Lab 1

Analysis and thoughts

Linear Search

- Linear Search is a "MacGuffin"
  - The Maltese Falcon, The Grail, The Golden Fleece, Hitchcock
- Algorithmic improvements
- Empirical Studies
  - how to get good results
- Recursion
  - the Stack Overflow problem
  - Tail Call Optimization

Linear Search

- General Idea:
  - start at beginning of list
  - step through list comparing list item with target to find the location of target in list
  - return the target location or -1 if target not in list
    - Why -1?

LinearSearch: Cormen's first Algorithm

```
procedure linearSearch(A, n, x)
Inputs:
 •A: An array (indexed from 0 onwards)
 •n: the number of elements in A to
  search through
 •x: the value being searched for.
Outputs: Either an index i for which
A[i] = x, or a special value, -1
1.Set answer to -1.
2.For each index i=0 to n-1:
  a.If A[i] = x then set answer to i.
3.Return the value of answer
```

- For each item in the list, we have to
  - increment i
  - compare i to n
  - compare A[i] to x

- If we assume that each of these takes the same amount of time (t), then total time is
  - 3tN??
    - should "increment i" be 1 or 2 (or 3) operations?
    - should "compare i to n" be 1 or 3?
    - how many should compare A[i] to x be
      - 1,3,5, more?

Primitive Operations

even knowing this in a PL is hard

```java
public class BasicOps {
    public static void main(String[] args) {
        long tt = 0;
        long maxx = (1 * 1000000000);
        for (int j = 0; j < 10; j++) {
            long i = 0;
            long st = System.nanoTime();
            while (i < maxx) {
                i = i + 1;

            }
            long nd = System.nanoTime();
            System.out.format("%2d %d\n", j, (nd - st) / 1000);
            tt += (nd - st);
        }
        System.out.format("AV %d\n", (tt / 10000));
    }
}
Average time: 0.362 seconds
```

```java
public class BasicOps {
    public static void main(String[] args) {
        long tt = 0;
        long maxx = (2 * 1000000000);
        for (int j = 0; j < 10; j++) {
            long i = 0;
            long st = System.nanoTime();
            while (i < maxx) {
                i = i + 1;
                i = i + 1;

            }
            long nd = System.nanoTime();
            System.out.format("%2d %d\n", j, (nd - st) / 1000
            tt += (nd - st);
        }
        System.out.format("AV %d\n", (tt / 10000));
    }
}


Average time: 0.360 sec
```

Better Linear Search

```
procedure betterLinearSearch(A, n, x)
Inputs & Outputs: Same as Version 1
1.For each index i=0 to n-1:
  a.If A[i] = x then return value of i.
2.Return -1
```

- Same operations as LinearSearch
- if x is in A
  - on average will only look at 1/2 of A
  - time = 3tN / 2
- If x NOT in A
  - no different from linear search
  - time = 3tN / 2

- EXPECTATION:
  - if x is at least sometimes in A, then this will be faster than linearSearch

Sentinel Linear Search

```
procedure SentinelLinearSearch(A, n, x)
Inputs & Outputs: Same as Version 1
1.Last ← A[n-1]
2.A[n-1] ← x.
3.Set i ← 0.
4.While A[i] ≠ x:
  a.Increment i by 1
5.A[n-1] ← Last
6.If i < n-1 or A[n-1] = x then return value of i.
7.Otherwise return -1
```

- Idea, guarantee search success so do not need to check i < n
  - removes one operation from loop
- Time:
  - x is in A: 2tN/2 ==> tn
  - x is NOT in A: 2tN
- EXPECTATION:
  - regardless of whether x is in A, this will be faster than linear search or better linear search.

Hypotheses / Expectations

- If x is in A,
  - better will be faster than basic
    - by about a factor of 2
  - sentinel will be faster than better
    - by about a factor of 1.5
- if x not in A,
  - better will be the same as basic
  - sentinel will be faster than basic
    - by about a factor of 1.5

Test Procedure

Times when x is in A

- First size problem so times recorded are "large" enough
  - Why large times anyway?
- Java on my laptop
  - A length is 10,000,000
  - Number of searches is 300
    - Why not just 1 search over 3,000,000,000?

- Procedure:
  - Populate A with 10,000,000 random number in range 0..100,000,000
  - targets <= int[300]
  - for i in 0..300
    - rr = random(10,000,000)
    - targets[i] = a[rr]

Timing

Beware the Jabberwock!

```java
public class NanoT {
    public static void main(String[] args) {
        long l = System.nanoTime();
        long s = l;
        long t = 0;
        int c = 0;
        double d = 0.0;
        for (int i = 0; i < 10000000; i++) {
            long m = System.nanoTime();
            t += (m - l) / 10;
            c += (m - l) / 10.0;
            l = m;
        }
        long n = System.nanoTime();
        System.out.println(t);
        System.out.println((n - s)/10);
        System.out.println(d);
    }
}
```

