Analysis & Design of Algorithms
Introduction

Angel Napa

January 1, 2026

1 Motivation & some definitions

Definition 1. *Algorithm: An explicit, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose*

Definition 2. *Computational problem: Mapping between inputs and a set of outputs*

In that sense, an algorithm is a procedure that solves a computational problem. Let's look at the following example

Example 1. *Sorting problem*
Input: Sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.
Output: A permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ *of the input array, such that* $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Definition 3. *Instance of a problem: Particular case of the problem*

In this example, an instance is $\langle 31, 41, 59, 26, 41, 58 \rangle$. An algorithm is *correct* if for each input, it returns the corresponding output. In that case, we say the algorithm *solves* the problem

Definition 4. *Analysis of Algorithms: The study of the amount of computational resources: time and memory, that the algorithms consume*

The goal is to identify common structural aspects of different algorithms when studying methods and paradigms of algorithm design. These techniques are common when solving much more applied problems related to the human genome, the internet, cryptography, linear programming (optimization), etc.

We will learn different paradigms of designs. The main ones this course study are:

- Divide & Conquer
- Dynamic Program,
- Greedy Algorithms.

For NP-hard problems, more advanced techniques are used: heuristics or approximation algorithms. However, those algorithms have so much in common with the ones we will see in this course.

2 Introduction to Efficiency Analysis

The classic metrics to measure the efficiency of an algorithm are execution and memory time consumed. In this section we will see simpler, but informal execution time of two algorithms.

Example 2. *We have two computers A and B with the following characteristics and algorithms to execute.*

- *Computer A:*
  - *Speed:* 10^{10} *instructions/s,*
  - *Algorithm to execute:* InsertionSort, with complexity of $2n^2$. This means, with size n input, it executes $2n^2$ instructions.
- *Computer B:*
  - *Speed:* 10^7 *instructions/s,*
  - *Algorithm to execute:* MergeSort, with complexity of $50n \lg n$.

In the example above, assuming we want to order $n = 10^7$ numbers, we know that

- A spends $\frac{2(10^7)^2 inst}{10^{10} inst/s} = 20000s = 5.55hs$.
- B spends $\frac{50(10^7)\lg 10^7 inst}{10^7 inst/s} = 1162.67s = 0.322hs$.

In the same example, if we sort $n = 10^8$ numbers, A spends 23 days, but B only spends 4 hrs.

We can conclude that, the bigger the size of n, the faster the algorithm MergeSort is compared to the algorithm InsertionSort, making the parameters of speed and constants irrelevant. This means, the relevant expressions have to be the number of instructions the algorithms make.

Exercise 1. *Suppose the same computer runs* InsertionSort and MergeSort, where now InsertionSort has $8n^2$ complexity and Mergesort has $64n \lg n$ complexity. For what values of n, the algorithm InsertionSort is more efficient?

We could tabulate values and observe that the answer is $n \in \{2, \ldots, 43\}$.

n	$8 \lg n$
1	0
2	8
3	12.67
10	26.57
20	34.57
30	39,...
40	42.57
43	43.4
44	43.67
...	...

But this argument has some blind spots: how to make sure these are ALL the numbers that satisfy that condition? We present a more formal proof.

We say a function over natural numbers is *increasing* if $f(x) \leq f(x+1)$ for all x in the domain of f.

Proposition 1. *The function $f(n) = \frac{n}{\lg n}$ over the natural numbers starting from 3 is an increasing function*

Proof. We will show by induction in n, that $f(n) \leq f(n+1)$, for all natural numbers $n \leq 3$

If $n = 3$, we have $f(3) = \frac{3}{\lg 3} < 2 = \frac{4}{\lg 4} = f(4)$, thus, the proposition is valid. Suppose now that $n \geq 4$. By induction hypothesis, we have

$$\frac{n-1}{\lg(n-1)} \leq \frac{n}{\lg n}.$$

Hence,

$$n = \frac{1}{1 - \frac{n-1}{n}} \leq \frac{1}{1 - \frac{\lg(n-1)}{\lg n}} = \frac{\lg n}{\lg n - \lg(n-1)} = \frac{\lg n}{\lg(n/(n-1))}.$$

