Design and Analysis of Algorithms, Honors

Raymond Bian

January 9, 2024

Contents

Lecture 1: Intro and Review

1 Intro to Algorithms

1.1 Properties of Algorithms

What kind of properties can an algorithm even have?

Property. Algorithms can be categorized by speed, memory, readability (simple to understand), accuracy (approximation quality), and requirements for the input.

The focus of this class will be on **speed**. We will also talk about accuracy, and algorithms will be mostly readable (but not always). We need some language that will allow us to describe algorithms.

1.2 Describing Algorithms

Example. Let's take for example selection sort.

Proof. Using plain english, selection sort is: reapeatedly finding the smallest element and appending it to the output.

Algorithms can be described at different levels of detail. One extreme is very informally, like the sentence above. This type helps convey the **key concepts**, but can be rather vague, ambiguous, and hard to understand.

The other extreme could be posting the source code of the program. This would be a complete description of an algorithm, because even the computer and execute it. However, source code includes details that **only the computer needs**, such as types, etc.

So, when talking about algorithms, it makes sense to use a middleground, known as **pseudocode**.

Definition 1. **Pseudocode** is a descriptive, step by step set of instructions that detail the structure of a program. English is allowed, and the level of detail depends on the context and the audience.

Selection Sort 1 Pseudocode for Selection Sort

Any array of elements *input*
A sorted array *output*
1: **for** $i \leftarrow 1 \dots n$ **do**
2: Find $j \geq i$ with smallest $arr[j]$
3: Swap $arr[i]$ and $arr[j]$
4: **end for**

More detailed descriptions can also be given with pseudocode.

1.3 Runtime of Algorithms

Many factors can impact the runtime of an algorithm. For example, Selection Sort, when ran on the professor's laptop had a runtime of $0.017n^2 + 0.669n + 0.114$ms. This runtime will vary depending on the context, things like hardware, the programming language, and temperature.

Because we cannot determine these constants for every computer, we will just **ignore them** for analyzing algorithms.

Definition 2. **Big-O Notation** is the notation used to analyze runtimes. It essentially ignores unknown constant factors. More formally, it states that $f(n) = O(g(n))$ if and only if there exists n_0 and $c > 0$ such that $f(n) \leq c \cdot g(n)\ \forall n \geq n_0$.

Lemma 1. $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = O(g(n))$.

We can find the runtime of algorithms by counting the number of operations.

Example. Selection sort runs in $O(n^2)$.

Proof. We have

$$\sum_{i=1}^{n}\sum_{j=i}^{n} 1 = \frac{n(n-1)}{2} = O(n^2).$$

Note that the second equality comes from taking the limit.

Definition 3. **Omega** notation denotes the relation $\geq$.

Lemma 2. $\lim_{n\to\infty} \frac{g(n)}{f(n)} \Rightarrow f(n) = \Omega(g(n))$.

Definition 4. **Little-O** notation denotes the strict relation $<$.

Lemma 3. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$.

Definition 5. **Little Omega** notation denotes the strict relation $>$.

Lemma 4. $\lim_{n\to\infty} \frac{g(n)}{f(n)} \Rightarrow f(n) = \omega(g(n))$.

Definition 6. **Theta** notation denotes the equality relation $=$.

Lemma 5. $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n)) \Rightarrow f(n) = \Theta(g(n))$.