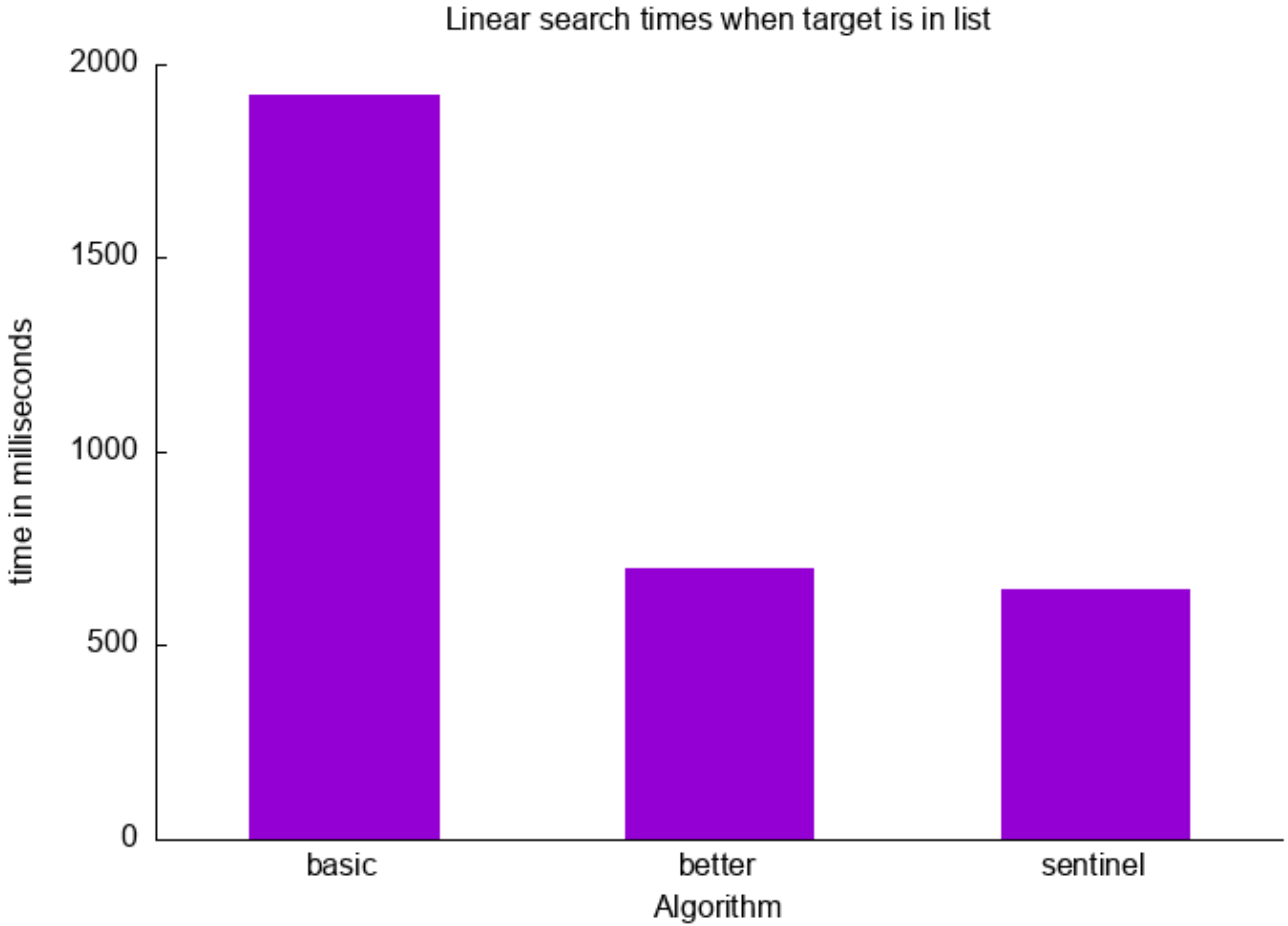
Jabberwocky

By Lewis Carroll

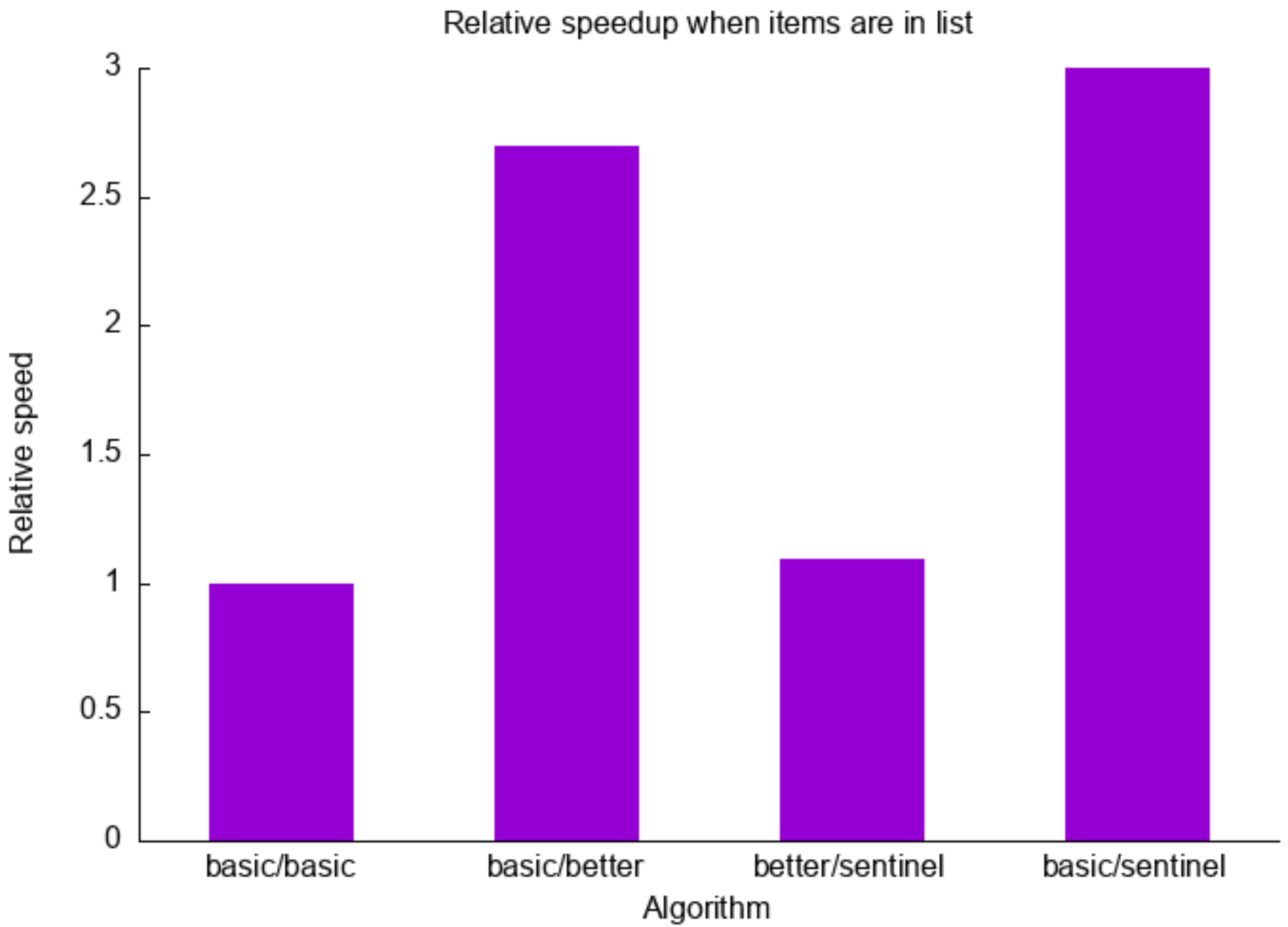
’Twas brillig, and the slithy toves
      Did gyre and gimble in the wabe:
All mimsy were the borogoves,
      And the mome raths outgrabe.

“Beware the Jabberwock, my son!
      The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
      The frumious Bandersnatch!”

....



Linear search times when target is in list
2000
1500
1000
500
0
time in milliseconds
basic
better
sentinel
Algorithm

