We look at some simple ways to sort and their run-times.

1. How to compare values. Consider how to swap items in a list.

2. Insertion sort.
   
   (a) Place a new value at the right of a sorted list. Now, push the value to its appropriate position to the left.
   
   (b) After each push, the list is fully sorted.

   
   (a) We can sort lists of items that are totally ordered: for any pair of values \(a\) and \(b\), either \(a \leq b\) or \(b \leq a\). Exactly when \(a \leq b \leq a\) we have \(a == b\).
   
   (b) Suppose we are given a list of (totally ordered) values. Then we have \(1[i] \leq 1[i+1]\) or \(1[i] > 1[i+1]\) for all \(0 \leq i < n-1\). Bubble sort simply makes passes through the array, comparing adjacent values, and swapping those out of order.
   
   (c) After each pass, we note the largest value encountered moves completely to the right.

4. A few notes on big-O notation.
   
   (a) We can frequently bound above some performance statistic by a mathematical function of problem size. Computer scientists are not concerned about the exact performance, but the dominant trend suggested by a curve.
   
   (b) When a function is a sum of several components, we ignore all by that component that is dominant in the limit. For polynomials, we select the leading term. Thus, a program that takes \(n^2 + \log n\) time is described as an \(O(n^2)\) algorithm.
   
   (c) Additionally, we generally don’t concern ourselves with multiplicative constants. They can generally be ignored. Thus a program that makes use of \(5n\) storage locations is described as \(O(n)\) (“linear”) in its space use.
   
   (d) Many of our early programs (“Hello world”, “counting to 10”) have used constant amounts of space. We write a constant bound as \(O(1)\).
   
   (e) Programs that print out tables of simple calculations (like “counting to 10”) frequently take \(O(n)\) or “linear” amounts of time.
   
   (f) Tractable problems using one computer are typically limited to running time that is a small power of the problem size. For example, multiplying two \(n \times n\) matrices seems to take no more than \(O(n^{2.5})\) time, and requires no more than \(O(n^2)\) space.
   
   (g) The total number of digits printed is \(O(n \log n)\). Most fast sorts of \(n\) values take \(O(n \log n)\) time when using one computer.

⋆