Based on K. H. Rosen: Discrete Mathematics and its Applications.

Lecture 13: Sorting and Greedy Algorithms. Section 3.1

1 Sorting

We continue studying the pseudocode associated to several algorithms, including sorting algorithms and some optimizations problems.

1.1 The Bubble sort

The **bubble sort** is one of the simplest sorting algorithms, but not one of the most efficient. It puts a list into increasing order by successively comparing adjacent elements, interchanging them if they are in the wrong order. To carry out the bubble sort, we perform the basic operation, that is, interchanging a larger element with a smaller one following it, starting at the beginning of the list, for a full pass.

procedure bubblesort($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $i = 1$ **to** $n - 1$:
    for $j = 1$ **to** $n - i$:
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
[$a_1, a_2, \ldots, a_n$ is in increasing order]

1.2 The insertion sort

The **insertion sort** is a simple sorting algorithm. To sort a list with n elements, the insertion sort begins with the second element. The insertion sort compares this second element with the first element and inserts it before the first element if it does not exceed the first element and after the first element if it exceeds the first element. At this point, the first two elements are in the correct order.

procedure insertion sort($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j = 2$ **to** n:
    $i = 1$
    while $a_j > a_i$ (The a_j should be inserted in the ith position)
        $i = i + 1$
    $m = a_j$
    for $k = 0$ **to** $j - i - 1$: (Move list to the right to create room for inserting)
        $a_{j-k} = a_{j-k-1}$
    $a_i = m$ (insert at position i if not there already)
[$a_1, a_2, \ldots, a_n$ is in increasing order]

Remark 1. Suppose that we have a random list of n numbers. The number of comparison in the bubble sort is approximately $\frac{n^2}{2}$. On the other hand, the number of comparisons for the insert sort is approximately $\frac{n^2}{4}$. We can check for example Bubble sort vs Insertion sort.

1.3 Greedy algorithms

We are going to study **optimization problems**. Optimization problems usually minimizes or maximizes the value of some parameter. One of the simplest approaches to an optimization problem often leads to a solution. This approach selects the best choice at each step, instead of considering all sequences of steps that may lead to an optimal solution. Algorithms that make what seems to be the "best" choice at each step are called **greedy algorithms**. Let us see it with one example:
Consider the problem of making n cents change with quarters, dimes, nickels, and pennies, and using the **least total number of coins**. We can devise a greedy algorithm for making change for n cents by making a locally optimal choice at each step; that is, at each step we choose the coin of the largest denomination possible to add to the pile of change without exceeding n cents. The pseudocode for the Greedy Make-Change algorithm could be written as:

procedure greedy make-change($c_1, c_2, \ldots, c_r$: values of denominations of coins,
    where $c_1 > c_2 > \cdots > c_r$ and n: a positive integer)
for $i = 1$ **to** r:
    $d_i = 0$ (d_i counts the coins of denomination c_i used)
    while $n \geq c_i$
        $d_i = d_i + 1$ (add a coin of denomination c_i)
        $n = n - c_i$
(d_i is the number of coins of denomination c_i in the change for $i = 1, 2, \ldots, r$)

Remark 2. If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies, and cannot have two dimes and a nickel. The amount of change in dimes, nickels, and pennies cannot exceed 24 cents. The greedy algorithm with these denominations **is optimal** in the sense that it produces change using the fewest coins possible. If we consider instead only quarters, dimes, and pennies, we can show the greedy algorithm is not optimal.