Notes on summations and related topics

James Aspnes

December 13, 2010

1 Summations

Summations are the discrete versions of integrals; given a sequence $x_a, x_{a+1}, \dots, x_b$, its sum $x_a + x_{a+1} + \dots + x_b$ is written as $\sum_{i=a}^{b} x_i$.

The large jagged symbol is a stretched-out version of a capital Greek letter sigma. The variable i is called the **index of summation**, a is the **lower bound** or **lower limit**, and b is the **upper bound** or **upper limit**. Mathematicians invented this notation centuries ago because they didn't have *for* loops; the intent is that you loop through all values of i from a to b (including both endpoints), summing up the body of the summation for each i.

If $b < a$, then the sum is zero. For example,

$$\sum_{i=0}^{-5} \frac{2^i \sin i}{i^3} = 0.$$

This rule mostly shows up as an extreme case of a more general formula, e.g.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2},$$

which still works even when $n = 0$ or $n = -1$ (but not for $n = -2$).

Summation notation is used both for laziness (it's more compact to write $\sum_{i=0}^{n}(2i+1)$ than $1 + 3 + 5 + 7 + \dots + (2n+1)$) and precision (it's also more clear exactly what you mean).

1.1 Formal definition

For finite sums, we can formally define the value by either of two recurrences:

$$\sum_{i=a}^{b} f(i) = \begin{cases} 0 & \text{if } b < a \\ f(a) + \sum_{i=a+1}^{b} f(i) & \text{otherwise.} \end{cases} \tag{1}$$

$$\sum_{i=a}^{b} f(i) = \begin{cases} 0 & \text{if } b < a \\ f(b) + \sum_{i=a}^{b-1} f(i) & \text{otherwise.} \end{cases} \tag{2}$$

In English, we can compute a sum recursively by computing either the sum of the last $n-1$ values or the first $n-1$ values, and then adding in the value we left out. (For infinite sums we need a different definition; see below.)

1.2 Choosing and replacing index variables

When writing a summation, you can generally pick any index variable you like, although i, j, k, etc., are popular choices. Usually it's a good idea to pick an index that isn't used outside the sum. Though

$$\sum_{n=0}^{n} n = \sum_{i=0}^{n} i$$

has a well-defined meaning, the version on the right-hand side is a lot less confusing.

In addition to renaming indices, you can also shift them, provided you shift the bounds to match. For example, rewriting

$$\sum_{i=1}^{n} (i-1)$$

as

$$\sum_{j=0}^{n-1} j$$

(by substituting j for $i-1$) makes the sum more convenient to work with.

1.3 Scope

The *scope* of a summation extends to the first addition or subtraction symbol that is not enclosed in parentheses or part of some larger term (e.g., in the numerator of a fraction). So

$$\sum_{i=1}^{n} i^2 + 1 = \left(\sum_{i=1}^{n} i^2\right) + 1 = 1 + \sum_{i=1}^{n} i^2 \neq \sum_{i=1}^{n} (i^2+1).$$

Since this can be confusing, it is generally safest to wrap the sum in parentheses (as in the second form) or move any trailing terms to the beginning. An exception is when adding together two sums, as in

$$\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n^2} i = \left(\sum_{i=1}^{n} i^2\right) + \left(\sum_{i=1}^{n^2} i\right).$$

Here the looming bulk of the second sigma warns the reader that the first sum is ending; it is much harder to miss than the relatively tiny plus symbol in the first example.

1.4 Sums over given index sets

Sometimes we'd like to sum an expression over values that aren't consecutive integers, or may not even be integers at all. This can be done using a sum over all indices that are members of a given index set, or in the most general form satisfy some given predicate (with the usual set-theoretic caveat that the objects that satisfy the predicate must form a set). Such a sum is written by replacing the lower and upper limits with a single subscript that gives the predicate that the indices must obey.

