The $\log \log n$ Priority Queue

Reference: P. Van Emde Boas, R. Kaas and E. Zijlstra, "Design and implementation of an efficient priority queue", *Math. Systems Theory* 10 (1977), 99-127.

Let x be a k-bit integer. Then $low_k(x)$ will denote the $k/2$ least significant bits of x and $high_k(x)$ the remaining bits.

typedef struct $Tree$ {
    int k; // the rank, i.e., number of bit
    int $size$;   // The number of leaves in $Tree$
    node *$list$; //ptr to the minimum element of a doubly linked circular list
        //containing all the leaves of $Tree$ in sorted order
    boolean $bit[0..2^k - 1]$; // $Tree.bit[i] = 1$ iff i belongs to $Tree$
    Tree $*top$;
      // A structure of rank $k/2$ with elements $\{high_k(x_1), ..., high_k(x_n)\}$
    Tree *$bottom[0..2^{k/2} - 1]$;
        // An array of structures of rank $k/2$,
        //such that $bottom[i]$ is a structure with elements $\{low_k(x) : x \in Tree,\ high_k(x) = i\}$.*/;
} TREE

TREE T;
Note that $MIN(T)$ is $T \rightarrow list \rightarrow value$ and $MAX(T)$ can be found via $T \rightarrow list \rightarrow prev$.

int $succ$(**int** x, Tree $*T$){
// finds the minimal element in T greater than x, or returns NULL if none exists
1. **If** (T is empty **or** $x > MAX(T)$) **return** NULL.
2. **If** ($T \rightarrow k == 1$ **or** $T \rightarrow size \leq 2$)
    find succ by sequential search of $T \rightarrow list$.
3. **If** ($T \rightarrow top \rightarrow bit[high_k(x)] == 1$ **and** $low_k(x) \leq MAX(T \rightarrow bottom[high_k(x)])$)
    return $high_k(x)2^{k/2} + succ(low_k(x), T \rightarrow bottom[high_k(x)])$.
4. $z = succ(high_k(x), T \rightarrow top)$;
   return $z2^{k/2} + MIN(T \rightarrow bottom[z])$.
}

$insert$(**int** x, Tree $*T$){
1. **If** ($T == empty$)
    initialize T, put x into the list and $T \rightarrow bit$. All substructures remain empty.
2. **If** ($T \rightarrow bit[x] == 1$) we are done.
3. $T \rightarrow bit[x] = 1$.
4. insert x into $T \rightarrow list$:
    $y = succ(x, T)$;
    put x before y.
5. **If** ($T \rightarrow size == 1$){ /¿//In this case $T \rightarrow top$ and $T \rightarrow bottom[*]$ are empty
    $z = MIN(T)$;
    $insert(high_k(x), T \rightarrow top)$;
    $insert(low_k(x), T \rightarrow bottom[high_k(x)])$;
    $insert(high_k(z), T \rightarrow top)$;
    $insert(low_k(z), T \rightarrow bottom[high_k(z)])$.
   }
6. **If** ($T \rightarrow size \geq 2$){
    $insert(high_k(x), T \rightarrow top)$;
    $insert(low_k(x), T \rightarrow bottom[high_k(x)])$.
   }
7. $T \rightarrow size{+}{+}$.
}