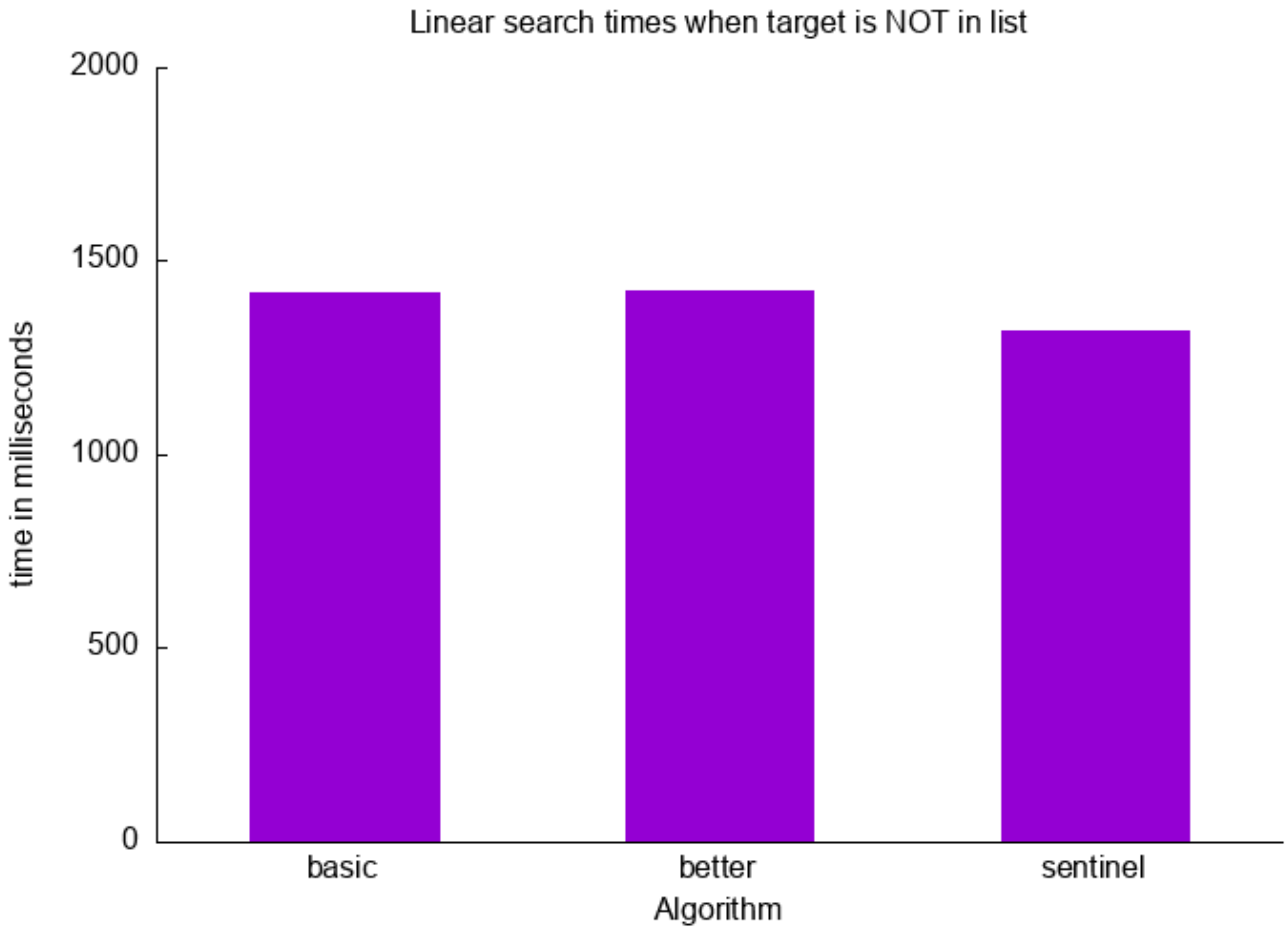


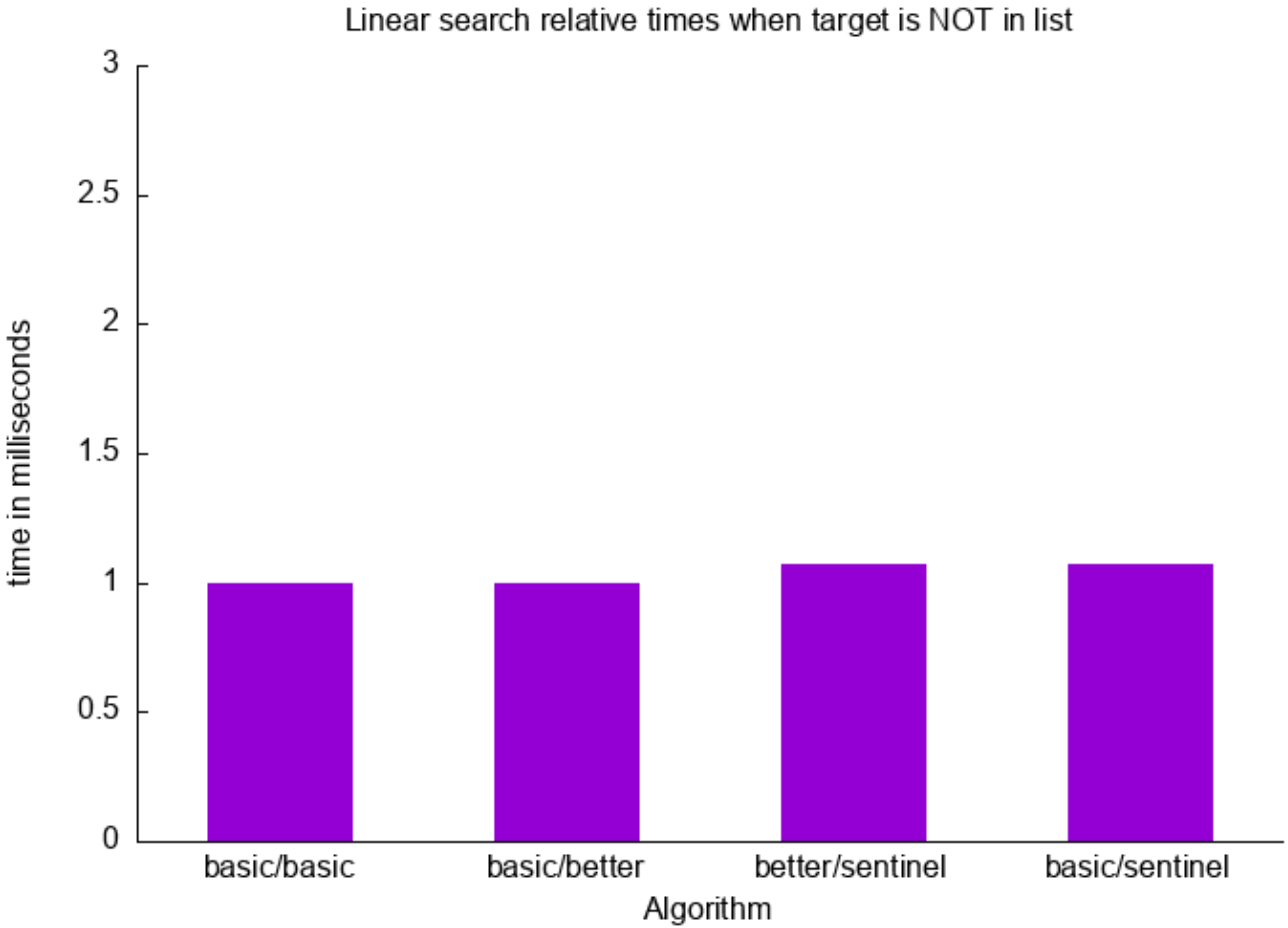
Relative speedup when items are in list
3
2.5
2
1.5
1
0.5
0
Relative speed
basic/basic
basic/better
better/sentinel
basic/sentinel
Algorithm

Thoughts

- Times are roughly in line with expectations BUT
  - basic - better outperforms
  - better - sentinel underperforms
- Why??
  - could / should look deeper here but other topics seem more interesting

	Expected Speedup	Observed Speedup
Basic - better	2.0	2.7
Better - Sentinel	1.5	1.09
Basic - Sentinel	3.0	3.0

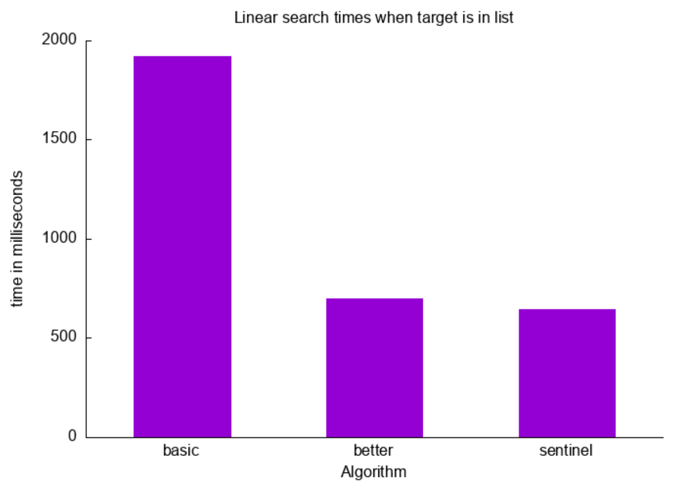

Linear search times when target is NOT in list
2000
1500
1000
500
0
time in milliseconds
basic
better
sentinel
Algorithm

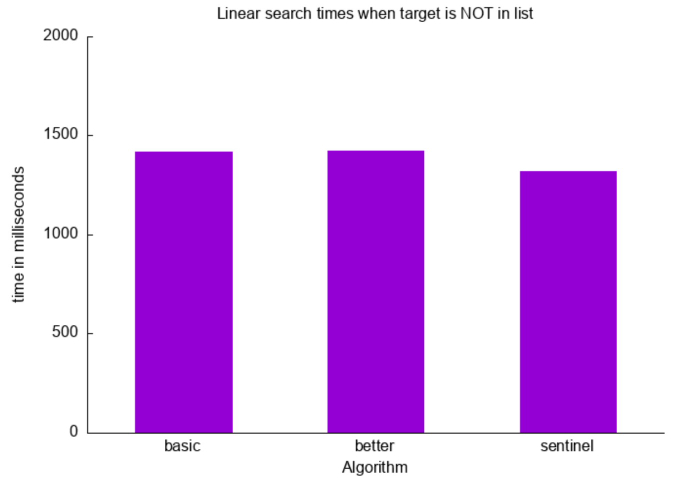


Linear search relative times when target is NOT in list
3
2.5
2
1.5
1
0.5
0
time in milliseconds
basic/basic
basic/better
better/sentinel
basic/sentinel
Algorithm

