csce350 — Data Structures and Algorithms
Fall 2019 — Lecture Notes: Brute Force

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with occasional supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 Introduction

Brute force is a straightforward approach to solving a problem, usually directly based on the 3
problem statement and the definitions of the concepts involved.

2 Example: Integer Powers

Suppose we have two positive integers a, n and we want to compute a^n.

```
INTEGERPOWER(a, n)
  r ← 1
  for i ← 1, ..., n do
    r ← r · a
  end for
  return r
```

3 The sorting problem

Sorting refers to the problem of rearranging an array so that its elements are in order.

- **Input:** An array of numbers $A[0, \ldots, n-1]$.
- **Output:** A reordering $A'[0, \ldots, n-1]$ such that

$$A'[0] \leq A'[1] \leq \cdots \leq A'[n].$$

Who cares?

- practically important
- useful for illustrating many recurring ideas in algorithms

Note that the idea of "sorting" is not restricted to just numbers. As long as the elements can be compared to each other —that is, as long as $<$ and $>$ make sense, then the problem is still well defined. We'll use numbers through this course because they make the intuition very easy.

4 Selection sort

Observation: In a sorted list, the smallest element comes first. 3.1

Algorithm idea: Find the smallest element and put it first. Then repeat.

5 Selection sort

```
SELECTIONSORT(A[0,...,n−1])
  for i ← 0,...,n−2 do
    m ← i
    for j ← i+1,...,n−1 do
      if A[j] < A[m] then
        m ← j
      end if
    end for
    swap A[i] and A[m]
  end for
```

6 Selection sort analysis

$$
\begin{aligned}
C(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-2} \left(n-i-1\right) \\
&= \sum_{i=0}^{n-2}(n-1) - \sum_{i=0}^{n-2} i \\
&= (n-1)(n-2-0+1) - \frac{(n-2)(n-1)}{2} \\
&= \cdots = \frac{n(n-1)}{2} \in \Theta(n^2)
\end{aligned}
$$

Hint: Identical to analysis of ELEMENTSUNIQUE.

7 The string matching problem

String matching problem: 3.2

- **Input**: Two strings: A pattern $P[0,\ldots,m-1]$ and a text $T[0,\ldots,n-1]$.
- **Output**: An index in T at which P appears, or "no match" if P does not appear in T.

8 Brute force string matching

Algorithm idea: Check each potential starting position for P within T. If we find a mismatch, move on to the next potential starting position.

```
BRUTEFORCESTRINGMATCH(T[0,...,n−1], P[0,...,m−1])
  for i ← 0,...n − m do
    j ← 0
    while j < m and T[i + j] = P[j] do
      j ← j + 1
    end while
    if j = m then
      return i
    end if
  end for
  return 'no match'
```

9 Brute force string matching analysis

$$C(m,n) = \sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1 = \sum_{i=0}^{n-m} m = m(n-m+1) \in \Theta(mn)$$

10 Traveling salesman problem

Problem: Find the shortest cycle that visits every node in a complete weighted graph. 3.4

11 Solving TSP

Algorithm idea: Exhaustive search. Try all permutations of the n nodes.

12 Exhaustive search for TSP: Analysis

This is **really** slow!
$\Theta(n \cdot n!)$

13 Knapsack problem

Input:

- n items, each with a **weight** and a **value**.

$$\begin{matrix} w_1 & w_2 & \cdots & w_n \\ v_1 & v_2 & \cdots & v_n \end{matrix}$$

- A knapsack with **capacity** W.

Output: A list of items to take that maximizes total value within the capacity constraint.

14 Solving the Knapsack Problem

Algorithm idea: Exhaustive search. Try all subsets of the n items. Select the best one.