Run Time, Generating Functions

CS 624 — Analysis of Algorithms

September 13, 2024

Order of Growth

Abstractions from concrete performance numbers:

- Ignore hardware platform, caches, different instructions.
- Ignore differences in constant numbers of instructions.
- Ignore constant factors in general.
- Ignore performance for "small" problem sizes.

What is left?

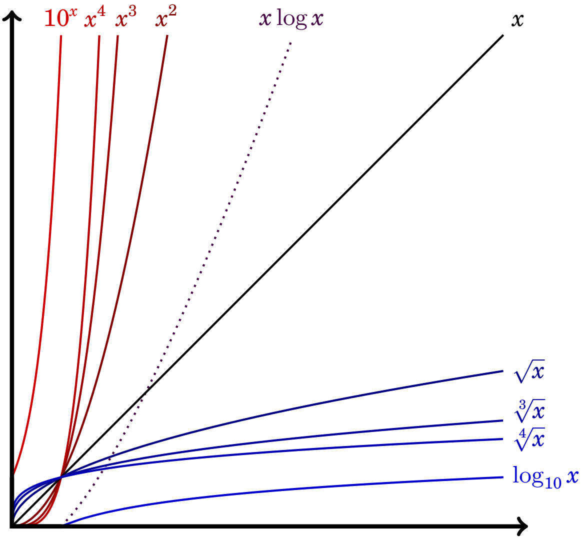
We focus on the **order of growth** of the time (or space) function.

This is also called the **asymptotic efficiency** of the algorithm.

Comparing Orders of Growth

- There are several standard "reference functions" that we use to classify orders of growth.
- It is important to be familiar with these functions and to be able to compare their growth rates.
- There are three main classes of common reference functions: exponentials, powers ("polynomial"), and logarithms.

Order of Growth



Runtime Table

n \ $f(n)$	$\log n$	n	$n \lg(n)$	n^2	2^n	$n!$
10	$0.003\mu s$	$0.01\mu s$	$0.033\mu s$	$0.1\mu s$	$1\mu s$	3.63 ms
20	$0.004\mu s$	$0.02\mu s$	$0.086\mu s$	$0.4\mu s$	1ms	77.1 y.
30	$0.005\mu s$	$0.03\mu s$	$0.147\mu s$	$0.9\mu s$	1 sec	8.4×10^{15} y.
40	$0.005\mu s$	$0.04\mu s$	$0.0213\mu s$	$1.6\mu s$	18.3 min	
50	$0.006\mu s$	$0.05\mu s$	$0.0282\mu s$	$2.5\mu s$	13 d.	
100	$0.007\mu s$	$0.1\mu s$	$0.644\mu s$	$10\mu s$	4×10^{13} y.	
10^3	$0.010\mu s$	$1\mu s$	$9.966\mu s$	1ms		
10^4	$0.013\mu s$	$10\mu s$	$130\mu s$	100ms		
10^5	$0.017\mu s$	$100\mu s$	1.67ms	10 sec		
10^6	$0.020\mu s$	1ms	19.93ms	16.7 min		
10^7	$0.023\mu s$	0.01 sec	0.23 sec	1.16 d.		
10^8	$0.027\mu s$	0.1 sec	2.66 sec	115.7 d.		
10^9	$0.030\mu s$	1 sec	29.9 sec	31.7 y.		

Quick Reminder: Logarithms and Exponents

If a, b, and x are all positive, then $\log_b x = \log_a x \cdot \log_b a$

Proof.

- Say $\log_b a = P$ and $\log_a x = Q$.
- Then we have $b^P = a$ and $a^Q = x$
- Hence: $b^{PQ} = (b^P)^Q = a^Q = x$
- That is, $b^{\log_b a \cdot \log_a x} = x$
- And so $\log_b a \cdot \log_a x = \log_b x$

$\square$

Quick Reminder: Logarithms and Exponents

In other words: all logs are equivalent up to a constant.

