CS624 - Analysis of Algorithms

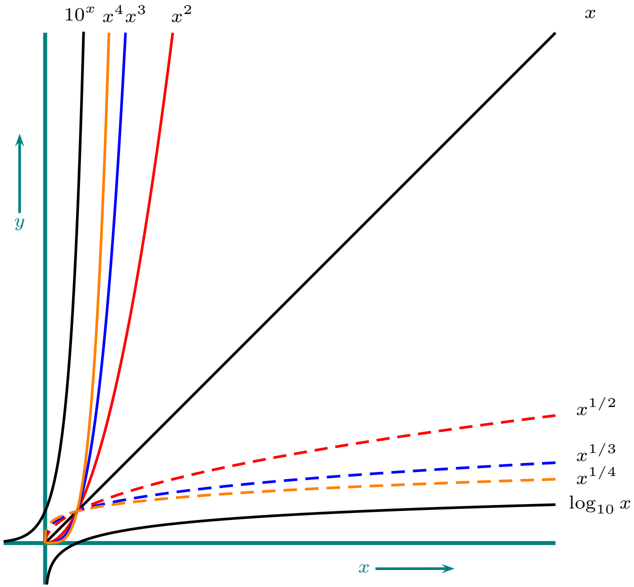
Runtime, Generating Functions

September 10, 2019

Order of Growth

- There are a lot of common mathematical functions that it is important to be familiar with.
- The first ones you need to really have a feeling for are powers, exponential functions, and logarithms.
- In particular, you really need to understand ”in your bones” how they grow for large values of their arguments, and how they compare to each other.

Order of Growth



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.

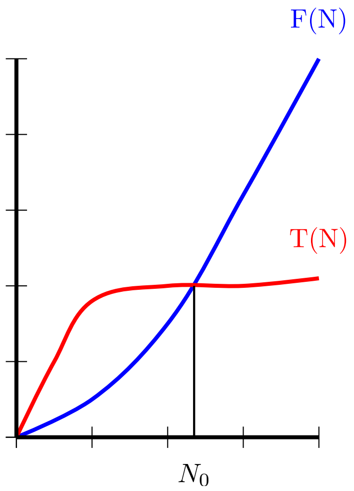
These computations are quite standard and you should be able to prove, for example, that:

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

Asymptotic Notation - big-Oh

- Define f and g as functions defined on positive numbers, taking positive numbers. $f \leq g$ iff $f(x) \leq g(x)$ for every x.
- Big-Oh is a slightly weaker notation: $f = O(g)$ if there are two numbers $c > 0$ and $x_0 > 0$ s.t. $f(x) \leq cg(x)$ for all $x \geq x_0$.
- To prove that $f = O(g)$ for some f and g, you must come up with two constants c and x_0 and show that the above is true.

illustration



Asymptotic Notation - big-Oh

- For instance, to prove that $2n^2 = O(n^3)$
- You have to 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, you should be able to see that $c = 1$ and $n_0 = 2$ works.
- Provided that $n \geq 2$, $2n^2 \leq n \cdot n^2 = n^3 = 1 \cdot n^3$.
- This is what I expect the answers to your homework/exams to look like.
- Notice that $f = O(g)$ doesn't mean mathematical equality.
- Notice also that the big-oh should only be on the right side of the equal sign.

Asymptotic notations – big-Oh

Example of usage:

- If we have a complicated function f whose exact formula we don't know we can still write:
- $f(n) = n^3 + O(n^2)$.
- This means that there is a function $h(n)$ such that: $f(n) = n^3 + h(n)$ where $h(n) = O(n^2)$.

Asymptotic Notation - big-Oh

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 the O-notation

Lemma

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

Proof.

- $f = O(h)$ and therefore there are constants $c > 0$ and $x_0 > 0$ s.t. $f(x) \leq ch(x)$ for all $x \geq x_0$.
- $g = O(h)$ and therefore there are constants $d > 0$ and $x_1 > 0$ s.t. $g(x) \leq dh(x)$ for all $x \geq x_1$.
- Notice that these are not the same constants!
- We need to put those together to come up with a formula for $f + g$.

□

Properties of the O-notation

Cont.

