Sums of Powers Of Two

Introduction

My goal is to explore the following relation:

$$1 + 2 + 4 + 8 + ... + 2^n = 2^{n+1} - 1 \tag{1}$$

I'll start by providing the standard proof by induction. Then I'll discuss how I personally reason about it and find it plausible. This article will include tree visualizations, focus on the recursive nature of this sum, discuss how I use approximations to simplify my reasoning, and finally, I'll conclude with 2 bonus sections where I walk through another sum of terms recurrence relation and where I discuss both problems from a Computer Science and programming perspective.

Prerequisites

Knowledge of proof by induction. Which I believe people learn in high school when they take Precalculus or in their second Algebra course.

For the bonus CS section at the end, the main prerequisite is knowing binary arithmetic and generally having some introductory level experience.

Proof By Induction

I'll focus on the left-hand side (LHS) and right-hand side (RHS) of Equation 1 separately and express them as functions.

For the LHS, I'll define the function:

$$S(n) = 2^0 + 2^1 + ... + 2^n \tag{2}$$

This function has $n + 1$ terms. And here it is expressed as a sum:

$$S(n) = \sum_{i=0}^{n} 2^i \tag{3}$$

For the RHS, I'll define the function:

$$F(n) = 2^{n+1} - 1 \tag{4}$$

Let P_n be the claim that $S(n) = F(n)$. I want to prove P_n is true for all $n = 0, 1, 2, ...$ To this end, I'll use induction. The first step is to check the base case, P_0. $S(0) = 2^0 = 1$ and $F(0) = 2^{0+1} - 1 = 1$. $S(0) = F(0)$. Done.

Next, if I can show $P_k \Rightarrow P_{k+1}$, I'll be done. By induction, I would have proved Equation 1.

Here is the claim P_k:

$$2^0 + 2^1 + ... + 2^k = 2^{k+1} - 1 \tag{5}$$

And here is the claim P_{k+1}:

$$2^0 + 2^1 + ... + 2^k + 2^{k+1} = 2^{k+2} - 1 \tag{6}$$

Let me prove that $P_k \Rightarrow P_{k+1}$ in 3 ways:

1. the textbook way
2. essentially the textbook way, but with a focus on the recursive natures of $S(n)$ and $F(n)$.
3. a visual approach, using tree diagrams

Approach 1: Textbook Induction

Assuming Equation 5 is true, I want to show that Equation 6 is true as well. Observe that I can transform the LHS of Equation 6 by plugging in the RHS of Equation 5. After the substitution:

$$\left(2^{k+1} - 1\right) + 2^{k+1} = 2^{k+2} - 1 \tag{7}$$

Write both sides in terms of 2^k using exponent rules:

$$2^k * 2 - 1 + 2^k * 2 = 2^k * 4 - 1 \tag{8}$$

Factor 2^k in the LHS:

$$2^k * 4 - 1 = 2^k * 4 - 1 \tag{9}$$

And I am done. I have successfully proved that $P_k \Rightarrow P_{k+1}$. And since I already verified the base case, my proof of Equation 1 is complete.

Approach 2: Induction emphasizing recursive definitions

So the previous proof felt slightly unsatisfactory. What I really want to know, besides simply proving the correctness of Equation 1 is more insight as to *why* it's true. If someone looks at $S(n)$ with fresh eyes, defined in Equation 2, would they be able to come up with $F(n)$, defined in Equation 4, if they have never seen $F(n)$ before? Why is $F(n)$ plausible? Well, the prior proof, at least to me, did not seem to help me too much answer these questions. So the proofs in this subsection and the next attempt to answer my questions.

Ok, so actually I slightly lied. The last proof actually did help me, namely one key step in it. And that key step was the substitution of the RHS of Equation 5 into Equation 6. This exploited, and more importantly, displayed, the recursive structure of $S(n)$. That is, $S(k+1)$ expanded out contains $S(k)$.

I'll explicitly write this out:

$$S(k+1) = S(k) + 2^{k+1} \tag{10}$$

Now I'll do the same for $F(n)$ and try to write $F(k+1)$ in terms of $F(k)$.

$$F(k+1) = 2^{k+2} - 1 \tag{11}$$

$$= 2^{k+1} * 2 - 1 \tag{12}$$

$$= \left(2^{k+1} - 1\right) + 2^{k+1} \tag{13}$$

And done because note that I've spotted, and wrapped in brackets, $F(k)$. Again, I'll explicitly write this out:

$$F(k+1) = F(k) + 2^{k+1} \tag{14}$$

Note that $S(k+1)$ and $F(k+1)$ as defined at Equation 10 and Equation 14 share the exact same recursive structure! In fact, now the proof of the inductive step, $P_k \Rightarrow P_{k+1}$, writes itself.

$$S(k+1) \stackrel{?}{=} F(k+1) \tag{15}$$

Use the recursive definitions from Equation 10 and Equation 14:

$$S(k) + 2^{k+1} \stackrel{?}{=} F(k) + 2^{k+1} \tag{16}$$

Subtract 2^(k+1) from both sides:

$$S(k) \stackrel{?}{=} F(k) \tag{17}$$

And done, because we assume P_k to be true.

