CSCI 2824 - CU Boulder, 2019 Summer

Lecture 13: Algorithm Analysis

24 June 2019

Lecturer: J. Marcus Hughes

Content is borrowed from Susanna Epp's Discrete Mathematics with Applications,
Rosens's Discrete Mathematics and its Applications,
Bettina and Thomas Richmond's A Discrete Transition to Advanced Mathematics, and Andrew Altomare's notes.

1 Review

1.1 Function problem

For each relation on 1, 2, 3, 4 determine if it is a function. If it is a function states its range and whether it is injective, surjective, bijective, or none.

1. $\{(1,2),(2,1),(3,4),(4,3)\}$
2. $\{(1,1),(3,1),(2,1),(4,1)\}$
3. $\{(1,1),(1,2),(1,3),(1,4)\}$
4. $\{(1,3),(2,4),(3,3),(4,3)\}$
5. $\{(1,3),(2,1),(3,2)\}$

1.2 Set Question

Let the following sets be defined as:

$$\mathcal{A} = \left\{\left(\frac{1}{n}, n+1\right) | n \in \mathbb{N}\right\}$$
$$\mathcal{B} = \left\{\left(\frac{-1}{n}, n\right) | n \in \mathbb{N}\right\}$$
$$\mathcal{C} = \{(n,\infty)|n \in \mathbb{N}\}$$
$$\mathcal{D} = \{(x,\infty)|x \in \mathbb{R}\}$$
$$\mathcal{E} = \{\{x \in \mathbb{R}|x^2 < n\}|n \in \mathbb{N}\}$$

Then, determine whether the following statements are true or false. Here (x, y) means the interval of the real numbers that starting from x and going to y not including x and y. Notice that all of these are sets of sets!

1. $\emptyset \in \mathcal{A}$
2. $\emptyset \subset \mathcal{A}$
3. If B is any element in $\mathcal{B}$, then $0 \in B$.
4. $\mathcal{B} \subseteq \mathcal{A}$
5. $\mathcal{C} \subseteq \mathcal{D}$
6. $\mathcal{A} \cap \mathcal{E} = \emptyset$
7. $\mathcal{B} \cap \mathcal{E} = \emptyset$
8. $(-2, 2) \subseteq \mathcal{E}$
9. $(-\sqrt{3}, \sqrt{3}) \in \mathcal{E}$
10. $\cap_{e \in \mathcal{E}} e \in \mathcal{B}$

2 Algorithms

Is it harder to make a Sudoku, solve it, or check someone's solution? We'll talk about this and many other ideas in the coming days.

Definition: *Algorithm*
An algorithm is a finite sequence of precise instructions for performing a computation or solving a problem.

Thus, we might have an algorithm that finds the maximum element in a finite list:

1. Initialize `max` to the first element in the list.
2. Compare this to the second element in the list. If the second element is larger, set `max` to the second element.
3. Repeat the previous step for all the remaining list elements.
4. Stop when there are no more terms.
5. `max` is now the largest element in the list.

When we think about algorithms, we expect some properties:

- **Input**: An algorithm has inputs from a particular set.
- **Output**: From each set of inputs, the algorithm produces outputs from a particular set. The outputs are the solution to the problem the algorithm tackles.

- **Definiteness**: The steps in the algorithm are precisely defined.
- **Correctness**: We want some assurance that the output is correct for the input.
- **Finiteness**: The algorithm should not run forever.
- **Generality**: The algorithm should be applicable for all inputs of that form.

Let's think of a specific algorithm: finding a specific element in a sorted list. If the element isn't there, you say it's not. An example of this could be in a library. You want to find a specific book and thankfully the library is organized. Maybe you try two strategies: either just going book by book or maybe you split the library in half and keep narrowing down. We call the former a linear search. You start from the first element and check each element in the list until you find what you're looking for. We call the second approach a binary search. You separate the list into two halves and see which half should contain the element you're looking for using its sorted property. You keep halving until you're stuck with one element only.

Let's try both to find the element 15 in a list $[1, 2, 7, 11, 15, 19, 30, 31]$. It might seem silly since you can just look at it and find the answer, but we do things like this all the time in computers. First, let's use a linear search.

1. Look at 1. It's not the element.
2. Look at 2. It's not the element.
3. Look at 7. It's not the element.
4. Look at 11. It's not the element.
5. Look at 15. We found it. Hurray!

Now, we'll try the binary search.