For example, we could sum i^2 for i in the set $\{3, 5, 7\}$:

$$\sum_{i \in \{3,5,7\}} i^2 = 3^2 + 5^2 + 7^2 = 83.$$

Or we could sum the sizes of all subsets of a given set S:

$$\sum_{A \subseteq S} |A|.$$

Or we could sum the inverses of all prime numbers less than 1000:

$$\sum_{p\,<\,1000,\ p \text{ is prime}} 1/p.$$

Sometimes when writing a sum in this form it can be confusing exactly which variable or variables are the indices. The usual convention is that a variable is always an index if it doesn't have any meaning outside the sum, and if possible the index variable is put first in the expression under the sigma if possible. If it is not obvious what a complicated sum means, it is generally best to try to rewrite it to make it more clear; still, you may see sums that look like

$$\sum_{1 \le i < j \le n} \frac{i}{j}$$

or

$$\sum_{x \in A \subseteq S} |A|$$

where the first sum sums over all *pairs* of values (i, j) that satisfy the predicate, with each pair appearing exactly once, and the second sums over all sets A that are subsets of S and contain x (assuming x and S are defined outside the summation). Hopefully, you will not run into too many sums that look like this, but it's worth being able to decode them if you do.

Sums over a given set are guaranteed to be well-defined only if the set is finite. In this case we can use the fact that there is a bijection between any finite set S and the ordinal $|S|$ to rewrite the sum as a sum over indices in $|S|$. For example, if $|S| = n$, then there exists a bijection $f : \{0 \dots n-1\} \leftrightarrow S$, so we can define

$$\sum_{i \in S} x_i = \sum_{i=0}^{n-1} x_{f(i)}.$$

If S is infinite, this is trickier. For countable S, where there is a bijection $f : \mathbb{N} \leftrightarrow S$, we can sometimes rewrite

$$\sum_{i \in S} x_i = \sum_{i=0}^{\infty} x_{f(i)}.$$

and use the definition of an infinite sum (given below). Note that if the x_i have different signs the result we get may depend on which bijection we choose. For this reason such infinite sums are probably best avoided unless you can explicitly use $\mathbb{N}$ as the index set.

1.5 Sums without explicit bounds

When the index set is understood from context, it is often dropped, leaving only the index, as in $\sum_i i^2$. This will generally happen only if the index spans all possible values in some obvious range, and can be a mark of sloppiness in formal mathematical writing. Theoretical physicists adopt a still more lazy approach, and leave out the $\sum_i$ part entirely in certain special types of sums: this is known as the Einstein summation convention after the notoriously lazy physicist who proposed it.

1.6 Infinite sums

Sometimes you may see an expression where the upper limit is infinite, as in

$$\sum_{i=0}^{\infty} \frac{1}{i^2}.$$

The meaning of this expression is the **limit** of the series s obtained by taking the sum of the first term, the sum of the first two terms, the sum of the first three terms, etc. The limit **converges** to a particular value x if for any $\epsilon > 0$, there exists an N such that for all $n > N$, the value of s_n is within ϵ of x (formally, $|s_n - x| < \epsilon$). We will see some examples of infinite sums when we look at generating functions.

1.7 Double sums

Nothing says that the expression inside a summation can't be another summation. This gives double sums, such as in this rather painful definition of multiplication for non-negative integers:

$$a \times b \stackrel{\text{def}}{=} \sum_{i=1}^{a} \sum_{j=1}^{b} 1.$$

If you think of a sum as a *for* loop, a double sum is two nested *for* loops. The effect is to sum the innermost expression over all pairs of values of the two indices.

Here's a more complicated double sum where the limits on the inner sum depend on the index of the outer sum:

$$\sum_{i=0}^{n}\sum_{j=0}^{i}(i+1)(j+1).$$

When $n = 1$, this will compute $(0+1)(0+1)+(1+1)(0+1)+(1+1)(1+1) = 7$. For larger n the number of terms grows quickly.

There are also triple sums, quadruple sums, etc.