Deeper thoughts

- Follows expectations / previous
  - better and basic essentially identical
  - sentinel better by about 1.1x

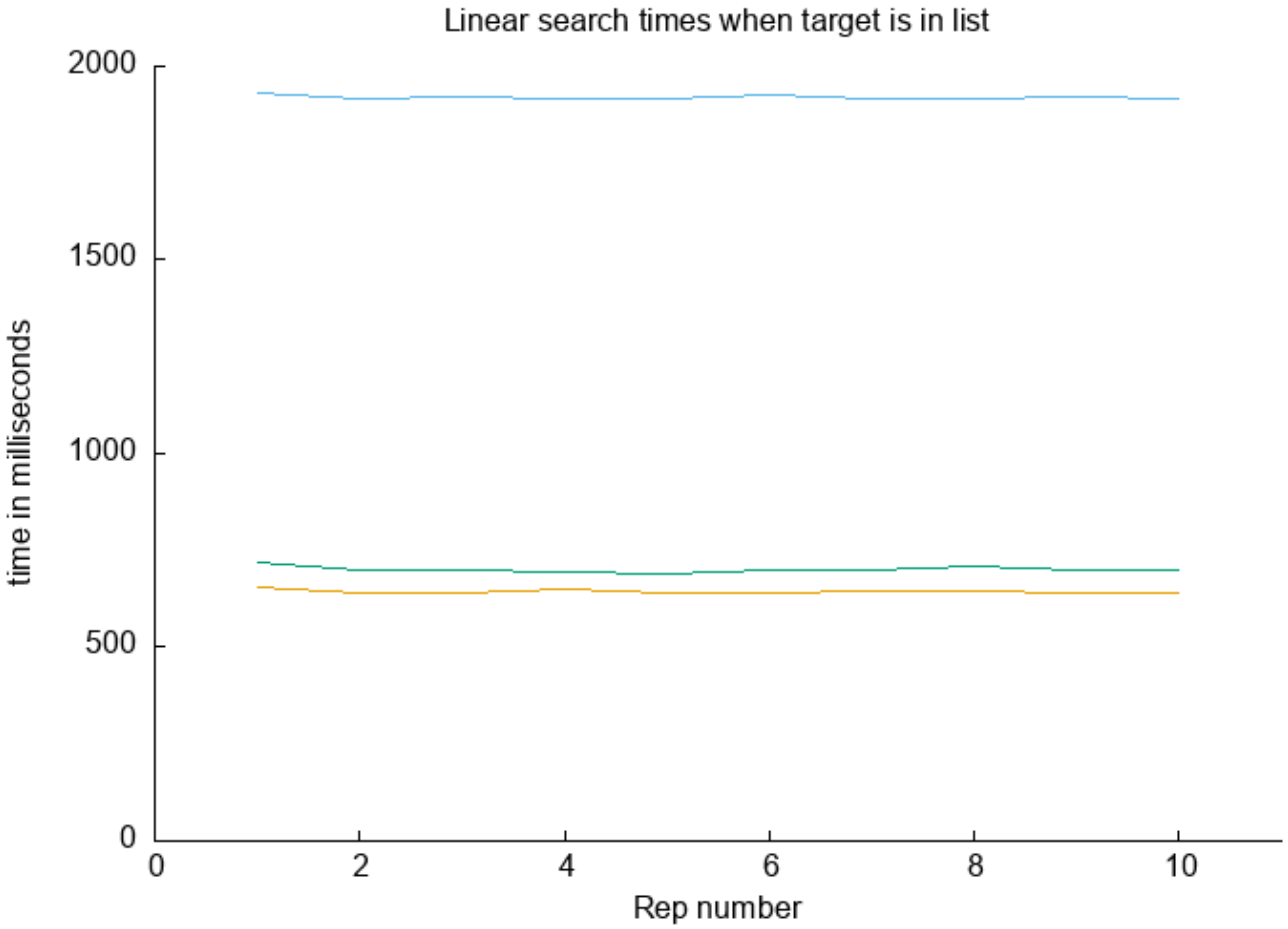




- BUT
  - times for basic between items are in and items are not in
    - In: 1920ms   Not In: 1420ms
    - Expectation: identical, results: 1.35x speedup on a set of numbers NOT in the list being searched
      - ARGGGG
      - Why is "not in" so much faster for basic??
        - happens in java when: below range, above range (better is unaffected)
        - does not happen in java when: item is the last (ie the worst case test)
        - does not happen in Rust.

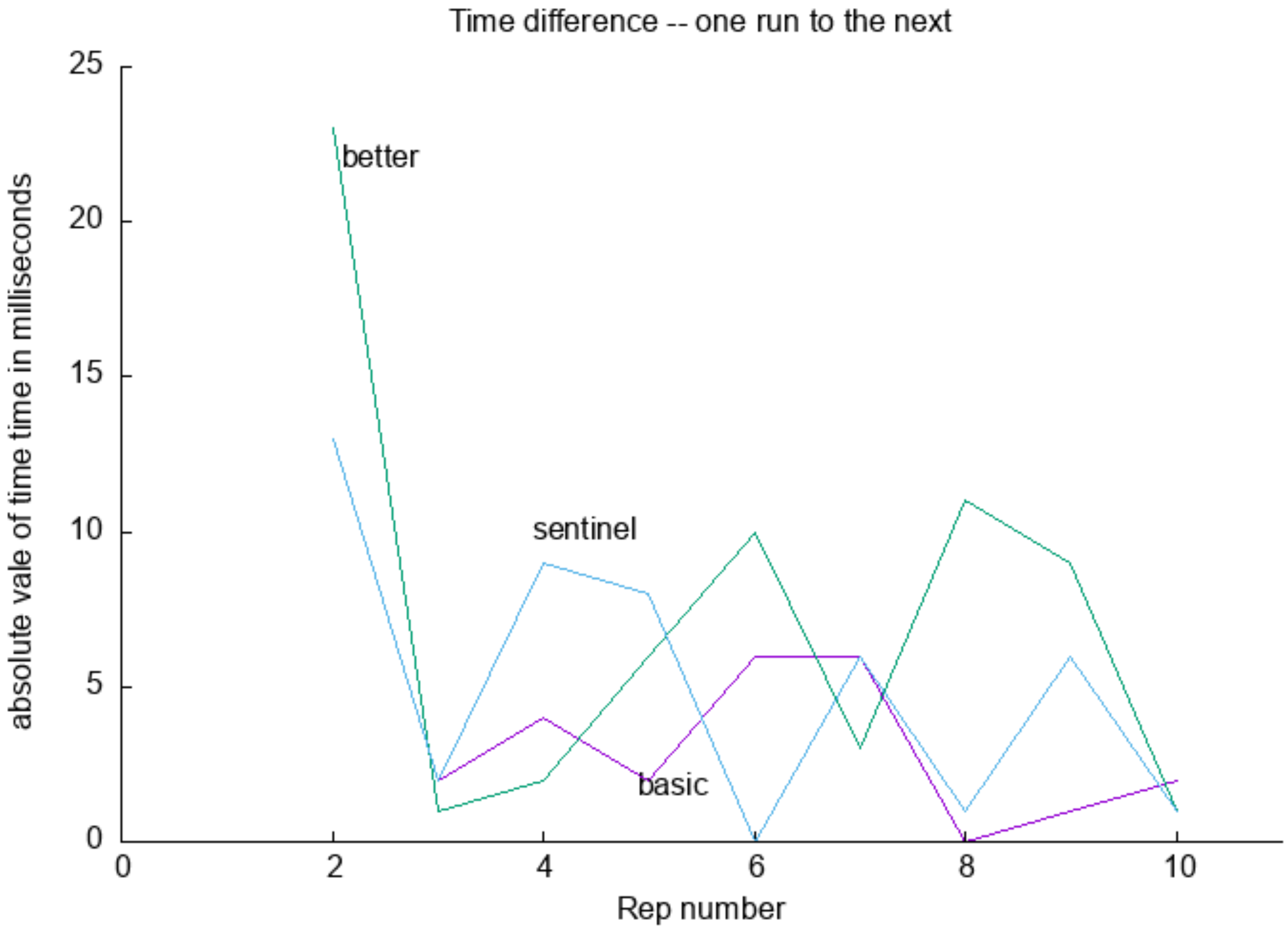
More Thoughts

- All data is average over 10 runs
  - Select 10,000,000 randoms
  - Select 300 targets
  - For i in 1..10
    - run linear -- get time
    - run better -- get time
    - run sentinel -- get time
- So conditions are as identical as I can make them



