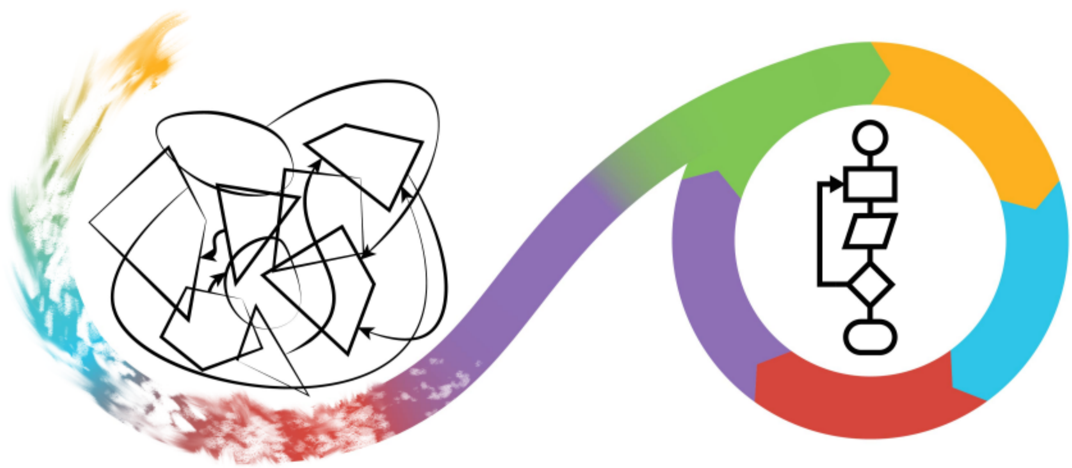
Optimisation

Richèl Bilderbeek

1 The Big Picture

https://github.com/UPPMAX/programming_formalisms/blob/main/tdd/tdd_lecture/tdd_lecture.qmd



1.1 Breaks

Please take breaks: these are important for learning.

It can sometimes be painful/annoying when there is a break in the middle of the exercise.

Ideally, do something boring (1)!

1.2 Schedule

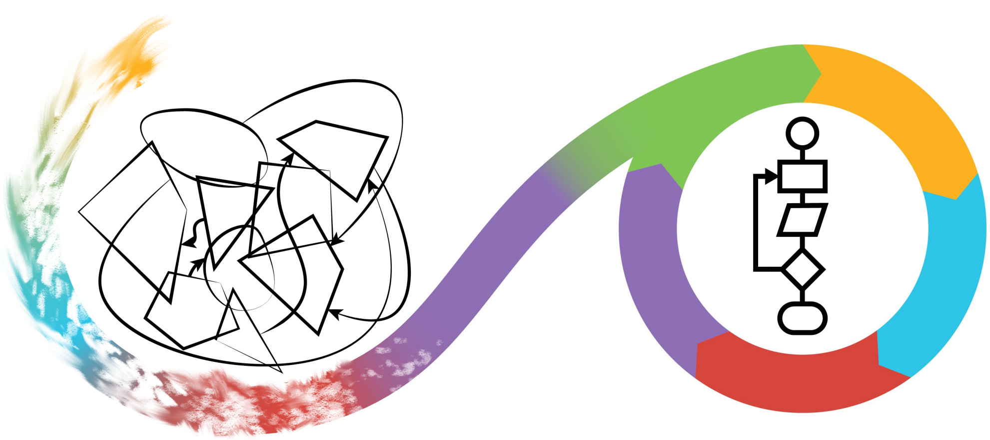
From	To	What
12:00	13:00	Lunch
13:00	13:45	Discuss Retrospect, misconceptions, get a speed profile
13:45	14:00	Break
14:00	14:45	Get a speed profile, ?case study
14:45	15:00	Break
15:00	15:30	Course recap, Open discussion
15:30	16:00	Reflection

2 Retrospect

- [ ] Discuss

3 Optimisation

https://github.com/UPPMAX/programming_formalisms/blob/main/optimisation/optimisation_lecture/optimisation_lecture.qmd



4 Why optimization?

To improve the runtime speed (or memory use) of a program

Captain Obvious

5 Misconceptions

Q: What would be **bad advice** to improve the run-time speed of an algorithm?

Fill in in the shared document!