2 Computing sums

When confronted with some nasty sum, it is nice to be able to convert into a simpler expression that doesn't contain any summation signs. It is not always possible to do this, and the problem of finding a simpler expression for a sum is very similar to the problem of computing an integral (see HowToIntegrate): in both cases the techniques available are mostly limited to massaging the summation until it turns into something whose simpler expression you remember. To do this, it helps to both (a) have a big toolbox of sums with known values, and (b) have some rules for manipulating summations to get them into a more convenient form. We'll start with the toolbox.

2.1 Some standard sums

Here are the three formulas you should either memorize or remember how to derive:

$$\begin{aligned}\sum_{i=1}^{n} 1 &= n\\ \sum_{i=1}^{n} i &= \frac{n(n+1)}{2}\\ \sum_{i=0}^{n} r^i &= \frac{1-r^{n+1}}{1-r}\end{aligned}$$

Rigorous proofs of these can be obtained by induction on n.

For not so rigorous proofs, the second identity can be shown (using a trick alleged to have been invented by the legendary 18th-century mathematician Carl Friedrich Gauss at a frighteningly early age by adding up two copies of the sequence running in opposite directions, one term at a time:

```
 S =     1    +    2     +     3    +   ....   +   n
 S =     n    +   n-1    +    n-2   +   ....   +   1
------------------------------------------------------
2S =   (n+1)  +  (n+1)   +   (n+1)  +   ....   + (n+1) = n(n+1),
```

and from $2S = n(n+1)$ we get $S = n(n+1)/2$.

For the last identity, start with

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r},$$

which holds when $|r| < 1$. The proof is that if

$$S = \sum_{i=0}^{\infty} r^i$$

then

$$rS = \sum_{i=0}^{\infty} r^{i+1} = \sum_{i=1}^{\infty} r^i$$

and so

$$S - rS = r^0 = 1.$$

Solving for S then gives $S = 1/(1-r)$.

We can now get the sum up to n by subtracting off the extra terms starting with $rn+1$:

$$\sum_{i=0}^{n} r^i = \sum_{i=0}^{\infty} r^i - r^{n+1} \sum_{i=0}^{\infty} r^i = \frac{1}{1-r} - \frac{r^{n+1}}{1-r} = \frac{1-r^{n+1}}{1-r}.$$

Amazingly enough, this formula works even when r is greater than 1. If r is equal to 1, then the formula doesn't work (it requires dividing zero by zero), but there is an easier way to get the solution.

Other useful sums can be found in various places. Rosen and Concrete Mathematics both provide tables of sums in their chapters on generating functions. But it is usually better to be able to reconstruct the solution of a sum rather than trying to memorize such tables.

2.2 Summation identities

The summation operator is *linear*. This means that constant factors can be pulled out of sums:

$$\sum_{i \in S} a x_i = a \sum_{i \in S} x_i$$

and sums inside sums can be split:

$$\sum_{i \in S} (x_i + y_i) = \sum_{i \in S} x_i + \sum_{i \in S y_i}.$$

With multiple sums, the order of summation is not important, provided the bounds on the inner sum don't depend on the index of the outer sum:

$$\sum_{i \in S} \sum_{j \in T} x_{ij} = \sum_{j \in T} \sum_{i \in S} x_{ij}.$$

Products of sums can be turned into double sums of products and vice versa:

$$\left(\sum_{i\in S} x_i\right)\left(\sum_{j\in T} y_j\right) = \sum_{i\in S}\sum_{j\in T} x_i y_j.$$

These identities can often be used to transform a sum you can't solve into something simpler.

2.3 What to do if nothing else works

If nothing else works, you can try using the "guess but verify" method, which is a variant on the same method for identifying sequences. Here we write out the values of the sum for the first few values of the upper limit (for example), and hope that we recognize the sequence. If we do, we can then try to prove that a formula for the sequence of sums is correct by induction.