Then,

$$\begin{aligned}
\frac{n}{n+1} &= 1 - \frac{1}{n+1} \\
&\leq 1 - \frac{1}{\frac{\lg n}{\lg(n/(n-1))} + 1} \\
&\leq 1 - \frac{\lg(n/(n-1))}{\lg(n^2/(n-1))} \\
&= \frac{\lg n}{\lg(n^2/(n-1))} \\
&\leq \frac{\lg n}{\lg(n+1+\frac{1}{n-1})} \\
&\leq \frac{\lg n}{\lg(n+1)}.
\end{aligned}$$

□

Proposition 2. *For all positive number n, $8n^2 < 64n \lg n$ if and only if $2 \leq n \leq 43$.*

Proof. If $n = 1$ then $8n^2 = 8 > 64n \lg n$. If $n = 2$ then $8n^2 = 32 < 64n \lg n$. Suppose then that $n \geq 3$. By proposition 1, $n/\lg n$ is an increasing function. Thus, for $n \leq 43$ we have that $\frac{n}{\lg n} \leq \frac{43}{\lg 43} < 8$. And for $n \geq 44$, we have $\frac{n}{\lg n} \geq \frac{44}{\lg 44} > 8$. We conclude that $8n^2 < 64n \lg n$ if and only if $2 \leq n \leq 43$. □

An important corollary of proposition 1 is the following:

Corolary 1. *For each positive number n, it holds $n > \lg n$.*

Proof. If $n = 1$, we have $n = 1 > 0 = \lg 1$. If $n \geq 2$, by proposition 1, we have: $\frac{n}{\lg n} \geq \frac{2}{\lg 2} = 2$. Then $n \geq 2 \lg n > \lg n$. □

Observation 1. *Logarithm notation:*

- $\lg x := \log_2 x$
- $\log x := \log_{10} x$

Exercise 2. *What is the minimum value of n such that an algorithm with execution time of $100n^2$ is faster than another one with execution time of 2^n on the same machine?*

$\lg 100n^2$	n
6.64...	1
8.64...	2
9.813 ...	3
13.2877...	10
...	...
14.25...	14
14.457...	15

We can take the base 2 logarithm of both expressions and tabulate the corresponding values. It can be observed that the minimum value is indeed $n = 15$. The formal proof is left as an exercise to the reader.

Exercise 3. *Make a formal proof that the $2^n/100n^2$ is an increasing function. Use this proof to finish the above exercise.*

Observation 2. *Basic logarithm properties.*

- $\log_a b = x$ *if and only if* $b = a^x$. *(Logarithm definition).*

- $\log_a(x^y) = y \log_a(x)$.
- $\log_a(xy) = \log_a(x) + \log_a(y)$.

The following properties can be proved based on the previous ones

- $\log_a b = 1/\log_b a$.
- $\log_a b = \log_a c \cdot \log_c a$.
- $\log_a(x/y) = \log_a(x) - \log_a(y)$.
- $a^{\log_b c} = c^{\log_b a}$.
- $a^{\log_a b} = b$.

3 Induction Technique

A proof technique widely used to show that all elements of an infinite set have a specified property.

We will see 3 applications of this technique.

Property 1. *Show that for all positive number n, $1+2+3+\cdots+n = \frac{n(n+1)}{2}$.*

Proof. By induction on n. When $n = 1$, we have $1 + 2 + \ldots + n = 1 = \frac{1\cdot 2}{2} = n(n+1)/2$ (base case).

