# Algorithm Design

*M. T. Goodrich and R. Tamassia*

John Wiley & Sons

# Solution of Exercise C-3.14

Assume, without loss of generality, that $n \geq m$.

Clearly, we can not use the general insertion/deletion tree operations to perform the joining; such a solution would take $O(n \log n)$ time.

Let $T$ and $U$ have heights $h_t$ and $h_u$ respectively. Our task is to "manually" join the two trees into one (2-4) tree $V$ in logarithmic time. $V$ must have all the properties of a (2-4) tree (that is, the size property, the depth property and the property of $V$ being a multi-way search tree). The idea here is very simple. Remove either the largest element of $T$ or the minimum element of $U$ (we name this element $e$). Without loss of generality, assume that after this removal $h_t \geq h_u$. If $h_t = h_u$, create a new node $v$ that stores the element $e$ that was initially removed and make $T$'s and $U$'s root $v$'s left and right respectively children. If $h_t > h_u$, insert the element $e$ into the rightmost node of tree $T$ at height $h_t - h_u - 1$ and link this node to the root of tree $U$. If $h_t < h_u$, the approach is symmetric to the one described above.

Readily, the time complexity is $O(\log n)$. Only one element is removed and re-inserted; it can be located in time proportional to tree's height and the insertion and deletion operations take each $O(\log n)$ time. The height of a tree - if it is not stored separately - can be computed in $O(\log n)$ time.

Below, we give pseudo-code for method join$(T,U)$. We assume that trees' heights are known. MostLeftRight$(T, T.\text{root}(), h, flag)$ returns the rightmost (leftmost) node of tree $T$ at height $h$, depending on if flag is set (not set).

**Algorithm** join$(T,U)$:
  $\{$remove the largest element from $T\}$
  $h_T \leftarrow$ height of $T$
  $MaxNodeT \leftarrow$ MostLeftRight$(T, T.\text{root}(), h_T, true)$
  $MaxKeyT \leftarrow T.\text{Key}(MaxNodeT, \text{Type}(MaxNodeT))$
  $MaxElemT \leftarrow T.\text{Elem}(MaxNodeT, \text{Type}(MaxNodeT))$
  $T.\text{Remove.}(MaxNodeT, MaxKey, MaxElemT)$
  $T.\text{Restructure}(MaxNodeT)$

```
{we consider three cases}
h_T ← height of T
h_U ← height of U

if h_T = h_U then {case 1}
V ← new tree
V.Insert(V.root(),MaxKeyT,MaxElemT)
V.Child(V.root(),1)← T.root()
V.Child(V.root(),2)← U.root()
return V
if h_T > h_U then {case 2}
v ← MostLeftRight(T,T.root(),h_T − h_U − 1, false)
T.Insert(v,MaxKeyT,MaxElemT)
T.Child(v,Type(v)+1)← U.root()
T.Restructure(v)
return T
if h_T < h_U then {case 2}
v ← MostLeftRight(U,U.root(),h_U − h_T − 1,true)
U.Insert(v,MaxKeyT,MaxElemT)
U.Child(v,1)← T.root()
U.Restructure(v)
return U
```

```
Algorithm MostLeftRight(T,v,h, flag):
  if h = 0 then
    return v
  if flag then
    return MostLeftRight(T,T.Child(v,T.Type(v)),h − 1, flag)
  else
    return MostLeftRight(T,T.Child(v,1),h − 1, flag)
```