```
boolean primarySeekForDelete(deleteSeekRecord, deleteKey)
{
  while(true)
  {
    loop until a null node is reached OR deleteKey is found
    if keyfound
    {
      update deleteSeekRecord with pnode,node,lastUnmarkedEdge
      return true
    }
    else
    {
      if lastRightNode's key has not changed
        return false
      else
        restart primary seek
    }
  }
}
//if special case secondary seek returns true
boolean secondarySeekForDelete(node,nRChild,secDeleteSeekRecord)
{
  if special case
    return true
  else
    keep going left until the secondary node is located //secondary node's lchild should
    be NULL
    once the secondary node is located, populate the secDeleteSeekRecord
    with secondary node,its left & right child, its parent and the
    lastSecondaryUnmarkedEdge
    return false
}

delete pseudocode
set CLEANUP and isSimpleDelete flags to false
while(true)
{
  if(primarySeekForDelete(deleteSeekRecord, deleteKey) returned false)
    return false
  else
    use the values of pnode, node and lastUnmarkedEdge from the deleteSeekRecord

  if(!CLEANUP)
  {
    try CAS(node->lChild,<nlChildAddr,x,0,0>,<nlChildAddr,x,1,0>)
    if CAS FAILED
      help
      continue from top of primary seek's while loop
    if CAS SUCCEEDED
      set CLEANUP to true and set storedNode = node
  }
  if(storedNode != node) //Someone removed the node for me. So DONE
  set "deleteFlag" on node's rChild using BTS
  if complex delete
  {
```

```
54        while(true) //secondary seek
55        {
56          nRChild = node->rChild
57          if(nRChild != NULL) //check the null bit to determine NULL values
58            isSplCase = secondarySeekForDelete(node,nRChild,secDeleteSeekRecord)
59            use the values of secondary node,its lchild,its rchild, its parent,
              SeclastUnmarkedEdge from secDeleteSeekRecord
60          else
61            if(key is marked)
62              set secDoneFlag to true
63            else
64              set isSimpleDelete flag to true
65              break from while loop
66          if node key is unmarked
67          {
68            try CAS(rnode->lChild,<anyAddress,1,0,0>,<nodeAddr,1,0,1>)
69            if CAS failed
70            {
71              if promoteFlag is set
72                if address does not match with node's address
73                  assert(node->secFlag == DONE)
74                  return
75              else
76                if address != NULL // (check the null bit to determine NULL values) restart
                  secondary seek
77                  continue from top of secondary seek's while loop
78                else
79                  assert(rnode->lChild's deleteFlag is set)
80                  help operation at secondaryLastUnmarkedEdge
81                  if secondaryLastUnmarkedEdge does not exist, then help node->rChild
                    //simplehelp(node,nrChild)
82                  continue from top of secondary seek's while loop
83            }
84            set promote flag on rnode->rChild using BTS
85            promote key using a simple write. Node's key changes from <0,kN> to <1,kRN>
86          }
87          if(!isSplCase)
88          {
89            if(rnodeRchild != NULL)
90              try CAS(rpnodeLChild,<rnode,0,0,0>,<rnodeRChild,0,0,0>) //remove secondary node
91            else
92              try CAS(rpnodeLChild,<rnode,0,0,0>,<rnode,1,0,0>) //remove secondary node
93            if CAS FAILED, help operation at secondaryLastUnmarkedEdge
94              if secondaryLastUnmarkedEdge doesn't exist, override CASinvariant and help
                node->rChild  //simplehelp(node,nrChild)
95              continue from top of secondary seek's while loop
96            if CAS SUCCEEDED, set node->secDoneFlag to true
97          }
98          else
99          {
100           if(rnodeRchild != NULL)
101             try CAS(nodeRChild,<rnode,0,1,0>,<rnodeRChild,0,1,0> //no problem if CAS fails
102           else
103             try CAS(nodeRChild,<rnode,0,1,0>,<rnode,1,1,0> //no problem if CAS fails
104           set node->secDoneFlag to true
```

```
105         }
106         oldNodeAddr = address of node
107         while(true)
108         {
109           create a fresh copy of node
110           newNodeKey as <0,kRN>
111           newNodeLChild as <node's lChildAddr,0,0,0>
112           newNodeRChild = <node's rChildAddr,0,0,0>
113           try CAS(pnode->lChild,<node,0,0,0>,<newNode,0,0,0>)
114           if CAS SUCCEEDED then DONE
115           if CAS FAILED
116             if address has changed
117               then someone helped me install a fresh copy.so DONE
118             else
119               CAS has failed coz the edge is marked.
120               if lastUnmarkedEdge is not (pnode,node) then help
121               do primarySeekForDelete(node->key) //restart primary seek with new key
122               if the new key is not found then someone has installed a fresh copy. So done

123               if key is found and newNodeAddr != oldNodeAddr then someone has installed a
                  fresh copy. So done
124         }
125       }
126     }
127     else //simple delete
128       set isSimpleDelete to true
129     if(isSimpleDelete)
130     {
131       if both l & r child of node are NULL
132         try CAS(pnode->lChild,<node,0,0,0>,<node,1,0,0>)
133       else
134         try CAS(pnode->lChild,<node,0,0,0>,<nonNullChild,0,0,0>)
135       if CAS SUCCEEDED, then DONE
136       if CAS FAILED
137         if lastUnmarkedEdge is NOT (pnode,node) help
138     }
139   }
140   //simplehelp(node,node's rChild)
141   simplehelp(pnode,node)
142   assert(pnode's secDoneFlag not set)
143   set delete flag on node->rChild using BTS
144   if complex delete
145     if node->secDoneFlag is set
146       create a fresh copy of node
147       try CAS(pnode->rChild,<node,0,1,0>,<newNode,0,1,0>)
148   else //simple delete
149     if both l & r child of node are NULL
150       try CAS(pnode->rchild,<node,0,1,0>,<node,1,1,0>)
151     else
152       try CAS(pnode->rchild,<node,0,1,0>,<nonNullChild,0,1,0>)
```