When $n > 1$, by induction hypothesis, we have that

$$1 + 2 + \cdots + (n-1) = \frac{(n-1)n}{2}.$$

Then, $1+2+\ldots+n = (1+2+\ldots+(n-1))+n = \frac{(n-1)n}{2}+n = n(\frac{n-1}{2}+1) = \frac{n(n+1)}{2}$. □

Property 2. *For each positive integer number n, $1+2+2^2+\cdots 2^n = 2^{n+1}-1$.*

Proof. By induction on n. When $n = 1$, we have $1+2+2^2+\ldots+2^n = 1+2 = 3 = 2^2-1 = 2^{n+1}-1$ (base case). When $n > 1$, by induction hypothesis, we have that

$$1 + 2 + 2^2 + \cdots 2^{n-1} = 2^n - 1.$$

Then, $1+2+2^2+\cdots+2^n = (1+2+2^2+\cdots 2^{n-1})+2^n = (2^n-1)+2^n = 2^n+2^n-1 = 2(2^n)-1 = 2^{n+1}-1$. □

Property 3. *Show that, for all positive integer n,*

$$\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{n(n+1)} = \frac{n}{n+1}$$

Proof. By induction on n. When $n = 1$, we have that $\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{n(n+1)} = \frac{1}{2} = \frac{n}{n+1}$ (base case). When $n > 1$, by induction hypothesis

$$\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{(n-1)n} = \frac{n-1}{n}$$

Then, $\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{n(n+1)} = (\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{(n-1)n}) + \frac{1}{n(n+1)} = \frac{n-1}{n} + \frac{1}{n(n+1)} = \frac{(n-1)(n+1)+1}{n(n+1)} = \frac{(n^2-1)+1}{n(n+1)} = \frac{n^2}{n(n+1)} = \frac{n}{n+1}$. □

Property 4. *Show that for all positive integer number $n \geq 1$, $\lg n \leq n$.*

Proof. We know that $f(x) = \lg x$ is an increasing function over the positive integer numbers. Then, the proposition is equivalent to show that: $n \leq 2^n$ for all natural numbers such that $n \geq 1$. We will prove this by induction on n. When $n = 1$, we have that $1 = n \leq 2^1 = 2$. When $n > 1$, by induction hypothesis we have.

$$n - 1 \leq 2^{n-1}.$$

Furthermore, $2^{n-1} \geq 2^0 = 1$. Then, $n \leq 2^{n-1} + 1 \leq 2^{n-1} + 2^{n-1} = 2^n$. □

Property 5. *Prove that, for all positive integer number $n \geq 44$, $8 \lg n \leq n$.*

Proof. We know that $f(x) = \lg x$ is an increasing function over the positive integer numbers. Then, the proposition is equivalent to show that: $n \leq 2^{n/8}$ for all natural numbers such that $n \geq 44$. We will prove this using induction on n. When $n = 44$, we have that $n^8 \leq 2^n$. When $n > 44$, by induction hypothesis, we have that

$$(n-1) \leq 2^{(n-1)/8}.$$

Also, because $n \geq 45$ we have $n/8 \geq 5$, then, $2^{-n/8} \leq 2^{-5}$.
Then

$$\begin{aligned} n &\leq 2^{(n-1)/8} + 1 \\ &\leq 2^{n/8}(2^{-1/8} + 2^{-n/8}) \\ &\leq 2^{n/8}(2^{-1/8} + 2^{-5}) \\ &\leq 2^{n/8}, \end{aligned}$$

as we wanted to prove. □

4 Sums

We can express the time of execution of an algorithm that has an iterative instruction, as the sum of the times of each iteration. This is why we have to remember the concepts of sums, and understand how to obtain upper and lower bounds of sums

4.1 Basic properties and formulas

Definition 5. *Given a sequence $a_1, a_2, \ldots, a_n$ of numbers, where n is a non-negative integer, we can write the sum $a_1 + a_2 + \cdots + a_n$ as*

$$\sum_{k=1}^{n} a_k.$$

If $n = 0$ then, the value of the sum is 0.

Linearity property

Given an arbitrary number c and two sequences $a_1, a_2, \ldots, a_n$ y $b_1, b_2, \ldots, b_n$, we have:

$$\sum_{k=1}^{n}(ca_k + b_k) = c\sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k.$$

Arithmetic Series

Series where the difference of two consecutive terms of a sequence is the same. An example:

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

Some known sums

It is known the following properties:

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}.$$

$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4} = \left(\frac{n(n+1)}{2}\right)^2.$$

Geometric series

Series where the division of two consecutive terms of a sequence is the same.