These computations are quite standard and you should be able to prove, for example, that:

$$a^{b(\log_a x)} = x^b$$

Comparing Functions

Definition ($f \leq g$)

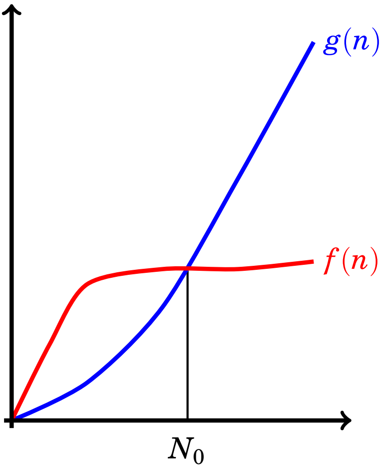
Let f and g be functions. Then $f \leq g$ iff $f(x) \leq g(x)$ for all x.

Definition ("big-Oh")

Let $f, g : \mathbb{R}^+ \to \mathbb{R}^+$. Then $f \in O(g)$ iff there are numbers $c > 0$ and $x_0 > 0$ such that $f(x) \leq c \cdot g(x)$ for all $x \geq x_0$.

To prove that $f \in O(g)$, you must come up with the two constants c and x_0 and show that the inequality above actually holds.

Illustration



$$\forall n \geq N_0,\ f(n) \leq g(n)$$

Asymptotic Notation

It is customary to write $f = O(g)$ instead of $f \in O(g)$.

This notation generalizes, but the big-Oh should only be on the right side of the equal sign.

Example

Suppose we have a complicated function f whose exact formula we don't know exactly. We can still write:

$$f(n) = n^3 + O(n^2)$$

That means that there is a function $h(n)$ such that:

$$f(n) = n^3 + h(n) \qquad \text{where } h(n) = O(n^2)$$

Note: That is a *more precise* statement than $f(n) = O(n^3)$. (Why?)

“big-Oh”: Example

Example

Let’s show that $2n^2 = O(n^3)$.

- We must find two actual numbers $c > 0$ and $n_0 > 0$ such that $2n^2 \leq cn^3$ for all $n \geq n_0$
- In this case, $c = 1$ and $n_0 = 2$ works, because when $2 \leq n$, then $2n^2 \leq n \cdot n^2 = n^3 = 1 \cdot n^3$.

This is what I expect your homework/exam answers to look like, when I ask you to prove $f = O(g)$ using the definition.

“big-Oh”: More Examples

Some examples (you have to be able to prove them):

- $n^2 = O(n^2 - 3)$
- $n^2 = O(n^2 + 3)$
- $100n^2 = O(n^2)$
- $n^2 = O(n^2 + 7n + 2)$
- $n^2 + 7n + 2 = O(n^2)$
- If $0 < p < q$, then $x^p = O(x^q)$
- For all $a > 0$ and $b > 0$, $\log_a x = O(\log_b x)$

Properties of “big-Oh” Notation

Lemma

If $f = O(h)$ *and* $g = O(h)$ *then* $f + g = O(h)$

Proof.

- $f = O(h)$ and therefore there are constants $c_1 > 0$ and $x_1 > 0$ such that $f(x) \leq c_1 h(x)$ for all $x \geq x_1$.
- $g = O(h)$ and therefore there are constants $c_2 > 0$ and $x_2 > 0$ such that $g(x) \leq c_2 h(x)$ for all $x \geq x_2$.
- Notice that these are not the same constants!
- We need to find constants that work for $f + g$.

Properties of “big-Oh” Notation

Proof (continued).

- We can use $c_1 + c_2$ and $\max(x_1, x_2)$.
- We must check that for all $x \geq \max(x_1, x_2)$, $f(x) + g(x) \leq (c_1 + c_2)h(x)$.
- This is because if $x \geq \max(x_1, x_2)$ then $x \geq x_1$, so $f(x) \leq c_1 h(x)$.
- Similarly, if $x \geq \max(x_1, x_2)$ then $x \geq x_2$, so $g(x) \leq c_2 h(x)$.
- Adding the inequalities, we see that when $x \geq \max(x_1, x_2)$ then $f(x) + g(x) \leq (c_1 + c_2)h(x)$

$\square$

Lower Bound: Ω Notation

Definition (Ω)

$f = \Omega(g)$ if there are constants $c > 0$ and $x_0 > 0$ such that $f(x) \geq c \cdot g(x)$ for all $x \geq x_0$.

Fact

$f = \Omega(g)$ *iff* $g = O(f)$.

Example

$\sqrt{n} = \Omega(\log(n))$

Tight Bound: Θ Notation

Definition (Θ)

$f = \Theta(g)$ if there are constants $a, b > 0$ and $x_0 > 0$ such that $ag(x) \leq f(x) \leq bg(x)$ for all $x \geq x_0$.

