Project 3: Practical analysis of algorithms

One way to analyze algorithms practically is to add in counters in the algorithms to get the number of the quantities that we are interested, e.g., the number of data comparisons, and then compare these results to the known functions, such as $n, n \log n, n^2$, etc, to find out the exact and/or upper bounds for these quantities. We can then choose the least expansive ones based on the orders of these magnitudes.

1 What to do?

1. Collect the codes for the sorting programs that you developed for the previous project, i.e., BubbleSort, InsertionSort, MergeSort, and SelectionSort.

2. Make necessary changes so that those algorithms will sort out a list of integers.

3. Add in counters in appropriate places within these algorithms to keep respective records for, e.g., data comparisons and data movement. For example, the following shows how to add in the counters into the insertion sort algorithm to get the number of comparisons with $C_{\text{Insertion}}$ and movements with $M_{\text{Insertion}}$.

```cpp
INSERTION-SORT(A)
for j <- 2 to length[A]
  do
    key <- A[j]
    //Increment the counter for move
    M_{\text{Insertion}}++
    i <- j-1
    //Increment the counter for comparisons for the very first one
    C_{\text{Insertion}}++
    while i>0 and A[i]>key
      do
        A[i+1]<-A[i]
        //Increment the counter for move
        M_{\text{Insertion}}++
        i<-i-1
```

Several of your could not work out the mergesort code. In this case, feel free to use the codes that Steve and Nathan sent in, which I posted on the Project page.

You either create lists of integer type; or change the type Comparable in the algorithms to int.
The following shows the initialization of both counters.

1. **Generate a list of integers, list**
2. \( C_{\text{Insertion}} = 0 \)
3. \( M_{\text{Insertion}} = 0 \)
4. **INSERTION-SORT(list)**
5. **Print** \( C_{\text{Insertion}} \)
6. **Print** \( M_{\text{Insertion}} \)

4. Use the random number generator that Java provides to come up with test data of decent size, e.g., \( n = 10, 20, 50, 100, 200, 500, 1000 \).

5. To iron out probable noise, run the respective sorting algorithm 5,000 times \(^3\) for each value of \( n \), and collect the average value of the data comparison and data movement numbers, in a spreadsheet. To further clarify your result, you might want to make use of the chart mechanism.

6. Compare the data that you collected in the previous step with the following theoretical data. They might not be identical, but should be pretty close.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comparison</th>
<th>Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>( \frac{n^2}{2} + O(n) )</td>
<td>See the next assignment</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>( \frac{n^2}{2} + O(n) )</td>
<td>( \frac{n^2}{2} + O(n) )</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>( \frac{n^2}{2} + O(n) )</td>
<td>( 3n + O(1) )</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>( n \log(n) + O(n) )</td>
<td>( n \log(n) + O(n) )</td>
</tr>
</tbody>
</table>

7. The expression of the average number of data movement for the bubble sort is quite messy. This is another place when practical analysis is valuable.

Find out the average number of movements made in the bubble sort for \( n = 10, 20, 50, 100, 200, 500, 1000, 2000, 5000 \), compare the growth rate of the results with that of \( n, n \log n \), and \( n^2 \), and make an educated guess as what is the closest match of the average number of movements associated with the bubble sort algorithm out of the aforementioned bounds.

---

\(^3\)Notice that \( 10! = 3,628,800 \).
2 What should be handed in?

Email me the source code, as well as a lab report containing the data and their comparison with the theoretical results; together with your educated guess of the exact bound for the average number of the data movement as associated with the bubble sort algorithm, and its justification.