Algorithm 1 insert(key K, pointer P)

if tree is empty **then**
    create an empty leaf node L, which is also the root
else
    Find the leaf node L that should contain key value K
end if
if L has less than $n-1$ key values **then**
    insert_in_leaf(L, K, P)
else
    Create node L'
    Copy $L.P_1, \ldots, L.K_{n-1}$ to a block of memory T that can hold n (key and pointer) pairs
    insert_in_leaf(T, K, P)
    Set $L'.P_n = L.P_n$
    Set $L.P_n = L'$
    Erase $L.P_1$ through $L.K_{n-1}$ from L
    Copy $T.P_1$ through $T.K_{\lceil n/2 \rceil}$ from T into L starting at $L.P_1$
    Copy $T.P_{\lceil n/2 \rceil+1}$ through $T.K_n$ from T into L' starting at $L'.P_1$
    Let K' be the smallest key in L'
    insert_in_parent(L, K', L')
end if

Algorithm 2 insert_in_leaf(node L, key K, pointer P)

if $K < L.K_1$ **then**
    insert P, K into L just before $L.P_1$
else
    Let K_i be the highest value in L that is less than or equal to K
    Insert P, K into L just after $L.K_i$
end if

Algorithm 3 insert_in_parent(node N, key K', pointer N')

if N is the root of the tree **then**
    Create a new node R containing N, K', N'
    Make R the root of the tree **return**
end if
Let $P = parent(N)$
if P has less than n pointers **then**
    insert(K', N') in P just after N
else
    Copy P to a block of memory T that hold P and ($K'\&N'$)
    Insert(K', N') into T just after N
    Erase all entries from P
    Create node P'
    Copy $T.P_1, \ldots, T.P_{\lceil (n+1)/2 \rceil}$ into P
    Let K'' = $T.K_{\lceil (n+1)/2 \rceil}$
    Copy $T.P_{\lceil (n+1)/2 \rceil+1}, \ldots, T.P_{n+1}$ into P'
    insert_in_parent(P, K'', P')
end if