Example

It should be easy for you to show that: $\frac{1}{2}n^2 + 2n = \Theta(n^2)$.

Solving Recurrences

Recurrences often arise from analyzing divide and conquer algorithms or other recursive functions.

Example

Run time for Merge Sort:

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ 2T(\frac{n}{2}) + n & \text{otherwise} \end{cases}$$

We would like to get an explicit formula whenever possible.

We will explore multiple techniques for solving recurrences.

Solving Recurrences by Guess and Prove

One approach:

1. Guess a formula or bound of the solution.
2. Prove it by induction, generally for any necessary constant.

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

where T(1) is a constant.

Note that we should actually write $T(n) = 4T(\lfloor \frac{n}{2} \rfloor) + n$ unless n is a power of 2, but this is not a major point at the moment.

Guess and Prove

1. Guess $T(n) = O(n^3)$, and guess that $n_0 = 1$ will work.
2. Prove this by induction:

Proof.

- Base case: $T(1) \leq c(1^3)$. Trivial, provided that c is big enough.
- Inductive case: $T(n) \leq cn^3$.
- Inductive hypothesis: Assume that $T(k) \leq ck^3$ for $1 \leq k < n$.
- Now we calculate starting with $T(n)$:

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n && \text{by recurrence} \\ &\leq 4c\left(\frac{n}{2}\right)^3 + n && \text{by IH, since } n/2 < n \\ &= \frac{c}{2}n^3 + n = cn^3 - \left(\frac{c}{2}n^3 - n\right) \end{aligned}$$

and $cn^3 - (\frac{c}{2}n^3 - n) \leq cn^3$ is true whenever $\frac{c}{2}n^3 - n \geq 0$, and this is certainly true if for instance $c \geq 2$ and $n \geq 1$. (Can you prove this?)

□

Guess and Prove

Our initial guess may not be the tight bound. In this case, actually $T(n) = O(n^2)$. Again:

1. Guess that $T(n) = O(n^2)$, and that $n_0 = 1$ will work.
2. Prove by induction.

Proof.

- Base case: $T(1) \leq c \cdot 1^2$. Trivial, for a big enough c.
- Inductive case: $T(n) \leq c \cdot n^2$.
- Inductive hypothesis: Assume $T(k) \leq c \cdot k^2$ for all $1 \leq k < n$.
- Now we calculate starting with $T(n)$:

$$
\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + n && \text{by recurrence} \\
&\leq 4c\left(\frac{n}{2}\right)^2 + n && \text{by IH} \\
&= cn^2 + n
\end{aligned}
$$

!!! WRONG !!! We cannot show that $cn^2 + n \leq cn^2$. It's not true for $c > 0$, $n > 0$!

Guess and Prove

Problem: there's a lower-order term "in the way"
Repair: refine the guess to subtract the lower-order term:

$$T(n) \leq c_1 n^2 - c_2 n = O(n^2)$$

Proof.

- Base case: $T(1) \leq c_1 \cdot 1^2 - c_2 \cdot 1$.
- Inductive case: $T(n) \leq c_1 \cdot n^2 - c_2 \cdot n$.
- Inductive hypothesis: Assume $T(k) \leq c_1 \cdot k^2 - c_2 \cdot k$ for all $1 \leq k < n$.
- Now we calculate starting with $T(n)$:

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n && \text{by recurrence} \\ &\leq 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\frac{n}{2}\right) + n && \text{by IH} \\ &= c_1 n^2 - (2c_2 - 1)n \end{aligned}$$

So we must show $c_1 n^2 - (2c_2 - 1)n \leq c_1 n^2 - c_2 n$, which is true if $c_2 \geq 1$. □

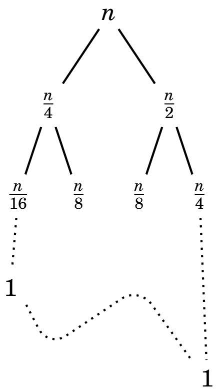
Solving Recurrences by Recursion Tree

Another approach (#2):

- Draw the **recursion tree** of problem sizes.
- Draw the corresponding tree of divide and combine costs.
- Sum the divide and combine costs per level.
- Calculate bounds on the *full* and *partial* tree levels.
- Run time = sum of divide and combine costs over all levels.

