I am indebted to my colleague, John David Stone, for assistance with this handout. 
This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/4.0/
or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.
public BinaryNode<AnyType> removeMin( BinaryNode<AnyType> t ) {
    if( t == null ) {
        throw itemNotFoundException( );
    } else if( t.left != null ) {
        t.left = removeMin( t.left );
        return t;
    } else {
        return t.right;
    }
}

public BinaryNode<AnyType> findMin( BinaryNode<AnyType> t ) {
    if( t != null ) {
        while( t.left != null ) {  t = t.left; }
        return t;
    }
}

• insert unsorted numbers in a binary search tree.
• Repeatedly call removeMin and findMin to extract elements in the sorted order
• Operations take $O(\log N)$ time.
Handle Worst Case

- Insert takes $O(N)$ time
- Total running time is $O(N^2)$ time.
- Original elements are reverse-sorted
  - removeMin also takes $O(N)$ time

- Shuffle input elements
- Less likely to generate stringy trees
- Compute internal path length and average it over all permuted inputs
- Average running time is $O(N \log N)$.
- Shuffling takes $O(N)$ time (overhead).

- Make trees bushy at all times
  - E.g., trees-balance after insertions.
- The order of inserting elements does not matter.