Inconsistent Times

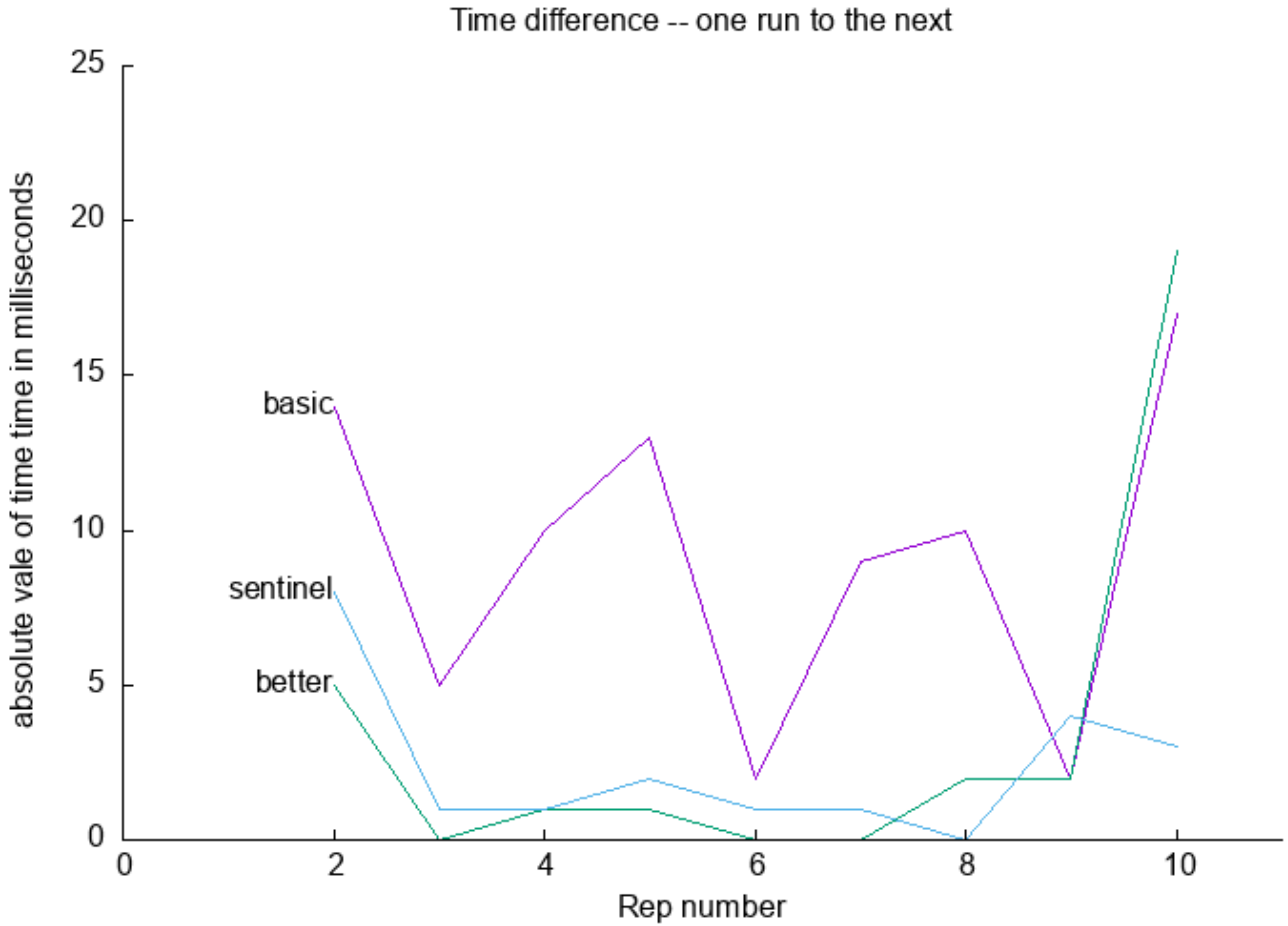
- Differences are small by they should be 0
- Why
  - 2 sources of inconsistency
    - Java
      - startup -- JVM issues
      - Other
    - Laptop



Inconsistent Times

Taken on a Lab machine

- Differences are smaller
  - Laptop a definite issue
  - Still have Java issues
    - Startup
    - Other
      - What is "other"

