Analysis for Stack::push():

```
bool Stack::push(int &newElement)
{
    if (isEmpty())
    {
        StackNode *temp = new StackNode;
        temp->data = newElement;
        temp->next = NULL;
        head->next = temp;
        return true;
    }
    else
    {
        StackNode *temp = new StackNode;
        StackNode *toInsert = new StackNode;
        temp = head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = toInsert;
        toInsert->data = newElement;
        toInsert->next = NULL;
        return true;
    }
}
```

When an element is inserted into an empty stack, a temporary node is created and is assigned values. The next member of head is initialised as the temporary node, and a Boolean value is returned.

This process amounts to 5*O(1) operations.

When an element is inserted into the stack, a temporary node is created which starts at the head, and then traverses the stack all the way to the end, where the new element is inserted.

The process of creating a temporary node, a node to insert the element, and assigning head node to temporary node all account for O(1), making our total 3*O(1)

In this process, the temporary node visits all existing stack elements at least once.
If there are ‘n’ number of elements present in the stack, this process takes ‘n’ number of operations to visit each one till it reaches the end of the stack.

This process amounts to n*O(1) operations, equating to O(n) time complexity.

After it reaches the end, it takes 3 operations to assign values to the nodes, and 1 operation to return a Boolean value.

This process amounts to 4*O(1) time complexity.

Therefore, we can assign the following execution times to these steps:

Inserting the first node: 5*O(1)
Inserting a node after the first one: O(n) + 4*O(1)

Now, since we are inserting ‘n’ nodes, the execution time would be as follows:

5*O(1) (for first node) + (n-1)*(O(n) + 4*O(1)) (for the remaining n-1 nodes)

= $O(5) + O(n^2 – n) + O(4n - 4)$

From this equation, we can extrapolate that a reasonable estimate for the time complexity for large values of ‘n’ would be $O(n^2)$.

Therefore,
Time complexity = $O(n^2)$.

Analysis for Stack::pop():

```cpp
bool Stack::pop()
{
    if (isEmpty())
    {
        cout << "Underflow" << endl;
        return false;
    }

    else
    {
        StackNode *temp = NULL;
        temp = head;
        while (temp->next->next != NULL)
        {
            temp = temp->next;
        }
        delete (temp->next);
        temp->next = NULL;
        return true;
    }
}
```

If the Stack is not empty, a temporary node is created and is set to head.

This accounts for 2*O(1) operations = O(2).

Then, each element in the stack is visited once until the penultimate element is reached, and temp is assigned the value of the penultimate element.

This accounts for (n-1)*O(1) = O(n-1) operations.

Then, the last element is deleted, and the next pointer for the penultimate element is set to NULL.

This accounts for 2*O(1) = O(2) operations

Finally, a Boolean value is returned, accounting for O(1) operations.

This brings the total to: O(2) + O(n-1) + O(2) + O(1) operations = O(n + 3) operations.

This operation is carried out for 'n' number of elements, bringing our total to:
n*O(n + 3) = O(n^2+3n)

A reasonable estimate for the time complexity for large 'n' should be O(n^2).

Therefore,
Time Complexity = $O(n^2)$