Example Sorting Algorithms

Dror Baron

This supplement provides some more details about two example sorting algorithms.

**Insert sort:** The main idea in insert sort is that we maintain a list of numbers processed so far sorted, and new numbers are inserted by scanning into the list. As an example, consider the input \( x = (1, 4, 2, -3, 7, 2, 10, 5) \). Let’s walk through the steps one by one.

- The first input is 1, and the list is \((1)\).
- The second input is 4, and the list is \((1, 4)\).
- The third input is 2, and the list is \((1, 2, 4)\).
- The fourth input is \(-3\), and the list is \((-3, 1, 2, 4)\).
- The fifth input is 7, and the list is \((-3, 1, 2, 4, 7)\). Because the new number was largest, we had to scan down the entire list.
- The sixth input is 2, and the list is \((-3, 1, 2, 4, 7)\).
- The seventh input is 10, and the list is \((-3, 1, 2, 4, 7, 10)\). Because the new number was largest, we had to again scan down the entire list.
- The eighth input is 5, and the list is \((-3, 1, 2, 2, 4, 5, 7, 10)\). The new number wasn’t largest, but was among the larger ones, and we had to scan quite a bit.

If the numbers happen to appear in sorted (or almost sorted) order, each one is inserted to the list in constant time, and the best case complexity is \(\Omega(n)\). However, in general each insert may require \(\Theta(n)\) computation, and with \(n\) such inserts the total computational complexity could be as large as \(O(n^2)\).

**Merge sort:** If the input is “short enough” we sort it (any technique will be reasonably efficient for short inputs). Else we partition the input into two sets, apply merge sort to each one recursively, and conclude by merging the outputs.

- Because the input is of length 8, we partition into \((1, 4, 2, -3)\) and \((7, 2, 10, 5)\).
- We process the first sub-problem recursively. The input is \((1, 4, 2, -3)\), and because it is relatively short suppose that we sort it using some brute force approach. The sorted output is \((-3, 1, 2, 4)\).
• We process the second sub-problem recursively. The input is (7, 2, 10, 5). We sort it using some brute force approach, resulting in the sorted output, (2, 5, 7, 10).

• We merge (−3, 1, 2, 4) and (2, 5, 7, 10) into (−3, 1, 2, 2, 4, 5, 7, 10). (This is the same output obtained earlier using insert sort.) Notably, if each of the outputs of the two sub problems that were solved recursively is of length $n$, then the merging subroutine can be implemented in linear time.

It can be shown that this divide and conquer algorithm for sorting requires $\Theta(n \log(n))$ running time. Contrast this computational complexity with the $\Omega(n)$ best case and $O(n^2)$ worst case (indeed, the typical running time is $O(n^2)$) of insert sort. Overall, merge sort provides a manageable running time with a greatly improved worst case.