A Greedy Algorithm for Huffman Encoding

**Input:** A vector of objects such that each object contains a symbol and the frequency of occurrence (or weight) of that symbol.

**Output:** An optimal Huffman tree for the given set of symbols.

All symbols in the tree will be stored at leaf nodes. The weight of a leaf node will be the frequency of the symbol stored at that node, and the weight of an internal node will be the sum of the frequencies of the descendant leaves.

Use a priority queue as the key data structure in constructing the Huffman tree. Start by inserting one-node trees, each consisting of just a leaf node in the priority queue. The queue elements themselves are sorted in the increasing order of their frequencies.

The tree is constructed bottom-up. The first step is to remove the first two trees from the priority queue and combine them to form a new tree. The weight of the root node for this tree will be the sum of the weights of its left sub-tree and right sub-tree. Insert the new tree back into the priority queue, and continue this process until there is just one tree in the queue.

**Algorithm**

1. for each symbol, construct a tree consisting of just one node that contains the symbol and its weight;
2. place the set of trees from Step 1 into a priority queue;
3. while the priority queue has more than one item
   3.1. remove the two trees with the smallest weights;
   3.2. combine them into a new binary tree in which the weight of the tree root is the sum of the weights of its children;
   3.3. insert the newly created tree back into the priority queue where its position is determined by the weight of its root;

End of algorithm
Remark: Each time through the while loop, two trees are removed from the priority queue and one is inserted. Thus, effectively, one tree is removed, and the priority queue gets smaller with each iteration through the loop.

Example: Consider the following symbols along with their weights:

\[ a (12), b (10), c (15), d (19), e (16). \]

Objective is to build an optimal Huffman tree and, subsequently, the Huffman code for each symbol.

The first step is to build a priority queue of one-node trees in the order of their weights.

\[
\begin{array}{cccc}
10 & 12 & 15 & 16 & 19 \\
\text{b} & \text{a} & \text{c} & \text{e} & \text{d}
\end{array}
\]

Remove the first two trees from the priority queue and combine them to form a new tree. The weight of the root node of the resulting tree will be the sum of the weights of its left sub-tree and right sub-tree. Appropriately insert the new tree into the priority queue.

\[
\begin{array}{cccc}
15 & 16 & 19 & 22 \\
\text{c} & \text{e} & \text{d} & 22
\end{array}
\]

Again remove the first two trees in the priority queue and combine them, and appropriately insert the new tree into the priority queue.

\[
\begin{array}{cccc}
19 & 22 & 31 \\
\text{d} & 22 & 31
\end{array}
\]

\[
\begin{array}{cccc}
10 & 12 & 15 & 16 \\
\text{b} & \text{a} & \text{c} & \text{e}
\end{array}
\]
The priority queue at the next step follows.

Finally combine the last two trees in the priority queue into a new tree and put it in the priority queue.

There being only one tree in the queue, the procedure terminates.

The Huffman tree and the corresponding Huffman code are as follows.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>111</td>
</tr>
<tr>
<td>b</td>
<td>110</td>
</tr>
<tr>
<td>c</td>
<td>00</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>e</td>
<td>01</td>
</tr>
</tbody>
</table>