Computability

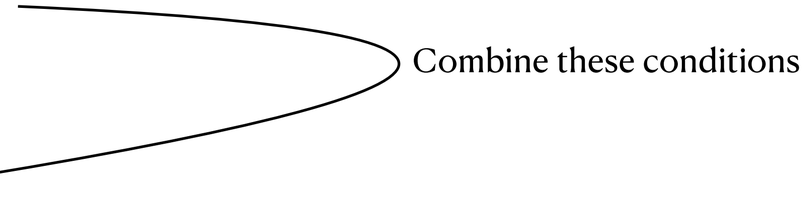
Computability

Question: Will a program crash?

- Can anyone write a program that takes some other program as input and simply answer the following yes/no question
  - Will the input program ever crash?
  - This is a variant on the Halting Problem (Turing)
- This is largely taken from MacCormack(2012) ch 10

Proof by Contradiction

- Suppose there exists a program "mayCrash" that will accepts some inputs then and after processing the inputs it does one of three things:
  - output "YES"
  - output "NO"
  - crash



- Suppose there exists a program "canCrash" that takes as input a program (like mayCrash) and a set of inputs for the input program and outputs
  - YES if the program could crash
  - NO otherwise

CanCrashMod

- CanCrashMod is identical to canCrash BUT
  - outputs
    - rather that saying yes, it crashes
    - NO otherwise

SelfCanCrashMod

- Modify CanCrashMod to SelfCanCrashMod
  - crashes when given itself and inputs that would cause CanCrashMod to crash
  - No otherwise
    - Side note: Even this is pretty much impossible. You need a program that is capable of running itself in simulation. Which means that you need the program to have as a part of itself a simulator that can run itself. ....
      - Can you write a compiler that compiles itself?

AntiSelfCanCrashMod

- The negative of SelfCanCrashMod
  - if input would cause a crash when run on itself, return YES
  - crash

CanCrash

Outputs:
- YES
- No

AntiSelfCanCrashMod

Outputs:
- Yes
- CRASH

CanCrashMod

Outputs:
- CRASH
- No

SelfCanCrashMod

Outputs:
- CRASH
- No

Contradiction

Danger, Will Robinson

https://www.google.com/search?client=firefox-b-1-d&q=danger+will+robinson#fpstate=ive&vld=cid:06d64c16,vid:OWwOJlOI1nU,st:0

- The YES statement of AntiCanCrashSelfMod contradictory!!!
  - program cannot output YES if it has crashed.
- Therefore such a program cannot exist
- QED

Optimizing IR

Distance over many words

- Problem: Find the minimum separation in a document of an unbounded number of words (over a set of documents)
- Two pairwise algorithms
  - O(n*m*D)
  - O((n+m)*D)

```
for D in documents
 for l1 in (w1 in D)
  for l2 in (w2 in D)
```

```
for D in documents
 While idx1<len1 and idx2<len2
  ...
```

- Can we use either of these algorithms directly for 3, 4, 5, 6, ... words?
  - if NOT, why?
    - What can we do?