An example:
For each real number $x \neq 1$,

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}.$$

- By the previous property, if we replace $x = 2$, we have

$$1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1.$$

- When $0 < |x| < 1$, it holds that $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$.
- From all the above, we can conclude that $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$. The proof is left as an exercise to the reader.

Harmonic Series

It is the series

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n},$$

which value is H_n, the *n Harmonic number*. It can be shown that $\mathrm{H}_n \sim \ln n + c$ for a constant $c \approx 0.5772$, the *Euler-Mascheroni constant*.

4.2 Bounding sums

There are many techniques to bound sums. These will help us bound sums that represent execution times of some algorithms

Bounding by induction

We will show the following two propositions using induction

- $\sum_{k=1}^{n} k \leq \frac{1}{2}(n+1)^2$., for all positive integer n
- Exists a constant $c > 0$, such that $\sum_{k=0}^{n} 3^k \leq c3^n$, for all positive integer n.

Proposition 3. *Show that, for all positive integer $n \geq 1$,*

$$\sum_{k=1}^{n} k \leq \frac{1}{2}(n+1)^2$$

Proof. Proof using induction on n. It's easy to see that the property holds for $n = 1$. Assume by induction hypothesis that the property holds for $n = m - 1$, for some $m > 1$. This means, $\sum_{k=1}^{m-1} k \leq \frac{1}{2}m^2$. Then,

$$\begin{aligned}
\sum_{k=1}^{m} k &= \sum_{k=1}^{m-1} k + m \\
&\leq \frac{1}{2}m^2 + m \text{ (by induction hypotesis)} \\
&\leq \frac{1}{2}m^2 + m + \frac{1}{2} \\
&= \frac{1}{2}(m+1)^2,
\end{aligned}$$

We conclude the proof. □

Proposition 4. *Show that exists a constant $c > 0$ such that*

$$\sum_{k=0}^{n} 3^k \leq c3^n$$

holds for all positive integer n.

Proof. We will show by induction on n, that for all natural number $n \geq 0$,

$$\sum_{k=0}^{n} 3^k \leq \frac{3}{2} \cdot 3^n.$$

This means, we will prove that the property holds for $c = 1.5$. When $n = 0$, we have $\sum_{k=0}^{n} 3^k = 1 \leq \frac{3}{2} \cdot 3^0$, thus, the base case holds. When $n > 0$, by induction hypothesis, we have $\sum_{k=0}^{n-1} 3^k \leq \frac{3}{2} \cdot 3^{n-1}$. Then,

$$\begin{aligned}
\sum_{k=0}^{n} 3^k &= \sum_{k=0}^{n-1} 3^k + 3^n \\
&\leq \frac{3}{2} \cdot 3^{n-1} + 3^n \\
&= \left(\frac{1}{3} + \frac{2}{3}\right) \frac{3}{2} \cdot 3^n \\
&\leq \frac{3}{2} \cdot 3^n.
\end{aligned}$$

□

Bounding terms

We can upper bound each term of the sequence. This way we can build an upper bound the series. Let's see some examples.

- $\sum_{k=1}^{n} k \leq \sum_{k=1}^{n} n = n^2$
- Let $a_{\max} = \max_{1 \leq k \leq n} a_k$. Then, $\sum_{k=1}^{n} a_k \leq \sum_{k=1}^{n} a_{\max} = n \cdot a_{\max}$.
- Suppose that $\frac{a_{k+1}}{a_k} \leq r$ for all $k \geq 0$, con $0 < r < 1$. We have that

$$\begin{aligned}
\sum_{k=0}^{n} a_k &\leq \sum_{k=0}^{\infty} a_0 r^k \\
&= a_0 \sum_{k=0}^{\infty} r^k \\
&= a_0 \cdot \frac{1}{1-r}.
\end{aligned}$$