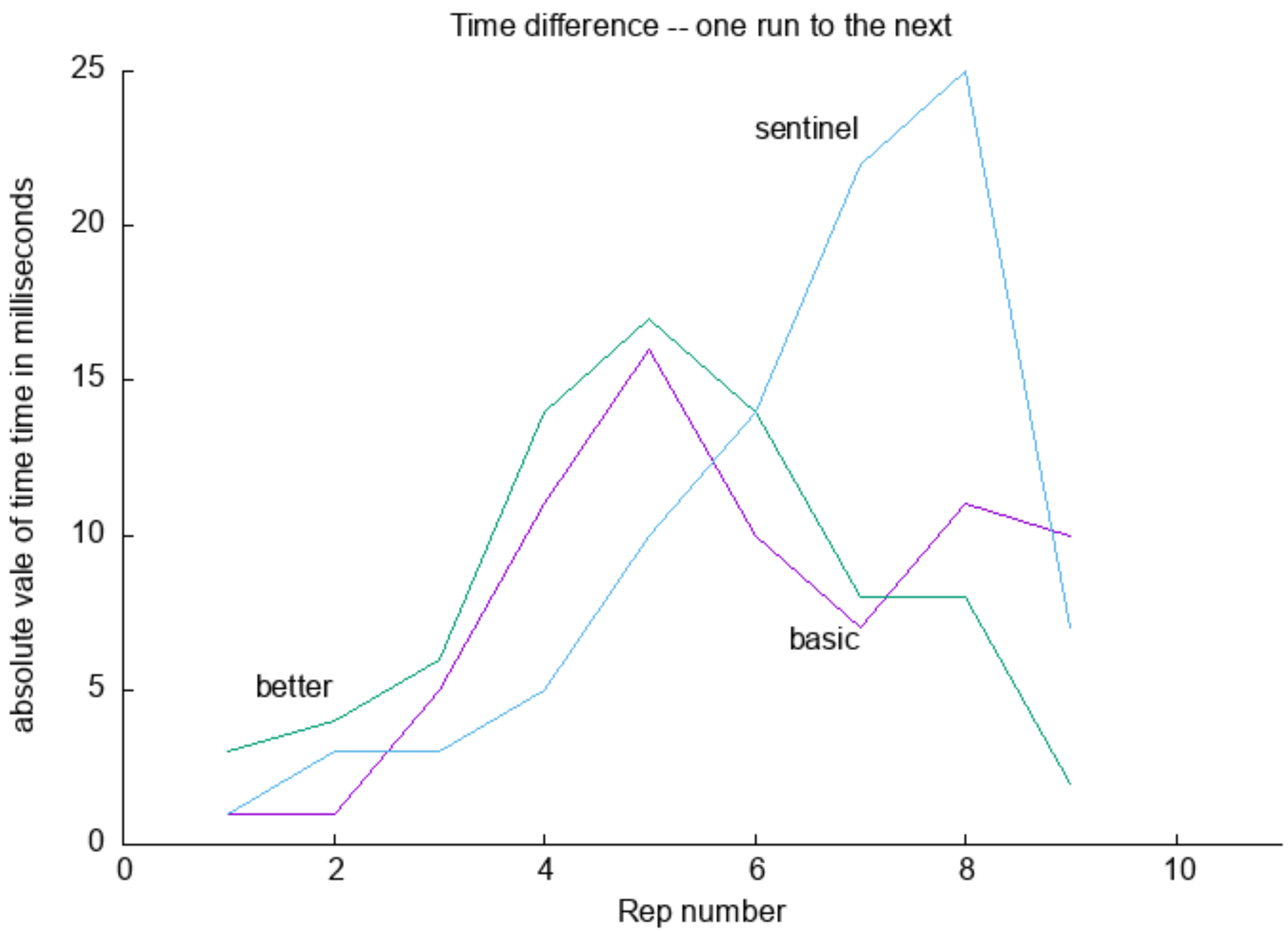


Hypothesis -- Garbage Collection

- Java has Garbage collection
- Garbage Collector runs when it chooses to run
- Realistically this program generates very little garbage, but ...
- Is this a testable hypothesis?
  - How?
    - Cannot just turn off GC
  - Replicate in a language without GC?

Turning off CG

- Cannot be done in Java
- So replicate code in Rust
  - No java "startup" issue
  - Still inconsistent!
  - Rust has a lot of debug stuff
    - Run for release??



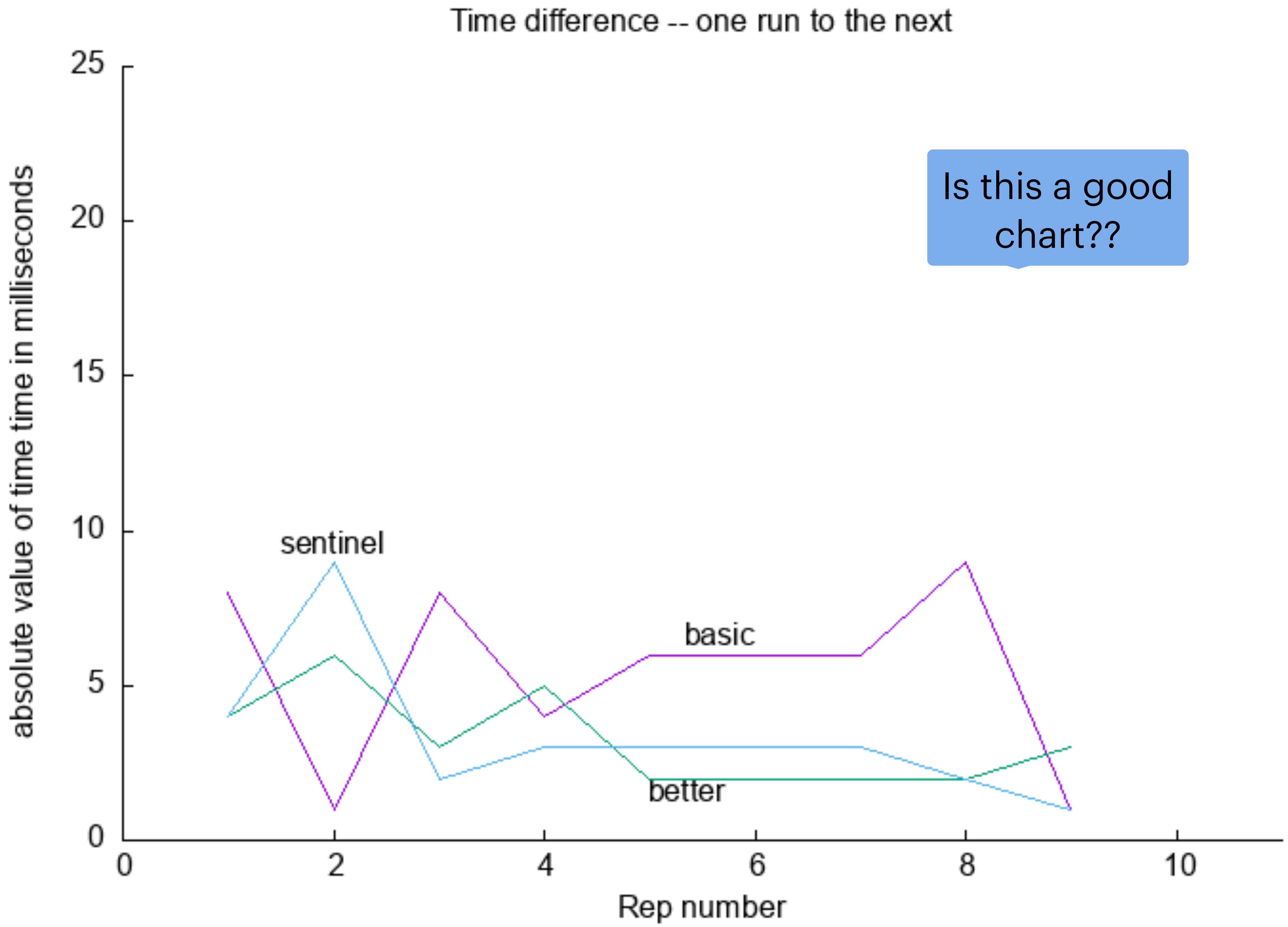
Rust reports 0ms runtime

When compiled for release

```
fn f1w(search_in: &Vec<u64>, search_for:&Vec<u64>) -> (Duration, i32) {
   let now = Instant::now();
   for i in 0..search_for.len() {
        let mut found = false;
        let tgt = search_for[i];
        let mut ii = 0;
        while ii<search_in.len() {
            if search_in[ii]==tgt {
               found = true;
            }
            ii += 1;
        }
    }
    return (now.elapsed(), nfc);
}
```