A more complicated recurrence: $T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + n^2$.

Problem Size



D&C Cost Tree

$$n^2$$

$$\left(\frac{n}{4}\right)^2 \quad \left(\frac{n}{2}\right)^2$$

$$\left(\frac{n}{16}\right)^2 \quad \left(\frac{n}{8}\right)^2 \quad \left(\frac{n}{8}\right)^2 \quad \left(\frac{n}{4}\right)^2$$

$$\vdots$$

D&C Cost

$$n^2 = \left(\frac{5}{16}\right)^0 n^2$$

$$\frac{5}{16}n^2 = \left(\frac{5}{16}\right)^1 n^2$$

$$\frac{25}{256}n^2 = \left(\frac{5}{16}\right)^2 n^2$$

$$\vdots$$

$$\left(\frac{5}{16}\right)^{\log_4 n} n^2$$

$$\vdots$$

$$\leq \left(\frac{5}{16}\right)^{\log_2 n} n^2$$

$T(n)$ is the *sum* of the *divide and combine cost* for each level.

Recursion Tree

Observations:

- The tree is fully filled up until the $log_4(n)$ level.
- The tree is partially filled up to the $log_2(n)$ level.

We can bound the runtime from above and below:

$$T(n) \leq n^2 \sum_{k=0}^{\log_2 n} \left(\frac{5}{16}\right)^k$$

$$T(n) \geq n^2 \sum_{k=0}^{\log_4 n} \left(\frac{5}{16}\right)^k$$

Recursion Tree

Observations:

- The tree is fully filled up until the $log_4(n)$ level.
- The tree is partially filled up to the $log_2(n)$ level.

We can bound the runtime from above and below:

$$T(n) \leq n^2 \sum_{k=0}^{\log_2 n} \left(\frac{5}{16}\right)^k \leq n^2 \sum_{k=0}^{\infty} \left(\frac{5}{16}\right)^k = n^2 \cdot \frac{1}{1 - \frac{5}{16}}$$

$$T(n) \geq n^2 \sum_{k=0}^{\log_4 n} \left(\frac{5}{16}\right)^k \geq n^2 \sum_{k=0}^{0} \left(\frac{5}{16}\right)^k = n^2 \cdot 1$$

That is, $c_1 n^2 \leq T(n) \leq c_2 n^2$, so $T(n) = \Theta(n^2)$.

Solving Recurrences with the Master Method

Another tool for solving recurrences (#3):

- Apply the **master theorem**.
- The master theorem applies only to recurrences of the form $T(n) = aT(\frac{n}{b}) + f(n)$ where $a \geq 1$, $b > 1$ and f is ultimately positive (that is, positive above some $x_0 > 0$).
  (So it doesn't apply to the previous example, for instance.)

Towards the Master Method

First, consider the recurrence $T(n) = aT(\frac{n}{b})$, where $a \geq 1, b > 1$.

A recurrence of this form arises from a divide and conquer algorithm that divides a problem into a sub-problems of size $\frac{n}{b}$.

Let's apply the guess and prove method:

- Let's assume that $T(n) = n^p$ for some p.
- Substituting n^p into the recurrence we get:
  $n^p = a\left(\frac{n}{b}\right)^p = \frac{a}{b^p}n^p$. So $b^p = a$.
- Taking $\log_b$ from both sides we get: $p = \log_b a$.
- Therefore, $T(n) = n^{\log_b a}$ is a solution to the recurrence.

The master theorem is based on this fact.

The Master Method

Unfortunately, divide and conquer recurrences are more complicated in general:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- The $aT(\frac{n}{b})$ term corresponds to conquering the sub-problems.
- The $f(n)$ part corresponds to the divide and combine costs.

The master theorem considers three cases ($p = \log_b a$):

1. $f(n)$ is small compared with n^p
2. $f(n)$ is comparable to n^p
3. $f(n)$ is large compared with n^p

The Master Method: f is Small

For this theorem (and not necessarily other cases), "$f(n)$ is small compared with n^p" means that there is an $\epsilon > 0$ such that

$$f(n) = O(n^{p-\epsilon}) = O(n^p/n^\epsilon)$$

That is, $f(n)$ grows more slowly than n^p by some positive power of n.