1. Break the list into two halves: $[1, 2, 7, 11]$ and $[15, 19, 30, 31]$.
2. Since $11 < 15$ it has to be in the right half. Let's split again $[15, 19]$ and $[30, 31]$.
3. Since $15 < 19$ it has to be in the left half. Split again into $[15]$ and $[19]$.
4. HEY! We found 15! CELEBRATE!

Which one of these searches is faster? That's a trick question since we really haven't defined what we mean by faster. We could mean the following:

- Which is faster in the *best-case* scenario?
- Which is faster in the *worst-case* scenario?
- Which is faster on *average*?

So, we have to figure out the best, worst, and average case scenarios somehow. Let's count how many *operations* it takes to perform each search.

3 Asymptotic Notation

Definition: *Asymptotic Notation*
Let f and g be real-valued functions defined on the same set of nonnegative integers with $g(n) \geq 0$ for every integer $n \geq r$, where r is a positive real number. Then:

- f is of order at least g, written $f(n)is\Omega(g(n))$ (f of n is big-Omega of g of n) if and only if, there exist positive real numbers A and $a \geq r$ such that $Ag(n) \leq f(n)$ for every integer $n \geq a$.

- f is of order at most g, written $f(n)isO(g(n))$ (f of n is big-O of g of n), if, and only if, there exist positive real numbers B and $b \geq r$ such that $0 \leq f(n) \leq Bg(n)$ for every integer $n \geq b$.

- f is of order g, written $f(n)is\Theta(g(n))$, (f of n is big-Theta of g of n), if, and only if, there exit positive real numbers A, B, and $k \geq r$ such that $Ag(n) \leq f(n) \leq Bg(n)$ for ever integer $n \geq k$.

Graphically, this looks like Figure 1.

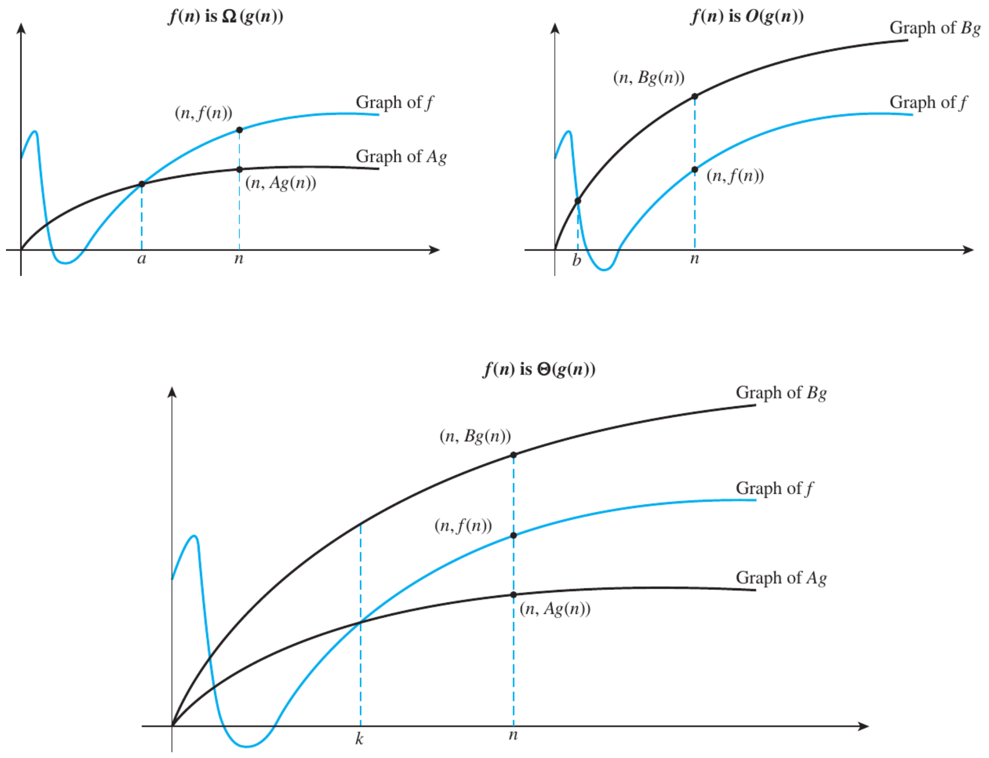

Figure 1: Examples of asymptotic notation.

We'll do lost of problems now.