- A particular exercise can be formed with the result above. Let's bound

$$\sum_{k=1}^{\infty} \frac{k}{3^k} = \sum_{k=0}^{\infty} \frac{k+1}{3^{k+1}}$$

We have that $a_0 = 1/3$. Also, $\frac{\frac{k+2}{3^{k+2}}}{\frac{k+1}{3^{k+1}}} = \frac{1}{3} \cdot \frac{k+2}{k+1} \leq \frac{2}{3}$. Thus, we can take $r = 2/3$, obtaining

$$\sum_{k=1}^{\infty} (k/3^k) \leq \frac{1}{3} \cdot \frac{1}{1-2/3} = 1.$$

Partitioning sums

- We will lower bound the sum $\sum_{k=1}^{n} k$. Note that (assuming that n is even)

$$\begin{aligned} \sum_{k=1}^{n} k &= \sum_{k=1}^{n/2} k + \sum_{k=n/2+1}^{n} k \\ &\geq \sum_{k=1}^{n/2} 0 + \sum_{k=n/2+1}^{n} (n/2) \\ &= (n/2)^2. \end{aligned}$$

- For each constant $k_0 > 0$, we have that

$$\begin{aligned} \sum_{k=0}^{n} a_k &= \sum_{k=0}^{k_0-1} a_k + \sum_{k=k_0}^{n} a_k \\ &= c + \sum_{k=k_0}^{n} a_k. \end{aligned}$$

Where c is a constant

- Let's bound $\sum_{k=0}^{\infty} k^2/2^k$. Note that, when $k \geq 3$, $\frac{(k+1)^2/2^{k+1}}{k^2/2^k} = \frac{(k+1)^2}{2k^2} \leq \frac{8}{9}$. Then,

$$\begin{aligned} \sum_{k=0}^{\infty} k^2/2^k &= \sum_{k=0}^{2} k^2/2^k + \sum_{k=3}^{\infty} k^2/2^k \\ &\leq \sum_{k=0}^{2} \frac{k^2}{2^k} + 9/8 \sum_{k=0}^{\infty} (8/9)^k \\ &= \sum_{k=0}^{2} \frac{k^2}{2^k} + 9/8 \cdot (1/(1-8/9)) \\ &= c \end{aligned}$$

for some constant c.

- Let's bound $H_n = \sum_{k=1}^{n} \frac{1}{k}$. Note that

$$\begin{aligned}
\sum_{k=1}^{n} \frac{1}{k} &\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j} \\
&\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i} \\
&= \sum_{i=0}^{\lfloor \lg n \rfloor} 1 \\
&\leq \lg n + 1
\end{aligned}$$

4.3 Binary tree

A *binary tree* T can be defined recursively as follows.

- If T has zero nodes, then is a binary tree, else
- T is composed of 3 sets of disjoint set of nodes, a *root* node, a binary tree called its *left subtree* and a binary tree called its *right subtree*.

A binary node with zero nodes is called *null* or *empty* and denoted by NIL. If the left subtree is nonempty, its root is called *left child*. The same way we can define *right child*. A *leaf* if a node with no children.

An *internal node* is a node that is not a leaf. The *depth* of a node is the length of the node to the root.

A binary tree is *full* if each internal node has two children. A binary tree is *complete* if it is full, and each leaf has the same depth.

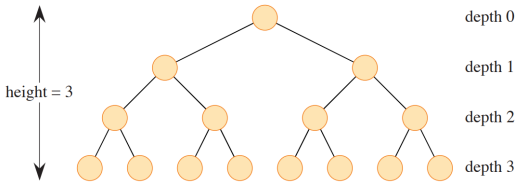

Figure B.8 A complete binary tree of height 3 with 8 leaves and 7 internal nodes.

Figure 1: Cormen, Introduction to Algorithms

The *height* of a node is the number of edges of the longest simple downward path from the node to a leaf. The *height* of a tree is the height of its root.

Property 6. *The height of a complete binary tree with k leaves is* $\lg k$.

Property 7. *The height of a complete binary tree with n nodes is* $\lg(n+1) - 1$