The Master Method: f is Large

Similarly, "$f(n)$ is large compared with n^p" means that there is an $\epsilon > 0$ such that

$$f(n) = \Omega(n^{p+\epsilon}) = \Omega(n^p n^\epsilon)$$

That is, $f(n)$ grows faster than n^p by some positive power of n.

Moreover, there has to be a constant $0 < c < 1$ and a constant n_0, so that for every $n > n_0$,

$$af\left(\frac{n}{b}\right) \leq cf(n)$$

where a and b are the same as in the recurrence formula.
(When does this hold for, say, $f(n) = n^k$?)

The Master Theorem

Theorem (Master Theorem)

If $a \geq 1$ *and* $b > 1$ *are constants,* $f(n)$ *is a function, and* $T(n)$ *is another function satisfying the recurrence* $T(n) = aT(n/b) + f(n)$ *where we interpret* n/b *to mean either* $\lfloor n/b \rfloor$ *or* $\lceil n/b \rceil$*, then* $T(n)$ *can be estimated asymptotically as follows:*

1. *If* $f(n) = O(n^{\log_b a - \epsilon})$ *for some constant* $\epsilon > 0$*,*
   then $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. *If* $f(n) = \Theta\left(n^{\log_b a}\right)$*,*
   then $T(n) = \Theta\left(n^{\log_b a} \log n\right)$.
3. *If* $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ *and if* $af(n/b) \leq cf(n)$ *for some constant* c *with* $0 < c < 1$ *and all sufficiently large* n*,*
   then $T(n) = \Theta(f(n))$.

The Cases of the Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad a \geq 1 \quad b > 1$$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
   When $f(n)$ is small compared with n^p, f essentially has no effect on the growth of T, and $T(n) = \Theta(n^p)$, just as it would if $f \equiv 0$.
   Compare with the example for the guess and prove technique.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
   This case is significant in that it applies to algorithms which are $O(n \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ *and* if $af(n/b) \leq cf(n)$ for some constant c with $0 < c < 1$ and all sufficiently large n, then $T(n) = \Theta\big(f(n)\big)$.
   In this case, the function f is what really contributes to the growth of T, and the recursion is immaterial.

The Master Theorem, Case 2

Case 2 is actually split in 2 in the text:

2a. If $f(n) = O(n^{\log_b a})$ then $T(n) = O(n^{\log_b a} \log n)$.

2b. If $f(n) = \Omega(n^{\log_b a})$ then $T(n) = \Omega(n^{\log_b a} \log n)$.

Putting the two together implies case 2, but case 2 doesn't immediately imply either of them.

Equivalently:

2a′. If $T(n) \leq aT(\frac{n}{b}) + f(n)$ where $f(n) = O(n^{\log_b a})$, then $T(n) = O(n^{\log_b a} \log n)$.

2b′. If $T(n) \geq aT(\frac{n}{b}) + f(n)$ where $f(n) = \Omega(n^{\log_b a})$, then $T(n) = \Omega(n^{\log_b a} \log n)$.

Example 1

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

Here we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n,\ n^p = n^2$.

Example 1

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

Here we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n,\ n^p = n^2$.

So this is case 1 where $f(n) = O(n^{2-\epsilon})$ for any $0 < \epsilon < 1$.

So $T(n) = \Theta(n^2)$.

Example 2

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

Here we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n^2,\ n^p = n^2$.

Example 2

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

Here we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n^2,\ n^p = n^2$.

So this is case 2 where $f(n) = \Theta(n^2)$.

So $T(n) = \Theta(n^2 \log(n))$.

Example 3

Example

$T(n) = 4T\left(\frac{n}{2}\right) + n^3.$

Now we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n^3,\ n^p = n^2.$

Example 3

Example

$T(n) = 4T\left(\frac{n}{2}\right) + n^3$.

Now we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n^3,\ n^p = n^2$.

We have $f(n) = \Omega(n^{\log_b a+\epsilon})$ for $0 < \epsilon < 1$. Thus we are in Case 3 provided we can show that the additional condition needed for Case 3 holds.

- We need to show that there is some constant $0 < c < 1$ and some n_0 such that for all $n > n_0$, $af\left(\frac{n}{b}\right) \leq cf(n)$.
- The condition $4f(n/2) \leq cf(n)$ becomes $4(n/2)^3 \leq cn^3$, or equivalently, $\frac{1}{2}n^3 \leq cn^3$.
- This holds for any $c \geq 1/2$.