I feel that viewing $S(n)$ and $F(n)$ as recurrent relations, yet again, defined at Equation 10 and Equation 14 is really helpful. $S(n)$ grows exponentially at each step, every time we extend the sum by 1 term, we add double the last term. This is evident simply by looking at the expanded definition of $S(n)$ at Equation 2. $F(n)$ also clearly grows exponentially as it contains 2^n. So both these functions grow exponentially the same way at each step. And both share the same base case, $S(0) = F(0) = 1$. So these functions grow in lockstep with each other and will always remain equivalent. Now I feel I have more insight in terms of considering growth. I feel there are definitely parallels to calculus that, while at the moment I'm unequipped to treat, may be worth exploring.

Approach 3: "Approximate" Visual Induction
Draw the tree and table. Possibly to do so side-by-side? As figuring out the pattern behind sums of powers of 2, it's likely an observer would simply notice the pattern by looking at this table. Maybe expanding out a few more levels to convince themselves of the increasingly promising pattern they've formulated that is $F(n)$.

This section, I'd like to introduce a way I reasoned about $F(n)$ being plausible. Again, from the previous section, the key idea is the exponential growth of $S(n)$. Visually at each level, 2^d more nodes are introduced. So my candidate function to match or approximate $S(n)$ could grow exponentially. And the base is 2. So why not simply try the function 2^{n+1}? Indeed $S(n) \cong 2^{n+1}$. And I'll present a visual "proof" of this. 2^n is very convenient with this visual tree approach because it corresponds to the number of leaves at a given level.

Approach 4: Binary, rectifying the approximation
Preface. Knowing binary will help. But it is not a hard prerequisite

consider (unsigned) binary representation of 7 (0b0111) and 8 (0b1000) using 4 bits For readers that are not acquinted with binary, this is not something to be scared about. this simply means 7 = 1*2^0 + 1*2^1 + 1*2^2 + 0*2^3 = 1 + 2 + 4 and 8 = 0*2^0 + 0*2^1 + 0*2^2 + 1*2^3 = 8

Quick introduction to binary: The 1′s and 0′s record presense or absence of a particular power of two, and the powers of 2 increase from rightmost to leftmost, just like our typical decimal notation where one's place is rightmost, followed by ten's, hundred's, and so forth. 0bABCD = D*2^0 + C*2^1 + B*2^2 + A*2^3 where A,B,C, and D are all binary digits meaning they take on values 0 or 1.

So the sum 1+2+4 or S(2) = 7. So just 1 off from 8, the next power of 2. If we add 1 to 0b0111, there's a domino effect of carrying over 1s and we get 0b1000 Even if you don't know binary consider what happens when I evaluate 1 + (1 + 2 + 4) as follows first let's rewrite all terms as powers of 2 as that's the heart of this document 2^0 + (2^0 + 2^1 + 2^2) group first 2 terms (2^0+2^0) + (2^1 + 2^2) simplify the grouping 2^1 + (2^1 + 2^2) group first 2 terms (2^1+2^1) + (2^2) simplify the grouping 2^2 + (2^2) group first 2 terms (2^2 + 2^2) simplify the grouping 2^3 Note the recursive nature of this process, the domino effect! Given 2^k + (2^k + 2^(k+1) + …) We perform the 2 steps of grouping first 2 terms and simplyfing 2^(k+1) + (2^(k+1) + 2^(k+2) + …) and we get expression of same structure except one higher power of 2

So there's repeated doubling To recap, last visual approach, we saw repeated halving. Each level S(k), we represented as 2^k + S(k-1) S(2) = 4 + S(1), S(1) = 2 + S(0) S(k) = 2^k + S(k-1) S(k-1) = 2^(k-1) + S(k-2) S(k-2) = 2^(k-2) + S(k-3) To solve problem K, we need problem K-1. To solve problem K-1, we need problem K-2 And so forth till 0th problem. From large problem we work backwards from small problem

But this way is more direct, from small subproblem, we work towards larger subproblems directly 1 + 1 = 2 = S(0) + 1 2 + 2 = 4 = S(1) + 1 4 + 4 = 8 = S(2) + 1 We're solving 1+problem K-1 along the route to solving 1+problem K In fact when solving 1 + problem K, we solve all intermediate 1 + problem K - J as those are the various powers of 2 being carried.

Bother bringing up domino effect in decimal (ex 1000 = 999+1) or arbitrary base like idea is get 111 in some base, b. Then scale by b-1 so that adding 1 causes the domino cascade?

CS aside: we see this idea all the time in bit manipulation that take advantage of 2s complement. Example that comes to mind is BIT or Fenwick tree technique to get least significant 1 bit.

domino affect from prev approach to 8 we subtracted 4 then we subtracted 2 then subtracted 1 and left with 0 =1 So binary has a domino affect