Extended capability for a more restrictive generic

I have a generic binary search tree based on Comparable:

public class BSTree<T: Comparable> {

...

public func insert(\_ val: T, \_ n: Int) {

...

}

@discardableResult

public func delete(\_ val: T, \_ n: Int) -> Int {

...

}

...

}

I want to add the ability to provide a sum of the values, if T is an arithmetic type. I tried the following:

public class BSTree<T: Comparable> {

private var sumStorage: T?

...

public func insert(\_ val: T, \_ n: Int) {

if let arithVal = val as? AdditiveArithmetic {

for \_ in 0 ..< n { sumStorage += arithVal }

}

...

}

@discardableResult

public func delete(\_ val: T, \_ n: Int) -> Int {

...

numDeleted = ...

if let arithVal = val as? AdditiveArithmetic {

for \_ in 0 ..< numDeleted { sumStorage -= arithVal }

}

}

...

}

extension BSTree where T: AdditiveArithmetic {

public var sum: T {

sumStorage as? T ?? T.zero

}

}

Of course, when I try to cast val as AdditiveArithmetic I get “Protocol 'AdditiveArithmetic' can only be used as a generic constraint because it has Self or associated type requirements”. Plus sumStorage isn’t AdditiveArithmetic, so I can’t add to it, and I can’t make it a stored property of the extension, because ... you just can’t.

What I finally came up with was to use inheritance:

class SummedBSTree<T>: BSTree<T> where T: AdditiveArithmetic & Comparable {

public var sum = T.zero

override public func insert(\_ val: T, \_ n: Int) {

super.insert(val, n)

for \_ in 0 ..< n { sum += val }

}

@discardableResult

override public func delete(\_ val: T, \_ n: Int) -> Int {

let numDeleted = super.delete(val, n)

for \_ in 0 ..< numDeleted { sum -= val }

return numDeleted

}

}

This works, but it seems like it’s a case of using a sledgehammer where a jeweler’s screwdriver should be able to do the trick. It’s frustrating that something that would be so easy to do in Objective-C (and other less strongly typed languages) is so difficult in Swift. Can someone come up with a way of adding the summing capability without subclassing?