Therefore we really are in Case 3, and the conclusion of the master theorem is that $T(n) = \Theta(n^3)$.

Example 4

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2/\log n$$

Here we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n^2/\log n,\ n^p = n^2$.

Example 4

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2/\log n$$

Here we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n^2/\log n,\ n^p = n^2$.

In this case the master theorem does not apply. (Why?)

Example 4

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2/\log n$$

Here we have: $a = 4,\ b = 2,\ p = \log_2 4 = 2,\ f(n) = n^2/\log n,\ n^p = n^2$.

In this case the master theorem does not apply. (Why?)

More precisely, the standard cases 1–3 don't apply. Case 2a applies, since $f(n) = n^2/\log n = O(n^2)$, so $T(n) = O(n^2 \log n)$.

Example 5

Example

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

Here we have: $a = 2,\ b = 2,\ p = \log_2 2 = 1,\ f(n) = cn,\ n^p = n$.

Example 5

Example

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

Here we have: $a = 2,\ b = 2,\ p = \log_2 2 = 1,\ f(n) = cn,\ n^p = n$.

So this is case 2 where $f(n) = \Theta(n)$.

So $T(n) = \Theta(n \log(n))$. This is the case of MergeSort, for example.

Generating Functions

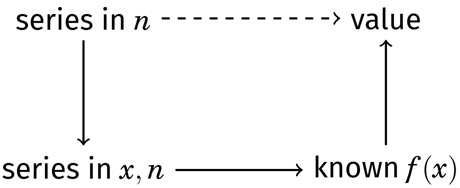
Puzzle: How can we compute the value of an infinite sum like the following?

$$\sum_{n=1}^{\infty} \frac{n}{2^n} = 2$$

Puzzle: How can we compute the value of an infinite sum like the following?

$$\sum_{n=1}^{\infty} \frac{n}{2^n} = 2$$

One approach:



Sequences and Generating Functions

Some important functions can be represented as power series:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \dots$$

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + x^4 + \dots \qquad \text{for } |x| < 1$$

Given a sequence $\{a_0, a_1, \ldots,\}$, the generating function of the sequence is defined as:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots = \sum_{n=0}^{\infty} a_n x^n$$

- The set of coefficients (like $a_n = \frac{1}{n!}$ in the case of $f(x) = e^x$) yield the power series for the function.
- If we *recognize* the power series and know what function it belongs to, we can use the function to gain knowledge about the sequence.

Generating Functions

We can use generating functions to derive the properties of sequences from properties of another sequence.

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n \qquad \text{for } |x| < 1$$

$$\frac{1}{(1-x)^2} = \sum_{n=0}^{\infty} n x^{n-1} = \sum_{n=1}^{\infty} n x^{n-1} \qquad \text{differentiate w.r.t } x$$

$$\frac{1}{\left(1-\frac{1}{2}\right)^2} = \sum_{n=1}^{\infty} n \left(\frac{1}{2}\right)^{n-1} \qquad \text{substitute } x = 1/2$$

$$2 = \sum_{n=1}^{\infty} \frac{n}{2^n} \qquad \text{simplify}$$

Another Example

The binomial theorem says that:

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

This just tells us that $(1+x)^n$ is the generating function for the finite sequence $\{\binom{n}{k} : 0 \leq k \leq n\}$.

Substituting $x = 1$ we get $2^n = \sum_{k=0}^{n} \binom{n}{k}$

Fibonacci Numbers via Generating Functions

- We let $\{f_0, f_1, f_2, \dots\}$ denote the Fibonacci numbers: $\{0, 1, 1, 2, 3, 5, 8, \dots\}$.
- For $n \geq 2$, $f_n = f_{n-1} + f_{n-2}$.
- We want to get a closed formula for f_n.
- We have a formula, but it is not obvious.
- We can use a generating function with the recurrence formula to derive it.