Rust actually gets pretty stable

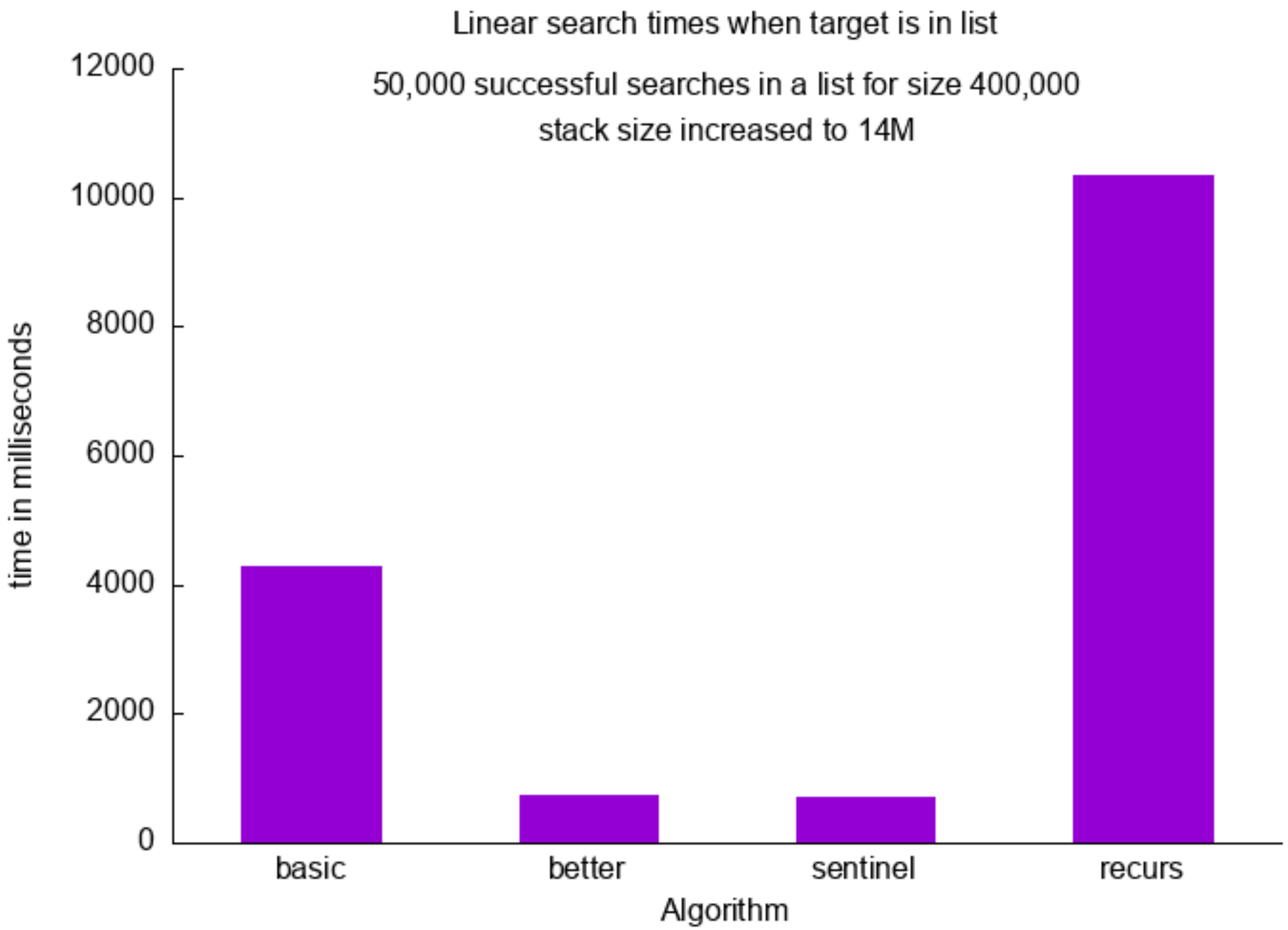
	Expected Speedup	Observed Speedup
Basic - better	2.0	1.35
Better - Sentinel	1.5	1.02
Basic - Sentinel	3.0	1.4



Is this a good chart??

Recursive Search

- Slow!!
- Stack Overflow!
  - java -Xss 1024000
    - 1MB the default
    - ~15000 before overflow at 1MB
    - to get chart used 14M of stack space



Tail Recursion Optimization

- basically rewrites recursion as a loop
- No stack overflow

- Very few procedural languages that have this
  - gcc -- with -O2 or higher
    - 2.16sec vs 0.000204sec -- 10000X speedup
      - more commonly <100x
  - kotlin -- if the function is marked for tail call optimization

Contributions

- Empirical results are roughly in line with Cormen and my theoretical analysis
  - primitive operations probably do NOT have equal cost
  - Whatever you code, the compiler might do something else!
- Empirical results are noisy
  - Java
    - startup effects
      - Ignore the first few results
    - GC
      - Always get averages over lots of samples
      - If possible, write code to minimize garbage generation
  - Rust -- no startup, no GC but still some noise!
  - Laptop effects
- Recursion
  - When you can, use a loop
  - When you cannot, worry about Stack Overflow

Winston on Presentations

Pick

- Time
- The room
  - Shape matters (Park 227, Park 338)
  - A happy place

Practice

- Pick your location
- AV issues
- Lights on
- Chat up early arrivers

The talk

- Be Happy
- VSN-C
  - Start with Vision
  - Steps
  - News
  - Finish with Contributions

Contributions == Conclusions

- No "thank you"
- No collaborators
  - if needed, do early

"you have too many slides and all of them have too many words"

Winston

Do not read

No cute clip art

Avoid bullet lists

Use big fonts

(use even bigger fonts)

Progress bars -- maybe

"page 1 or 12"?

Props

Titles

Um
like
er...
you know

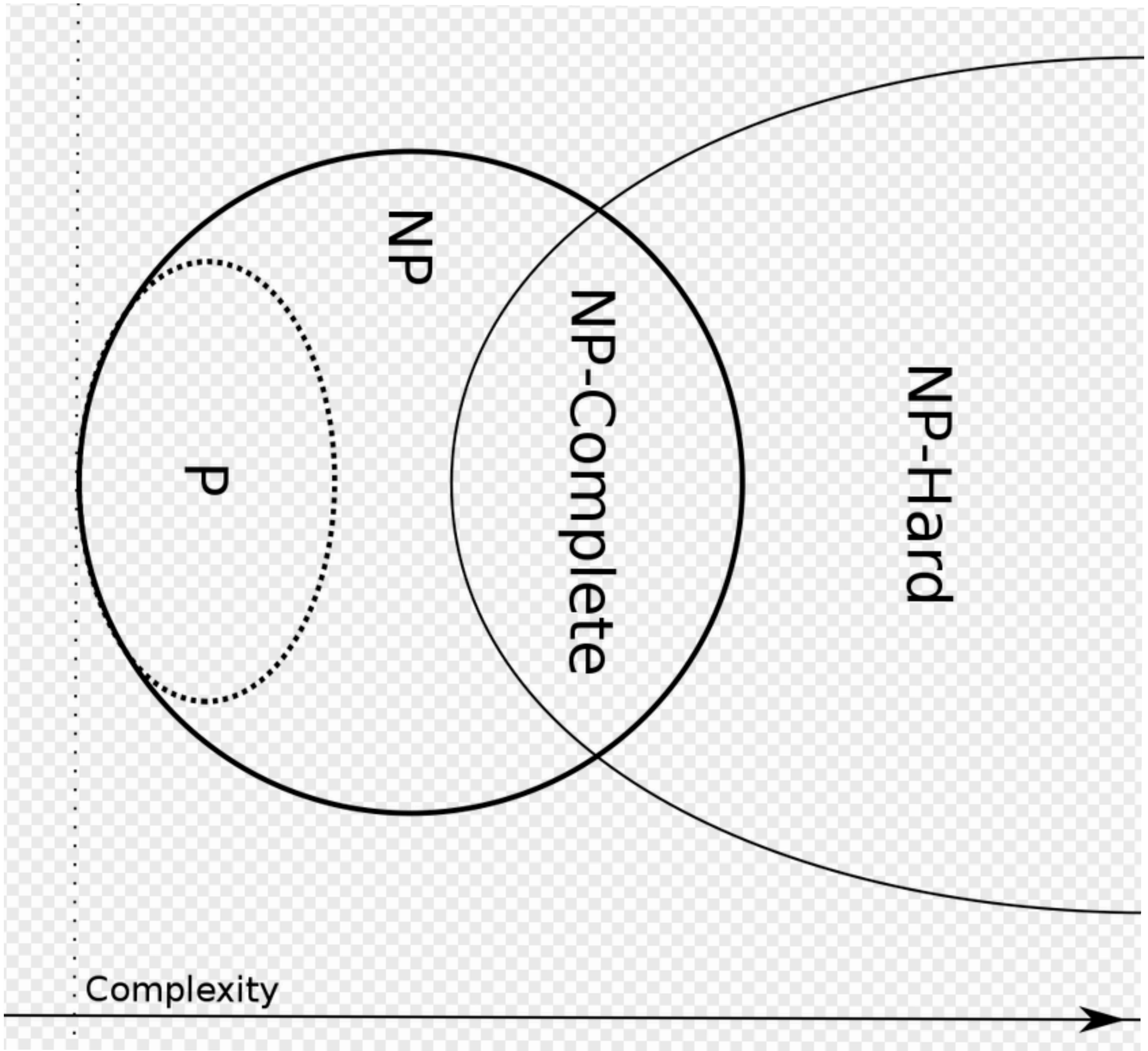
Bellow!

(use a mic, practice)

~~Monotone~~

~~Pockets~~

P = NP?