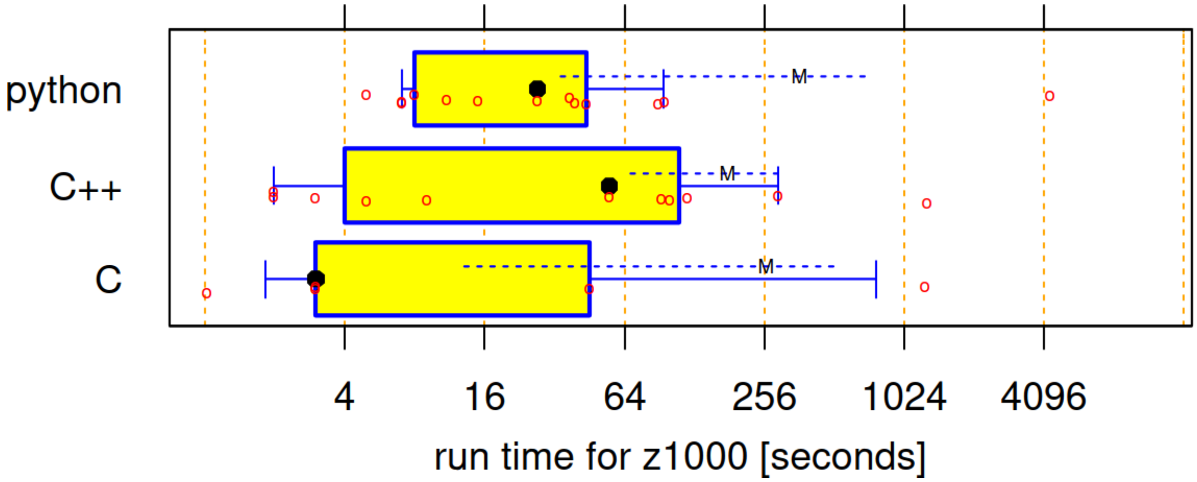
(if you dare and have time: add good advice too)

5.1 Bad advice 1

'Use C or C++ or Rust'

. . .

Variance within programming languages is bigger than variance between languages (adapted fig 2, from (2))



5.2 Bad advice 2

'No for loops', 'unroll for-loops', any other micro-optimization.

. . .

Premature optimization is the root of all evil. It likely has no measurable effect.

5.3 Bad advice 2

> We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.
>
> Donald Knuth
>
> Source: Wikipedia

5.4 Bad advice 3

'Always parallelize'

. . .

- Maximum gain depends on proportion spent in the parallelized part (3)
- Overhead is underestimated

