Approach:

We initially store elements in a vector, thereafter following the algorithm's steps as explained in the video titled “Next Greater Element In LL”. However, an alternative approach involves leveraging a stack of pairs. The first element of each pair tracks the index, while the second element holds the linked list's node value. This approach traverses the linked list until the head reaches null. During traversal, it verifies whether the element is greater than the top element in the stack (which should not be empty). If this condition is met, it stores the top element, pops it, and proceeds further. Otherwise, it pushes elements into the stack and continues traversal.

Code Section:

```
vector<int> nextLargerNodes(ListNode* head) {
// Creating a stack of pairs to hold index-value pairs
stack<pair<int, int>> st;

// Creating a vector to store the final result
vector<int> ans;

// Index variable 'i' to keep track of the current index
int i = 0;

// Iterate through the linked list until head becomes NULL
while(head){
// Add a placeholder (0) to the answer vector for the current node
ans.push_back(0);

// Check if the stack is not empty and the current node's value is greater
// than the value at the top of the stack. If true, update the answer vector.
while(!st.empty() && st.top().second < head->val){
auto top = st.top();
st.pop();
ans[top.first] = head->val;
}

// Push the current index-value pair into the stack
st.push({i++, head->val});

// Move to the next node in the linked list
head = head->next;
}
```