Title: Reversing the First K Elements of a Queue using Stack

Problem Statement: The task is to reverse the first **k** elements of a queue while keeping the rest of the elements in the same order. To achieve this, we can use a stack as it follows the Last-In-First-Out (LIFO) property, which helps in reversing the order of elements efficiently.

User's Code Implementation:

```
#include<iostream>
#include<stack>
#include<queue>
using namespace std;

void display(queue<int> &q){
  for(int i=0;i<q.size();i++){
    int x = q.front();
    cout << x<<" ";
    q.pop();
    q.push(x);
  };
  cout<<endl;
  return;
}

void reverse_first_k(queue<int> &q,int k){
  stack<int> st;
  for(int i=1;i<k;i++){
    st.push(q.front());
    q.pop();
  }
  while(st.size()>0){
    q.push(st.top());
    st.pop();
  }
  for(int i=0; i<=q.size()-k; i++){
    int x = q.front();
    q.pop();
    q.push(x);
  }
  return;
}

int main(){
  queue<int> q;
  q.push(1);
  q.push(2);
  q.push(3);
  q.push(4);
  q.push(5);
  q.push(6);
  display(q);
  reverse_first_k(q,3);
  cout<<endl;
  display(q);
  return 0;
}
```

Code Breakdown:

1. **Header Files:**
   - <iostream>: Provides input/output functionalities.
   - <stack>: Used for reversing the first **k** elements.
   - <queue>: The main data structure used for the problem.
2. **display(queue<int> &q)**
   - Iterates through the queue to print all elements.

    - Uses a loop that dequeues and enqueues each element to maintain the original order.

3. **reverse_first_k(queue<int> &q, int k)**
    - Uses a stack to store the first **k** elements (helps in reversing them).
    - Pops elements from the queue and pushes them onto the stack.
    - Pops from the stack and enqueues them back into the queue, reversing the first **k** elements.
    - The remaining elements are moved back to the queue to maintain their original order.

4. **main() Function:**
    - Creates a queue and enqueues values **1 to 6**.
    - Displays the original queue.
    - Calls reverse_first_k(q, 3) to reverse the first **3** elements.
    - Displays the modified queue.

Thought Process Behind the Code:

- **Using a Stack for Reversal:** Since stacks operate in **LIFO** order, they help in reversing elements efficiently.
- **Maintaining Queue Order:** After reversing the first **k** elements, we need to shift the remaining elements to maintain their original relative positions.
- **Time Complexity:**
    - **Pushing first k elements into the stack:** O(k)
    - **Popping from stack and pushing back to queue:** O(k)
    - **Reordering remaining elements:** O(n-k)
    - **Overall Complexity:** O(n)

Conclusion: This implementation correctly reverses the first **k** elements of a queue while keeping the rest unchanged. The use of a stack ensures efficient reversal, and queue operations maintain order for the remaining elements. This approach is optimal and well-suited for solving this problem efficiently.