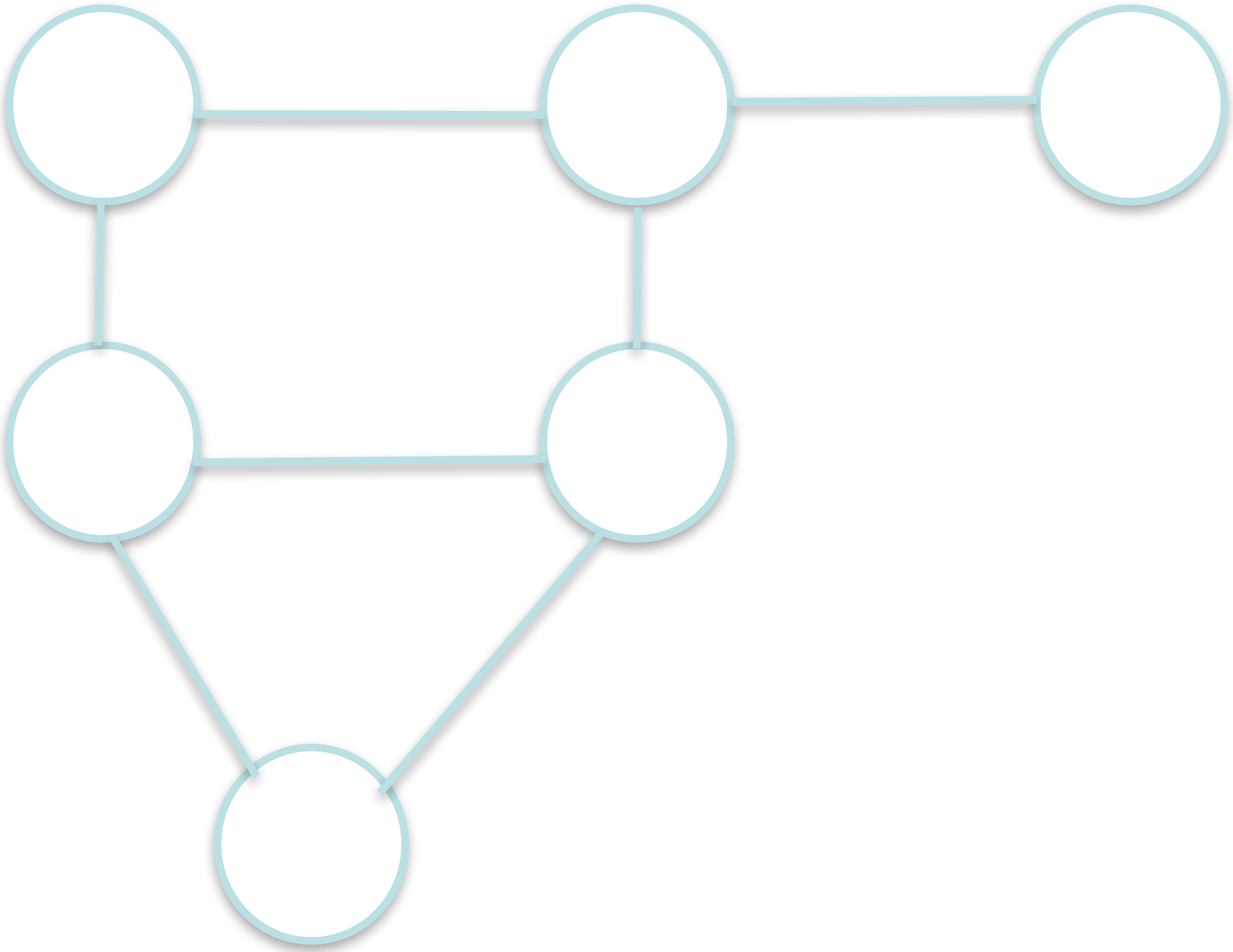


from wikipedia

NP-Complete

- NP = Non-determinitic Polynoimial
- in NP == Solution is verifiable in P time
- problem is provably equivalent to other NP complete problems

- vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge.



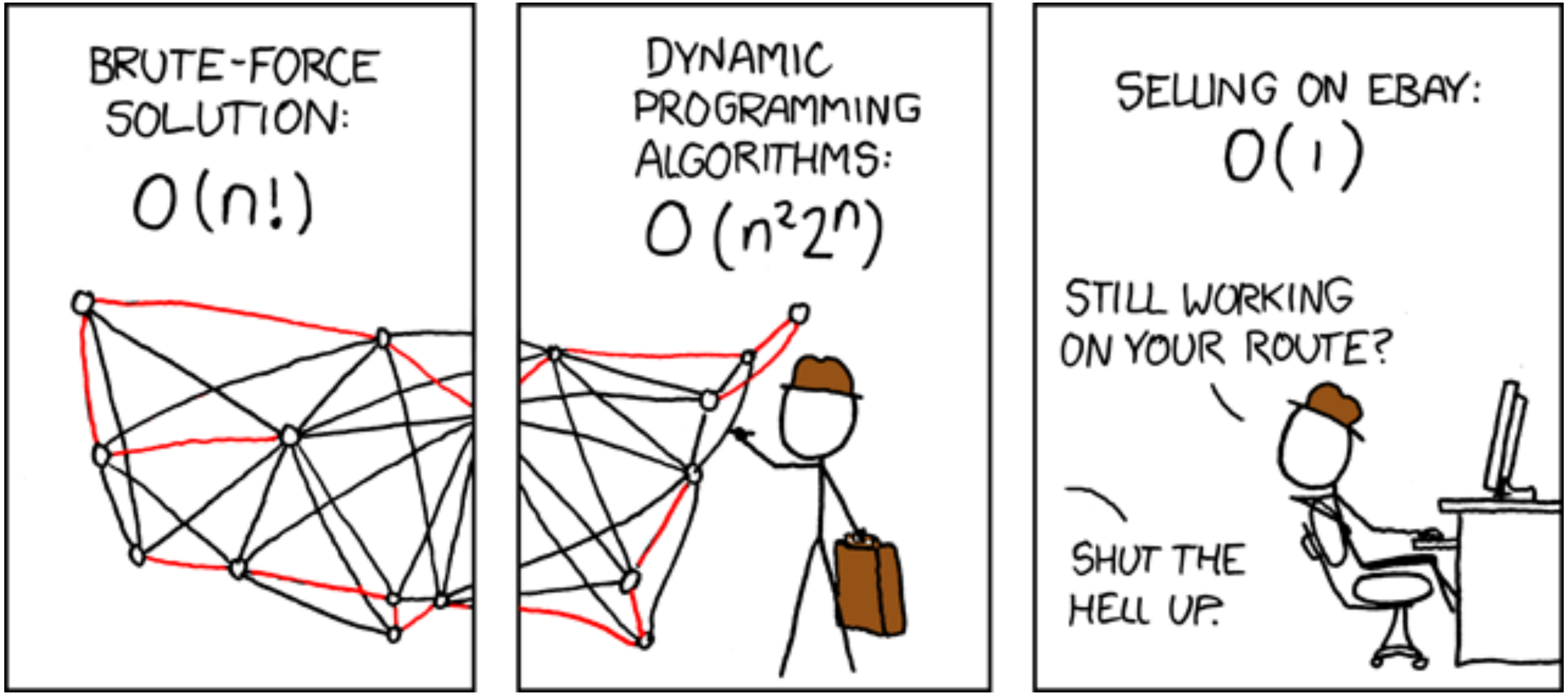
Vertex Cover Algorithm

- Find the minimum vertex cover of a graph
  - We will discuss graph representations, just make something up for now

Vertex Cover Algorithms

- Optimal algorithm
- Naive algorithm
- Greedy Algorithm

xkcd??



- [More on xkcd.com](https://xkcd.com)

An algorithm to consider

- Given two lists of integers
  - call these A and B
- Find: min(abs(A[i]-B[j])