CIS 575. Introduction to Algorithm Analysis
Material for January 19, 2024

A Top-Down Sorting Algorithm

1 Top-Down Approach

This course has frequently been taught using Rodney Howell's textbook

> *Algorithms: A Top-Down Approach*

which explains its title:

> Solving a problem by reducing it to one or more simpler problems is the essence of the top-down approach to designing algorithms.

We shall now show how to apply that approach to the sorting problem.

2 Top-Down Sorting

Given an array $A[1..n]$ to be sorted, we observe:

- if $n = 1$, or even $n = 0$ (the array is empty), we do not need to do any work
- if $n > 1$, we can sort the array as follows:
  1. first solve a simpler (smaller) problem: sort the first $n - 1$ elements of A
  2. next put the last element of A into its proper spot.

What we have described above is the essence of the well-known *insertion sort* algorithm (which is not the fastest way to sort, but surely the conceptually simplest!) We can express (leaving the procedure `InsertLast` for later) the algorithm in pseudocode:

```
Input: A[1..n] is an array of numbers.
Output: A[1..n] is a permutation of its original values
            such that A[1..n] is non-decreasing.
InsertionSort(A[1..n])
  if n > 1
    InsertionSort(A[1..n − 1])
    InsertLast(A[1..n])
```

Let us explain some parts of the notation which may be unfamiliar:

- a conditional may not have an explicit **else** branch, in which case that branch is an implicit **skip**
- we use indentation, rather than explicit delimiters, to determine the scope of constructs (thus the **then** branch of the above conditional encompasses *two* procedure calls).

When an array is passed as a parameter to a procedure, the convention is that

- we pass a *pointer* to the array, rather than make a copy (which would be very expensive, and prevent the original from being modified)
- we provide the area of the array that the called procedure may access (the code for InsertionSort must *not* read or write $A[n+1]$).