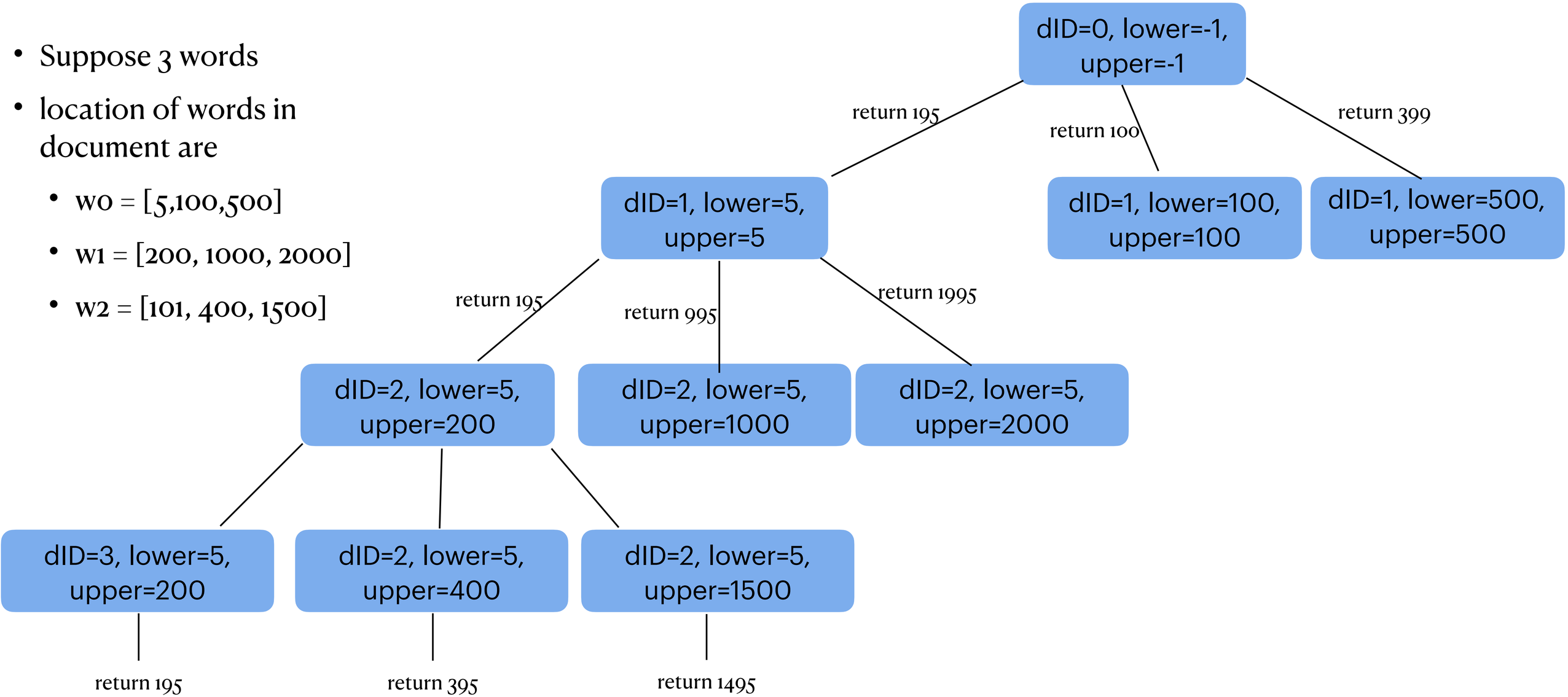
Recursion to the rescue!

- I -- inverted index
- W -- list of words
- wi -- index of the word to work on now
- d -- document id
- lower -- lower bound
- upper -- upper bound

```
func closest(I,W,wi,d,lower,upper)
    if len(W) <= wi
        return upper-lower
    wl = locations of W[wi] in d extracted from I
    best = length of d (in words)
    for wwll in wl
        let tl=lower
        let tu=upper
        if wwll < tl or tl < 0
            tl=wwll
        if wwll > tu
            tu=wwll
        let q = closest(I, W, wi+1, d, tl, tu)
        if q < best
            best = q
    return best
```

Walk through the algorithm

- Suppose 3 words
- location of words in document are
  - w0 = [5,100,500]
  - w1 = [200, 1000, 2000]
  - w2 = [101, 400, 1500]



Data

- emma elizabeth and but
  - 437 ms
- rob rich the and
  - 233 sec
- to be or not
  - 119 minutes (on lab computer)

- Works!!
- BUT it really slows down on common words
- Why
- What can we do?
  - Analyse!
    - Order Matters!

