pointers to the left child, right child, and the parent nodes of the node pointed by $p$.

Write the pseudocode for an efficient algorithm (in terms of time and memory) which takes two integers, $a$ and $b$ with $a \leq b$, as inputs and returns all the numbers between $a$ and $b$ which are stored in $T$. The algorithm must output the numbers in non-decreasing order (smallest to largest).

Determine the time complexity of your algorithm. Also determine how much additional memory is used (i.e., not including the memory used by the input tree). State both in big-O terms.

**First Solution** This solution will require $O(Depth(T))$ memory.

```
p := root;
ListRange(p);
```

**Algorithm 1:** Listing all integers in the range $[a,b]$

```
if p.value < a AND p.right ≠ null then
    ListRange(p.right);
else
    if p.value > b AND p.left ≠ null then
        ListRange(p.left);
    else
        if p.left ≠ null then
            ListRange(p.left)
        end
        ;
        Print p.value;
        if p.right ≠ null then
            ListRange(p.right);
        end
    end
end
```

**Algorithm 2:** $ListRange(p)$

**Second Solution** Here we will describe a more efficient solution which requires only $O(1)$ additional memory. We will use two pointers, $p$ and $q$. $p$ will point to the current vertex and $q$ will point to the vertex which was the previous current vertex. If $q$ is the parent of $p$, then we can deduce that this is the first visit to the current vertex. If $q$ is the left child of $p$, then we need to move to the right subtree of the current vertex. Finally if the $q$ is the right child of $p$, then the search at the current vertex is over and we must return to the parent of $p$.

```
p := root;
q := root;
/* initially locate the left most number which belongs to the Range.
   */
while [(p.value ≥ a) AND (p.left ≠ null) AND (p.parent = q)] OR [(p.value < a) AND
 [(p.parent.value ≤ b) OR (p.right ≠ null]] do
    if p.value ≥ a AND q = p.parent AND p.left ≠ null then
        q := p;
        p := p.left;
    else
        if p.value < a AND (p.right = null OR q = p.right) AND p.parent.value ≤ b then
            q := p;
            p := p.parent;
        else
            q := p;
            p := p.right;
        end
    end
end
/* Now we traverse through the nodes to read the successive values to
   the right.                                                          */
while p.value ≤ b OR (p.left ≠ null AND q ≠ p.left) do
    if q = p.right then
        q := p;
        p := p.parent;
    else
        if p.value ≤ b AND q = p.parent AND p.left ≠ null then
            q := p;
            p := p.left;
        else
            if (a ≤ p.value ≤ b) AND (p.left = null OR q = p.left) then
                print p.value;
                if p.right ≠ null then
                    q := p;
                    p := p.right;
                else
                    q := p;
                    p := p.parent;
                end
            end
        end
    end
end
```

In addition to these, we must also check the value of the current node to ensure that the search remains confined to the range $[a,b]$.

**Question 2.** [Marks 15].

A 1-player board game uses a chess like board with $n$ rows and $m$ columns. A number (positive or negative) is written on each square. Initially the game piece is placed on square $(1,1)$ (top left corner). The player can move the piece one square either to right or down each time. The player must continue to play as long as the piece is in side the board. The initial score is zero and the value in each visited square is added to the score. The goal of the game is to make moves which lead to the maximum possible score after the trail ends. *No need to compute the sequence of the moves*.

(a) Write the pseudocode of an efficient (dynamic programming based) algorithm to determine the highest possible score a player can achieve, for a given board and numbers written on the squares. Assume that the board has $n$ rows and $m$ columns. The values written on the board is given in an array $B$, i.e., number $B[i,j]$ is written on block $(i,j)$ (at row $i$ and column $j$).

(b) Justify the correctness of the recurrence relation used in the algorithm.

(c) Derive the time complexity of your algorithm.

**Solution**

(b) Let $s(i,j)$ denote the maximum possible score that a player can achieve starting from the square $(i,j)$. Define $s[i,m+1] = s[n+1,j] = 0$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$. Then the recurrence relation is as follows:

Base case: $s(n,m) = B[n,m]$,

Recurrence: $s(i,j) = \max\{s(i,j+1), s(i+1,j)\} + B[i,j]$

The base case is trivial because starting from square $(n,m)$ the only trail is the one in which only $(n,m)$ square occurs. The correctness of the recurrence is due to the fact that the game piece which is currently placed at $(i,j)$ square, must move to either $(i+1,j)$ or to $(i,j+1)$. Thus the former choice can lead to at most $s(i+1,j)+B[i,j]$ and the latter can give at most $s(i,j+1)+B[i,j]$. Thus $s(i,j)$ must be the better of the two.

(a) Assume that $m \leq n$.

(c) Since this algorithm computes each entry of the table in constant time the time complexity is $O(n+m+n.m) = O(n.m)$.

```
for i := 1 to n do
    S[i, m+1] := 0;
end
for j := 1 to m do
    S[n+1, j] := 0;
end
S[n, m] := B[i, j];
for L := n+m−1 to m+2 do
    for j := m to L−n do
        i := L− j;
        S[i, j] := max{S[i, j+1], S[i+1, j]} + B[i, j];
    end
end
for L := m+1 to 2 do
    for j := L−1 to 1 do
        i := L− j;
        S[i, j] := max{S[i, j+1], S[i+1, j]} + B[i, j];
    end
end
return S[1, 1];
```