- We can use $c + d$ and $max(x_0, x_1)$.
- Therefore, for all $x \geq max(x_0, x_1)$, $f(x) + g(x) \leq (c + d)h(x)$.
- This is because if $x \geq max(x_0, x_1)$ then $x \geq x_0$, so
- $f(x) \leq ch(x)$
- Similarly, if $x \geq \max(x_0, x_1)$ then $x \geq x_1$, so
- $g(x) \leq dh(x)$
- Adding the above we see that for $x \geq max(x_0, x_1)$
- $f(x) + g(x) \leq (c + d)h(x)$

$\square$

Lower Bound – Ω Notation

$f = \Omega(g)$ if there are constants $c > 0$ and $x_0 > 0$ s.t. $f(x) >= c * g(x)$ for all $x \geq x_0$.

You should show pretty easily that $f = \Omega(g)$ iff $g = O(f)$.

For example: $\sqrt{n} = \Omega(\log(n))$

Tight Bound – Θ Notation

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

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

Solving Recursions

- Recurrences often arise from solving divide and conquer problems or other recursive functions.
- Example – the Merge Sort algorithm we previously saw.

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

- We would like to get an explicit formula whenever possible.

Solving Recurrences

Substitution – proof by induction:

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

Example: $T(n) = 4T(\frac{n}{2}) + n$
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.

Solve Recurrences by Induction

1. Guess $T(n) = O(n^3)$.
2. Prove this by induction:

Proof.

- Base case: $T(1) \leq c(1^3)$ provided that c is big enough.
- Assume that $n_0 = 1$ – we will prove that $T(k) \leq ck^3$ for all $k \geq 1$.
- Inductive hypothesis – assume that $T(k) \leq ck^3$ for $1 \leq k < n$.

Therefore we have

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

the last inequality being 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?) □

Solve Recurrences by Induction

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

1. Guess that $T(n) = O(n^2)$.
2. Prove by induction.

Proof.

- Base case: $T(1) \leq c * 1^2$ for a big enough c.
- We assume that $n_0 = 1$ so that we will show $T(k) \leq c * k^2$ for all $k \geq 1$.
- Inductive hypothesis: Assume this is true for all $1 \leq k < n$ and prove that it is true for n.

Trying again to use the recurrence formula:

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

!!! WRONG !!! □

Solve Recurrences by Induction

- The last step is wrong!
- $cn^2 + n$ is never smaller than or equal to cn^2 for positive n, c.
- Change the inductive hypothesis by subtracting the lower order term.
- Now we assume that $T(k) \leq c_1 k^2 - c_2 k$ for all $1 \leq k < n$ and for big enough c_1, c_2.

$$T(n) = 4T\left(\frac{n}{2}\right) + n \leq 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\frac{n}{2}\right) + n$$
$$= c_1 n^2 - 2c_2 n + n = c_1 n^2 - c_2 n - (c_2 - 1)n \leq c_1 n^2 - c_2 n$$

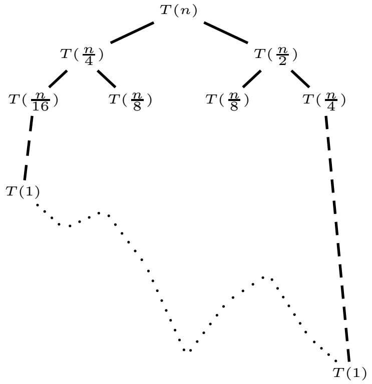
Which is true for all $c_2 \geq 1$.

Solve Recurrences by Induction

- Assume $c_2 = 1$, then T(1) needs to be bound by $c_1 1^2 - c_2 1$. We can assume that c_1 is big enough.
- In general – this is just another proof by induction.
- Remember to state the inductive hypothesis explicitly and show how the inductive step works.
- Expressing the hypothesis as a sequence of statements may be useful.

Recursion Tree

A more complicated formula: $T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + n^2$. We can build a recursion tree:



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

$$\left(\frac{n}{4}\right)^2 \quad \left(\frac{n}{2}\right)^2 \qquad \frac{5}{16}n^2 \qquad \left(\frac{5}{16}\right)^1 n^2$$

$$\left(\frac{n}{16}\right)^2\left(\frac{n}{8}\right)^2\left(\frac{n}{8}\right)^2\left(\frac{n}{4}\right)^2 \qquad \frac{25}{256}n^2 \qquad \left(\frac{5}{16}\right)^2 n^2$$

$$\vdots$$

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

$$\vdots$$

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

Recursion Tree

- The tree is filled up until the $log_4(n)$ level and partially filled up to the $log_2(n)$ level.
- We can bound the runtime from above and below:

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

and

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

Recursion Tree

- However, the two sums are just the beginning of a convergent geometric series, both bounded from above by a constant: $1 - \frac{1}{1-\frac{5}{16}}$
- They are also bounded below by 1 when n=1.
- So $n^2 \leq T(n) \leq cn^2 \Rightarrow T(n) = \Theta(n^2)$.