Generating Function for Fibonacci

$$F(x) = f_0 + f_1 x + f_2 x^2 + \cdots = \sum_{n=0}^{\infty} f_n x^n$$

$$\begin{aligned}
F(x) &= f_0 + f_1 x + f_2 x^2 + f_3 x^3 + f_4 x^4 + f_5 x^5 + \ldots \\
xF(x) &= \qquad f_0 x + f_1 x^2 + f_2 x^3 + f_3 x^4 + f_4 x^5 + \ldots \\
x^2 F(x) &= \qquad\qquad f_0 x^2 + f_1 x^3 + f_2 x^4 + f_3 x^5 + \ldots
\end{aligned}$$

Remember also that $f_{n+2} = f_{n+1} + f_n$, and $f_0 = 0,\ f_1 = 1$.

Generating Function for Fibonacci

$$F(x) = f_0 + f_1 x + f_2 x^2 + \cdots = \sum_{n=0}^{\infty} f_n x^n$$

$$\begin{aligned}
F(x) &= f_0 + f_1 x + f_2 x^2 + f_3 x^3 + f_4 x^4 + f_5 x^5 + \ldots \\
xF(x) &= \qquad f_0 x + f_1 x^2 + f_2 x^3 + f_3 x^4 + f_4 x^5 + \ldots \\
x^2 F(x) &= \qquad\qquad f_0 x^2 + f_1 x^3 + f_2 x^4 + f_3 x^5 + \ldots \\
\hline
(1 - x - x^2) F(x) &= f_0 + (f_1 - f_0) x
\end{aligned}$$

Remember also that $f_{n+2} = f_{n+1} + f_n$, and $f_0 = 0,\ f_1 = 1$.

Generating Function for Fibonacci

Adding the second and third row and subtracting from the first cancels most terms out, leaving:

$$F(x)(1 - x - x^2) = x$$

and so

$$F(x) = \frac{x}{(1 - x - x^2)}$$

We need to figure out a formula for the coefficient of the power series representing the right hand term.

We already know that for $|x| < 1$, $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$.

Generating Function for Fibonacci

- Our formula is not of this type, we have to convert it.
- It is a quadratic polynomial, so it can be converted into a formula of the kind:
- $(1 - x - x^2) = (1 - \alpha x)(1 - \beta x)$.
- Multiplying the right side we get: $\alpha\beta = -1$; $\alpha + \beta = 1$.
- $\alpha(1 - \alpha) = -1$; $\alpha^2 - \alpha - 1 = 0$.
- This is a quadratic equation whose solution is $\alpha = \frac{1 \pm \sqrt{5}}{2}$.

Generating Function for Fibonacci

- The two solutions add up to 1, so let's make: $\alpha = \frac{1+\sqrt{5}}{2}$ and $\beta = \frac{1-\sqrt{5}}{2}$
- We now know that: $F(x) = \frac{x}{1-x-x^2} = \frac{x}{(1-\alpha x)(1-\beta x)}$
- Now we can decompose it into two fractions without a quadratic term.
- For this we can find two numbers A and B such that: $\frac{x}{(1-\alpha x)(1-\beta x)} = \frac{A}{1-\alpha x} + \frac{B}{1-\beta x}$
- Which is true if: $A(1-\beta x) + B(1-\alpha x) = x$

Generating Function for Fibonacci

- This gives us two equations: $A + B = 0\,; A\beta + B\alpha = -1$.
- We know that $B = -A$ and we know that $\beta = 1 - \alpha$.
- Substituting, we get:

$$A(1-\alpha) - A\alpha = -1$$
$$A - A\alpha - A\alpha = -1$$
$$A(1-2\alpha) = -1$$

Generating Function for Fibonacci

- From previous calculation we know that: $1 - 2\alpha = -\sqrt{5}$.
- So we have: $A = \frac{1}{\sqrt{5}}$
- Knowing that $A + B = 0$ we get: $B = -A = -\frac{1}{\sqrt{5}}$
- Finally, putting it all together:

$$F(x) = \frac{A}{1 - \alpha x} + \frac{B}{1 - \beta x}$$
$$= A \sum_{n=0}^{\infty} \alpha^n x^n + B \sum_{n=0}^{\infty} \beta^n x^n$$
$$= \frac{1}{\sqrt{5}} \sum_{n=0}^{\infty} (\alpha^n - \beta^n) x^n$$

Generating Function for Fibonacci

Since the coefficients of F are the fibonacci numbers we get for the n^{th} coefficient:

$$f_n = \frac{1}{\sqrt{5}}(\alpha^n - \beta^n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$