SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2025

Topic Notes: Brute-Force Algorithms

Our first category of algorithms are called *brute-force algorithms*.

Levitin defines brute force as a straightforward approach, usually based directly on the problem statement and definitions of the concepts involved.

We have already seen a few examples:

- consecutive integer checking approach for finding a GCD
- matrix-matrix multiplication

Another is the computation of a^n by multiplying by a n times.

Brute-force algorithms are not usually clever or especially efficient, but they are worth considering for several reasons:

- The approach applies to a wide variety of problems.
- Some brute-force algorithms are quite good in practice.
- It may be more trouble than it's worth to design and implement a more clever or efficient algorithm over using a straightforward brute-force approach.

Brute-Force Sorting

One problem we will return to over and over is that of *sorting*. We will first consider some brute-force approaches.

We will usually look at sorting arrays of integer values, but the algorithms can be used for other comparable data types.

Bubble Sort

Right at the start of our class, we looked at this very intuitive sort. We just go through our array, looking at pairs of adjacent values, and swapping them if they are out of order.

It takes $n - 1$ "bubble-ups", each of which can stop sooner than the last, since we know we bubble up one more value to its correct position in each iteration. Hence the name *bubble sort*.

The version we came up with earlier looked like this:

ALGORITHM BUBBLESORT(A)
  //Input: an array $A[0..n-1]$
  for $i \leftarrow 0..n-2$ **do**
    for $j \leftarrow 0..n-2-i$ **do**
      if $A[j+1] < A[j]$ **then**
        swap $A[j+1]$ and $A[j]$

We will analyze this in class to see (as expected) that it has $\Theta(n^2)$ comparisons. There is also a swap, potentially, after each comparison, giving a worse case behavior of $\Theta(n^2)$ swaps.

Selection Sort

A simple improvement on the bubble sort is based on the observation that one pass of the bubble sort gets us closer to the answer by moving the largest unsorted element into its final position. Other elements are moved "closer" to their final position, but all we can really say for sure after a single pass is that we have positioned one more element.

So why bother with all of those intermediate swaps? We can just search through the unsorted part of the array, remembering the index of (and hence, the value of) the largest element we've seen so far, and when we get to the end, we swap the element in the last position with the largest element we found. This is the *selection sort*.

ALGORITHM SELECTIONSORT(A)
  //Input: an array $A[0..n-1]$
  for $i \leftarrow 0..n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1..n-1$ **do**
      if $A[j] < A[min]$ **then**
        $min \leftarrow j$
    swap $A[i]$ and $A[min]$

We will complete the analysis in class for both the number of comparisons and the number of swaps.

Here, we do the same number of comparisons, but at most $n-1 \in \Theta(n)$ swaps.

Sequential Search

This topic is covered in its entirety on the class handout.

Brute-Force String Match

The *string matching* problem involves searching for a *pattern* (substring) in a string of *text*.

The basic procedure:

1. Align the pattern at beginning of the text
2. Moving from left to right, compare each character of the pattern to the corresponding character in the text until
   - all characters are found to match (successful search); or
   - a mismatch is detected
3. While pattern is not found and the text is not yet exhausted, realign the pattern one position to the right and repeat Step 2

The result is either the index in the text of the first occurrence of the pattern, or indices of all occurrences. We will look only for the first.

Written in pseudocode, our brute-force string match:

ALGORITHM BRUTEFORCESTRINGMATCH(T, P)
  //Input: a text string $T[0..n-1]$
  //Input: a pattern string $P[0..m-1]$
  for $i \leftarrow 0..n-m$ **do**
    $j \leftarrow 0$
    while $j < m$ **and** $P[j] = T[i+j]$ **do**
      $j \leftarrow j+1$
    if $j = m$ **then**
      return i
  return -1

Analysis will be done in class.

We will consider improvements for string matching later in the semester.

In class exercise: Exercise 3.1.4, p. 102