Example: Suppose we want to compute

$$S(n) = \sum_{k=1}^{n}(2k-1)$$

but that it doesn't occur to us to split it up and use the $\sum_{k=1}^{n} k$ and $\sum_{k=1}^{n} 1$ formulas. Instead, we can write down a table of values:

n	S(n)
0	0
1	1
2	$1+3=4$
3	$1+3+5=9$
4	$1+3+5+7=16$
5	$1+3+5+7+9=25$

At this point we might guess that $S(n) = n^2$. To verify this, observe that it holds for $n = 0$, and for larger n we have $S(n) = S(n-1) + (2n-1) = (n-1)^2 + 2n - 1 = n^2 - 2n + 1 - 2n - 1 = n^2$. So we can conclude that our guess was correct.

If this doesn't work, you could always try using generating functions.

2.4 Strategies for asymptotic estimates

Mostly in algorithm analysis we do not need to compute sums exactly, because we are just going to wrap the result up in some asymptotic expression anyway (see asymptotic notation). This makes our life much easier, because we only need an approximate solution.

Here's my general strategy for computing sums:

2.4.1 Pull out constant factors

Pull as many constant factors out as you can (where constant in this case means anything that does not involve the summation index). Example: $\sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} = nH_n = \Theta(n \log n)$. (See harmonic series below.)

2.4.2 Bound using a known sum

See if it's bounded above or below by some other sum whose solution you already know. Good sums to try (you should memorize all of these):

Geometric series $\sum_{i=0}^{n} x^i = \frac{1-x^{n+1}}{1-x}$ and $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$.

The way to recognize a geometric series is that the ratio between adjacent terms is constant. If you memorize the second formula, you can rederive the first one. If you're Gauss, you can skip memorizing the second formula.

A useful trick to remember for geometric series is that if x is a constant that is not exactly 1, the sum is always big-Theta of its largest term. So for example $\sum_{i=1}^{n} 2^i = \Theta(2^n)$ (the exact value is $2^{n+1} - 2$), and $\sum_{i=1}^{n} 2^{-i} = \Theta(1)$ (the exact value is $1 - 2^{-n}$). This fact is the basis of the *Master Theorem*, described in SolvingRecurrences. If the ratio between terms equals 1, the formula doesn't work; instead, we have a constant series (see below).

Constant series $\sum_{i=1}^{n} 1 = n$.

Arithmetic series The simplest arithmetic series is $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. The way to remember this formula is that it's just n times the average value $(n+1)/2$. The way to recognize an arithmetic series is that the difference between adjacent terms is constant. The general arithmetic series is of the form $\sum_{i=1}^{n}(ai + b) = \sum_{i=1}^{n} ai + \sum_{i=1}^{n} b = an(n+1)/2 + bn$. Because the general series expands so easily to the simplest series, it's usually not worth memorizing the general formula.

Harmonic series $\sum_{i=1}^{n} 1/i = H_n = \Theta(n \log n)$.

Can be rederived using the integral technique given below or by summing the last half of the series, so this is mostly useful to remember in case you run across H,,n,, (the "n-th harmonic number").

2.4.3 Bound part of the sum

See if there's some part of the sum that you can bound. For example, $\sum_{i=1}^{n} i^3$ has a (painful) exact solution, or can be approximated by the integral trick described below, but it can very quickly be solved to within a constant factor by observing that $\sum_{i=1}^{n} i^3 \leq \sum_{i=1}^{n} n^3 = O(n^4)$ and $\sum_{i=1}^{n} i^3 \geq \sum_{i=n/2}^{n} i^3 \geq \sum_{i=n/2}^{n} (n/2)^3 = \Omega(n^4)$.

2.4.4 Integrate

Integrate. If $f(n)$ is non-decreasing and you know how to integrate it, then $\int_{a-1}^{b} f(x)dx \leq \sum_{i=a}^{b} f(i) \leq \int_{a}^{b+1} f(x)dx$, which is enough to get a big-Theta bound for almost all functions you are likely to encounter in algorithm analysis. If you don't remember how to integrate, see HowToIntegrate.

