ESO207 Programming Assignment-2.1 Solutions

1 Pseudo-Code

```
Types

Node :=  twoNode|threeNode|leafNode
Tree :=  Node

A twoNode has fields: value, lchild, rchild, parent.

A threeNode has fields: value1, value2, lchild, mchild, rchild, parent.

leafNode has fields: value, parent.
```

Type Node is thought of as a class. Tree nodes are objects of this class and can be of any of the three types, twoNode, threeNode, and leafNode.

Fields lchild, mchild, rchild, parent are of type Tree. Fields value, value1 and value2 are of natural numbers type.

Algorithm 1: Insert($T1, T2$)

Data: $2-3$ trees $T1, T2$
Result: Merges $T1, T2$ into a single $2-3$ tree and returns the merged tree
if $T1 == nil$ **then**
  return $T2$
end
if $T2 == nil$ **then**
  return $T1$
end
$h1 \leftarrow ht(T1), \quad h2 \leftarrow ht(T2)$; //hi is height of Ti.
$m2 \leftarrow min(T2)$; //$m2$ is the minimum key stored in $T2$.
if $h1 == h2$ **then**
  return *makeTree*$(T1, T2, m2)$; //make a tree with a new root node.
end
if $h1 < h2$ *;* //We find the leftmost node of $T2$ at height $h1$
then
  $x \leftarrow T2$; //x is the root of tree $T2$
  for $i = h2$ **downto** $h1 + 1$ **do**
    $x = x.left$; //explore the leftmost path
  end
  ; //x is the leftmost node of $T2$ at height $h1$
  return *InsertionLeft*$(T1, x, T2, m2)$
end
if $h1 > h2$ *;* //We find the rightmost node of $T1$ at height $h2$
then
  $x \leftarrow T1$; //x is the root of $T1$
  for $i = h1$ **downto** $h2 + 1$ **do**
    $x = x.right$; //explore the rightmost path
  end
  ; //x is the rightmost node of $T1$ at height $h2$
  return *InsertionRight*$(T1, x, T2, m2)$
end

Algorithm 2: makeTree($T1, T2, m$)

Data: $2-3$ trees $T1, T2$ of same height such that any value in $T1$ is less than any value in $T2$. m is the min key in $T2$.
Result: Creates a $2-3$ tree with root as a new twoNode, $T1$ as left child and $T2$ as right child of this node. Returns this tree.
U = new twoNode();
$U.parent \leftarrow nil$;
$U.lchild \leftarrow T1$;
$U.rchild \leftarrow T2$;
$U.value \leftarrow m$;
return U

Algorithm 3: InsertionLeft($T1, x, T2, m$)

Data: $T1, T2$ are $2-3$ trees. x is the leftmost node of $T2$ at height $ht(T1)$,
m is the min key in T_x
Result: Makes $T1$ the left sibling of x and the resulting $2-3$ tree is returned.

```
if x == T2 then
    return makeTree (T1, x, m2) ;                    //x is the root node of tree T2
end
y ← x.parent;
if y == twoNode ; //y is made into three node by adding T1 as its leftmost child
 then
    z = new threeNode() ;                                          //z is a new three node
    z.lchild ← T1 ;                                  //Initializing various fields of z
    z.mchild ← y.lchild;
    z.rchild ← y.rchild;
    z.value1 ← m2;
    z.value2 ← y.value;
    z.parent ← y.parent;
    if z.parent == nil then
        T2 ← z ;                                                          //new value of T2
    else
        (z.parent).lchild = z ;                      //z is put back in T2 in place of y
    end
    return T2
end
if y == threeNode ;                                              //Split y into two nodes
 then
    u ← new twoNode() ;                  //u is to be the right node after splitting
    u.lchild ← y.mchild;
    u.rchild ← y.rchild;
    u.value ← y.value2;
    u.parent ← y.parent;
    if u.parent == nil then
        T2 ← u ;                                                          //new value of T2
    else
        (u.parent).lchild = u;                 //u is put back in T2 at the place of y
    end
    v ← new twoNode() ;                   //v is to be the left node after splitting
    v.lchild ← T1;
    v.rchild ← y.lchild;
    v.value ← m2;
    v.parent ← nil;
    return InsertionLeft(v, u, T2, y.value1) ;          //Recursive call to make v as left
                                                        //sibling of u.  ht(v) = ht(T1) + 1
end
```

Procedure InsertionRight is completely analogous to InsertionLeft, we have omitted presenting it here. Functions ht, min have obvious $O(h)$ implementation. $Extract(T)$ can be be implemented using level-wise tree traversal, covered in one of the early lecture in the course.

maketree takes $O(1)$ time. InsertionLeft takes $c.(h2 - h1)$ time, number of recursive calls in InsertionLeft is $O(h2 - h1)$ and each call takes $O(1)$ time apart from the time in a tail recursive call. Using the above facts, Insert takes $O(h1 + h2)$ time.

2 Implementation

We have thought of nodes as objects of Type (class) Node. Types twoNode, threeNode and leafNode may be implemented as subclasses of class Node.

If Node is a structure (as in C) then Tree should be defined as a pointer to Node. In C, it may be difficult to combine three structures twoNode, threeNode and leafNode naturally into a single type/structure.

A conceptually not so nice implementation of type Node by a single C structure is possible. This structure has fields of all three structures and also has a tag field to indicate if it is a leafNode or a twoNode or a threeNode. This may even result in shorter code than the one presented here.

[No marks will be deducted for your choice of types.]