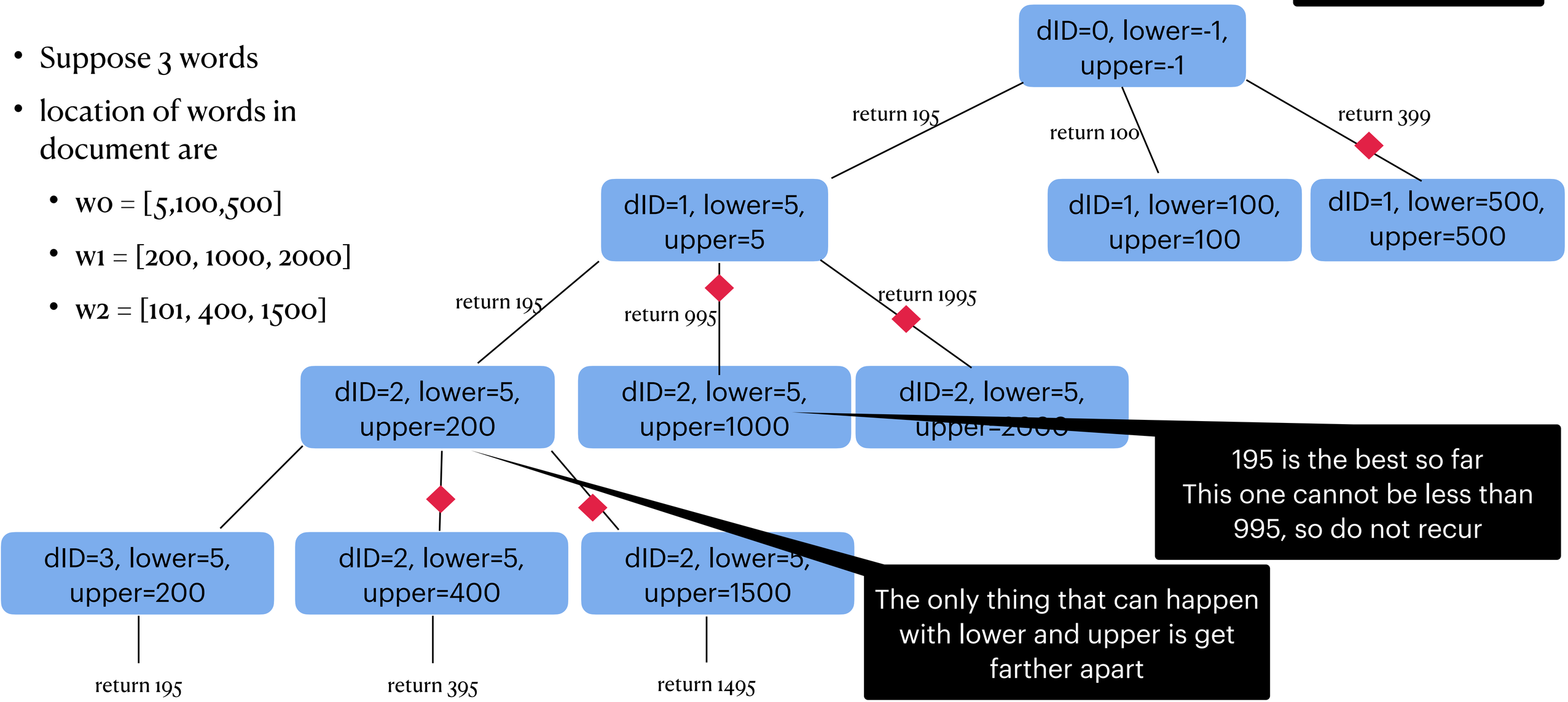
rob rich the and	250	emma eliz and but	239ms
the and rob rich	285	emma and but eliz	366
the rob rich and	258	and but emma eliz	276
rich the and rob	179	eliz and but emma	66

- tentative conclusion: start smallest, then largest, to smallest

Looking at the Algorithm

Reminiscent of alpha,beta pruning in game play

- Suppose 3 words
- location of words in document are
  - w0 = [5,100,500]
  - w1 = [200, 1000, 2000]
  - w2 = [101, 400, 1500]



195 is the best so far
This one cannot be less than 995, so do not recur

The only thing that can happen with lower and upper is get farther apart

Data V2

- emma elizabeth and but
  - 18ms
    - @30x
- rob rich the and
  - 958 ms
    - @200x
- to be or not
  - 9.6 sec

- Works!!
- Speedup of 1--400+
  - speedup depends on how much can be pruned
  - No change to worst case complexity

Wisdom is to reorder words in query

rarest first

most common next

Can we do better?

Where is the time going?

Lots of instrumentation later

the transformation of [{document, location}..]

into [location...]

In particular, at the lowest level of the recursion, my code does this A LOT

each time it does this, it throws the result away!

Further improve v2?

- Where is the time going?
- Lots of instrumentation later
  - Lots of time is going into
    - the transformation of [{document, location}..]
      - into [location...] for a single document
    - and then garbage collecting
- Why?
  - Being done at every level of recursion
    - lowest level of the recursion does this A LOT
      - each time it does this, it throws the result away!
      - Meta question: Do we really even need this transformation?

V3

do the [{docId loc} ...] to [loc,...] transition exactly once!

- docLocs -- a 2-d array holding the locations in documents of the exact words
- W -- list of words
- wi -- index of the word to work on now
- d -- document id
- lower -- lower bound
- upper -- upper bound

```
func closest(docLocs,W,wi,d,lower,upper)
    if len(W) <= wi
        return upper-lower
    wl = docLocs[wi]
    best = length of d (in words)
    for wwll in wl
        let tl=lower
        let tu=upper
        if wwll < tl or tl < 0
            tl=wwll
        if wwll > tu
            tu=wwll
        let q = closest(I, W, wi+1, d, tl, tu)
        if q < best
            best = q
    return best
```

Data V3

- emma elizabeth and but
  - 1.3ms
    - @10x over v2
- rob rich the and
  - 63 ms
    - @15x over v2
- to be or not
  - 178ms
    - @100x over v2

- Works!!
- Speedup of 30-200+

Can we do better?

Where is the time going?

Lots of instrumentation later

the transformation of [{document, location}..]

into [location...]

In particular, at the lowest level of the recursion, my code does this A LOT

each time it does this, it throws the result away!