Figure 1: Donald Knuth

- Hard to debug

5.5 Bad advice 3

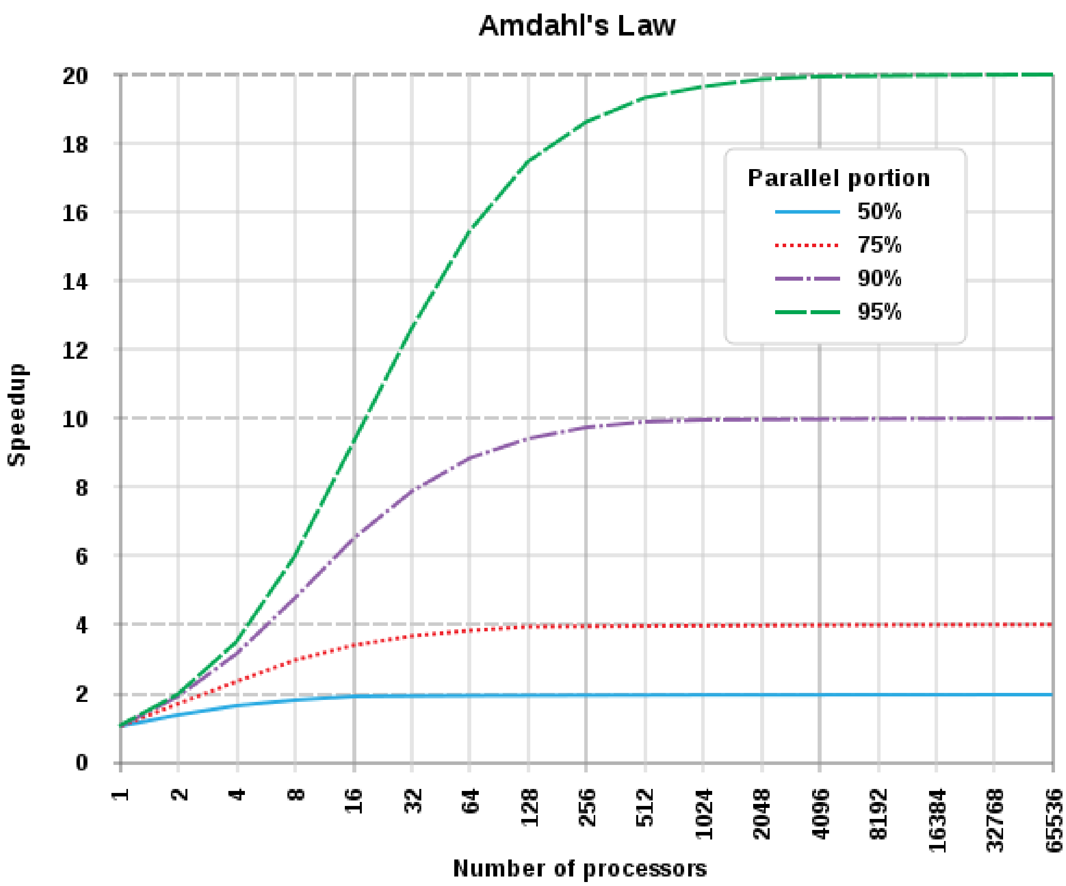


Figure 2: https://en.wikipedia.org/wiki/File:AmdahlsLaw.svg#file

5.6 Bad advice 4

'Optimize the function where you feel the performance problem is'

Developers -also very experienced developers- are known to have a bad intuition (4)

Instead, from (5):

1. finding code program spends most time in

2. measure timing of that code
3. analyze the measured runtimes

5.7 Bad advice 5

'Optimize each function'

- The 90-10 rule: 90% of all time, the program spends in 10% of the code.
- Your working hours can be spent once

6 Proper method

6.1 Problem

Q: When to optimize for speed?

. . .

A:

- C++ Core Guidelines: Per.1: Don't optimize without reason
- C++ Core Guidelines: Per.2: Don't optimize prematurely
- C++ Core Guidelines: Per.3: Don't optimize something that's not performance critical

6.2 Problem

Q: How to improve the run-time speed of an algorithm?

. . .

> Make it work, make it right, make it fast.
>
> Kent Beck

A (simplified):

1. Measure (hard to do (6))
2. Think
3. Change code
4. Measure again

6.3 Problem

Q: How to improve the run-time speed of an algorithm?

A (simplified):