2.4.5 Grouping terms

Try grouping terms together. For example, the standard trick for showing that the harmonic series is unbounded in the limit is to argue that $1+1/2+1/3+1/4+1/5+1/6+1/7+1/8+\cdots \geq 1+1/2+(1/4+1/4)+(1/8+1/8+1/8+1/8)+\cdots \geq 1+1/2+1/2+1/2+\ldots$. I usually try everything else first, but sometimes this works if you get stuck.

2.4.6 Oddities

One oddball sum that shows up occasionally but is hard to solve using any of the above techniques is $\sum_{i=1}^{n} a^i i$. If $a < 1$, this is $\Theta(1)$ (the exact formula for $\sum_{i=1}^{\infty} a^i i$ when $a < 1$ is $a/(1-a)^2$, which gives a constant upper bound for the sum stopping at n); if $a = 1$, it's just an arithmetic series; if $a > 1$, the largest term dominates and the sum is $\Theta(a^n n)$ (there is an exact formula, but it's ugly—if you just want to show it's $O(a^n n)$, the simplest approach is to bound the series $\sum_{i=0}^{n-1} a^{n-i}(n-i)$ by the geometric series $\sum_{i=0}^{n-1} a^{n-i} n \leq a^n n/(1-a^{-1}) = O(a^n n)$. I wouldn't bother memorizing this one provided you remember how to find it in these notes.

2.4.7 Final notes

In practice, almost every sum you are likely to encounter in algorithm analysis will be of the form $\sum_{i=1}^{n} f(n)$ where $f(n)$ is exponential (so that it's bounded by a geometric series and the largest term dominates) or polynomial (so that $f(n/2) = \Theta(f(n))$) and the sum is $\Theta(nf(n))$ using the $\sum_{i=n/2}^{n} f(n) = \Omega(nf(n))$ lower bound).

Concrete Mathematics spends a lot of time on computing sums exactly. The most useful technique for doing this is to use generating functions.

3 Products

What if you want to multiple a series of values instead of add them? The notation is the same as for a sum, except that you replace the sigma with a pi, as in this definition of the factorial function for non-negative n:

$$n! \overset{\text{def}}{=} \prod_{i=1}^{n} i = 1 \cdot 2 \cdot \cdots \cdot n.$$

The other difference is that while an empty sum is defined to have the value 0, an empty product is defined to have the value 1. The reason for this rule (in both cases) is that an empty sum or product should return the **identity element** for the corresponding operation—the value that when added to or multiplied by some other value x doesn't change x. This allows writing general rules like:

$$\begin{aligned} \sum_{i\in A} f(i) + \sum_{i \in B} f(i) &= \sum_{i \in A\cup B} f(i) \\ \left(\prod_{i\in A} f(i)\right) \cdot \left(\prod_{i\in B} f(i)\right) &= \prod_{i\in A\cup B} f(i) \end{aligned}$$

which holds as long as $A \cap B = \emptyset$. Without the rule that the sum of an empty set was 0 and the product 1, we'd have to put in a special case for when one or both of A and B were empty.

Note that a consequence of this definition is that $0! = 1$.

4 Other big operators

Some more obscure operators also allow you to compute some aggregate over a series, with the same rules for indices, lower and upper limits, etc., as $\sum$ and $\prod$. These include:

- Big AND:
$$\bigwedge_{x\in S} P(x) \equiv P(x_1) \wedge P(x_2) \wedge \ldots \equiv \forall x \in S : P(x).$$
- Big OR:
$$\bigvee_{x\in S} P(x) \equiv P(x_1) \vee P(x_2) \vee \ldots \equiv \exists x \in S : P(x).$$
- Big Intersection:
$$\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap \ldots \cap A_n.$$
- Big Union:
$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \ldots \cup A_n.$$

These all behave pretty much the way one would expect. One issue that is not obvious from the definition is what happens with an empty index set. Here the rule as with sums and products is to return the identity element for the operation. This will be True for AND, False for OR, and the empty set for union; for intersection, there is no identity element in general, so the intersection over an empty collection of sets is undefined.