The Master Method

- It 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 (positive above some positive $x > x_0$ for some x_0).
- So it doesn't apply to the last recurrence we talked about.
- Let us first look at the recurrence $aT(\frac{n}{b})$.
- This recurrence usually appears in a divide and conquer problem where a and b are constants.
- The problem is divided into a sub-problems of size $\frac{n}{b}$
- Let's assume for simplicity that a is divisible by b.

The Master Method

- Let's assume also that $T(n) = n^p$ for some p.
- Substituting n^p into the recurrence we get: $n^p = a(\frac{n}{b})^p = a\frac{n^p}{b^p} \Rightarrow b^p = a$
- Taking $\log_b$ from both sides we get: $p = \log_b a$.
- Therefore – $T(n) = n^{\log_b a}$
- The master theorem is based on this fact.

The Master Method

- The original recurrence is slightly more complicated: $T(n) = aT(\frac{n}{b}) + f(n)$.
- In the divide and conquer algorithm we merge the a sub-problems of size $\frac{n}{b}$.
- The conquer part (the total cost of solving the sub-problems) is described by $aT(\frac{n}{b})$.
- Merging them into one complete solution is described by $f(n)$.

The Master Method

The master theorem considers three cases:

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.

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

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

This means that $f(n)$ grows more slowly than n^p by some positive power of n.
Remember that $p = \log_b a$

The Master Method

- Similarly, "$f(n)$ is large compared with n^p" means that there is an $\epsilon > 0$ s.t. $f(n) = \Omega(n^{p+\epsilon}) = \Omega(n^p n^\epsilon)$
- This means that $f(n)$ 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 $n > n_0$, $af(\frac{n}{b}) \leq cf(n)$.
- a and b are the same as in the recurrence formula.

The Master Theorem – Formulation

Theorem

If $a \geq 1$ and $b \geq 1$ 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 $T(n) = \Theta(n^{\log_b a})$.*
   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 $f \equiv 0$.
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 is actually split in 2 in the text:

1. If $f(n) = O(n^{\log_b a})$ then $T(n) = O(n^{\log_b a} \log n)$.
2. 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 any of them.
Equivalently:

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

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

Here we have: a=4, b=2, $f(n) = n$, $n^{\log_b a} = n^2$

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

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

Example 2

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

Here we have: a=4, b=2, $f(n) = n^2$, $n^{\log_b a} = n^2$

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

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

Example 3

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

Now we have: a=4, b=2, $f(n) = n^3$ so again $n^{\log_b a} = n^2$.
We have $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $0 < \epsilon < 1$. Thus we will be 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 s.t. for all $n > n_0$, $af(\frac{n}{b}) \leq cf(n)$
- $4f(n/2) \leq cf(n) \Rightarrow 4(n/2)^3 \leq cn^3$
- Or equivalently, $\frac{n^3}{2} \leq cn^3$
- This certainly holds for any $c > 1/2$. So we could take $c = 3/4$, for example.

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

Example 4

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

Here we have: a=4, b=2, $f(n) = n^2/\log n$, $n^{\log_b a} = n^2$

In this case the master theorem does not apply (any guesses why?).

Example 5

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

Here we have: a=2, b=2, $f(n) = cn$, $n^{\log_b a} = n$

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).

Sequences and Generating Functions

Some important functions can be represented as power series:

- $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \dots$
- $\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$
- $\cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$
- $\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + x^4 + \dots$ (makes sense for $|x| < 1$)

Generating Functions

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.
- This function can also give us a lot of information about the sequence.

Generating Functions

- We can use generating functions to derive the properties of sequences from properties of another sequence.
- For example: $\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$ (for $|x| < 1$)
- Differentiating both sides of the equation w.r.t x: $\frac{1}{(1-x)^2} = \sum_{n=0}^{\infty} nx^{n-1} = \sum_{n=1}^{\infty} nx^{n-1}$
- Substituting $x = 1/2$ we get $\sum_{n=1}^{\infty} \frac{n}{2^{n-1}} = 4$
- Or equivalently (multiplying both sides by $1/2$ to make it look simpler): $\sum_{n=1}^{\infty} \frac{n}{2^n} = 2$

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

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

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

Generating Function for Fibonacci

- Adding the second and third row and subtracting from the first cancels most terms out, leaving:
- $F(x)(1-x-x^2) = x$ so $F(x) = x/(1-x-x^2)$.
- 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:

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

Generating Function for Fibonacci

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

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