1. Measure big-O
2. Measure speed profile
3. Think
4. Change code
5. Measure again

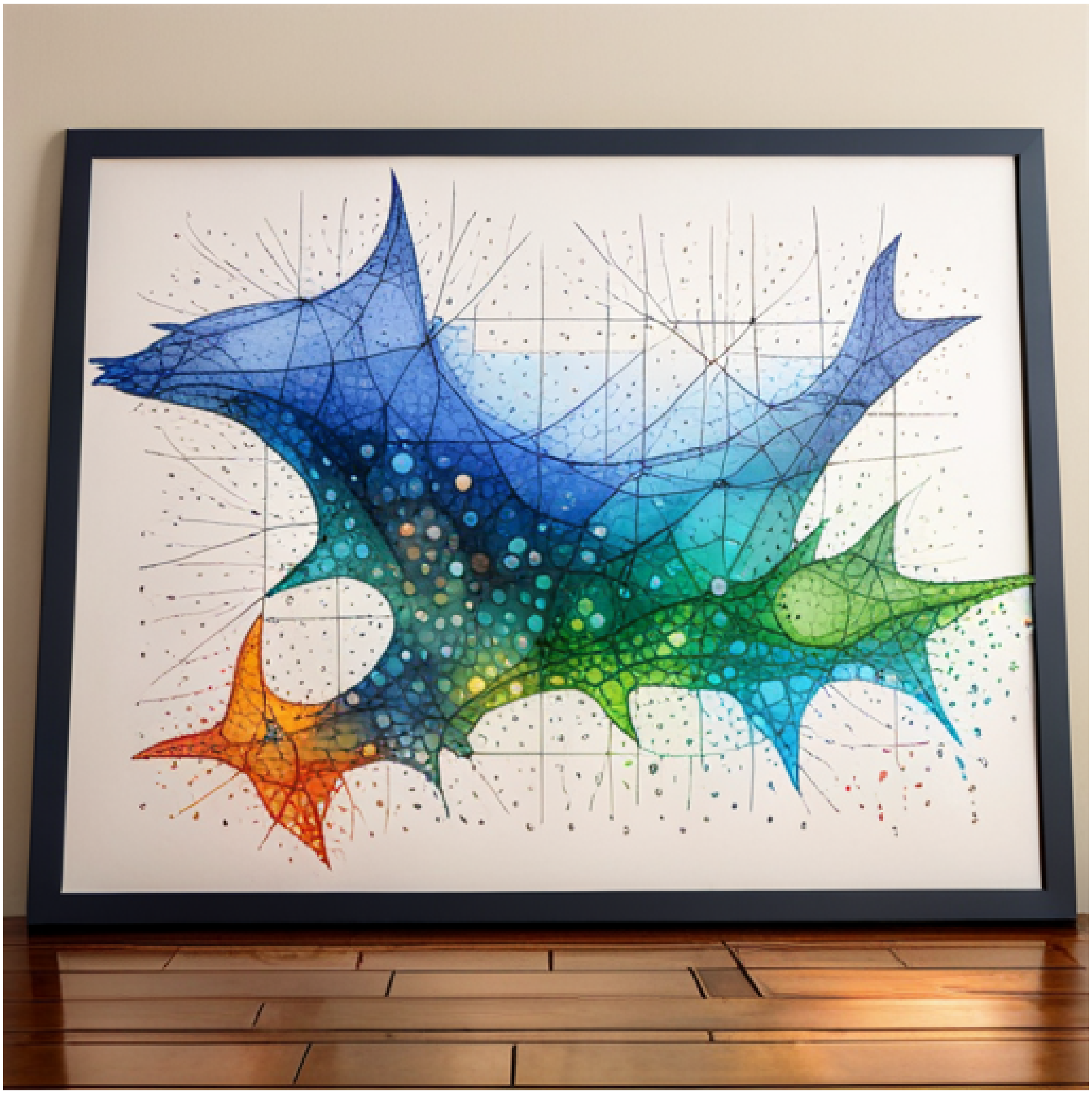
6.4 Measurement 1: big-O

How your (combination of) algorithms scales with more complex input.

- Counting the words in a book: O(n)
- Looking up a word in a dictionary: O(log2(n))

Do measure big-O in release mode!

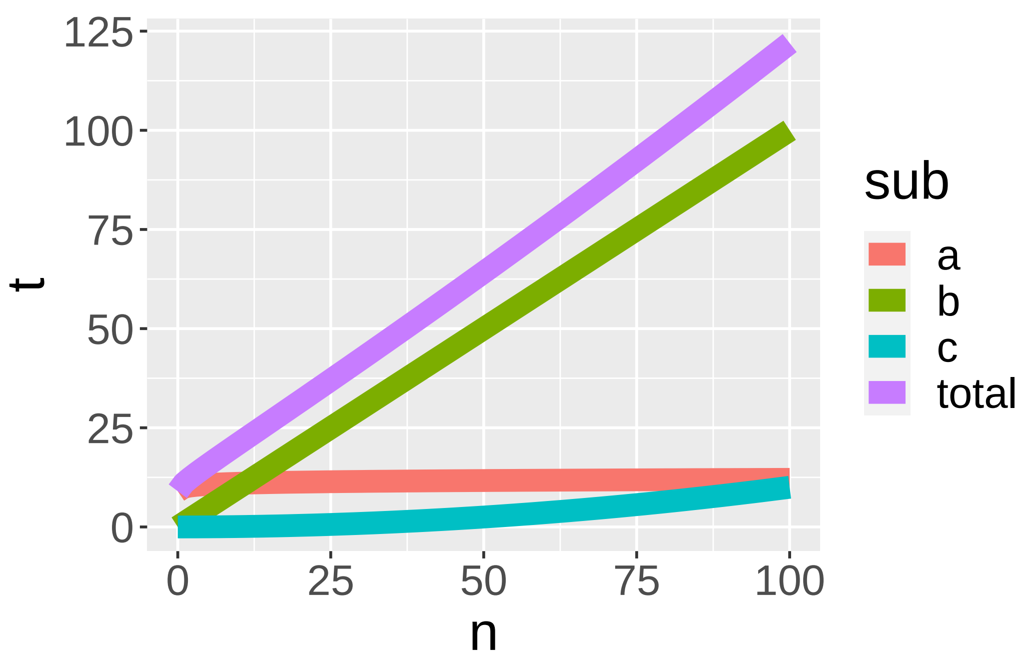
6.5 Your algorithm



6.6 Example

```
create_big_o_example <- function(n = seq(0, 100)) {
  t_wide <- tibble::tibble(n = n)
  t_wide$a <- 10 + log10(t_wide$n + 0.1)
  t_wide$b <- t_wide$n
  t_wide$c <- 0.001 * (t_wide$n ^ 2)
  t_wide$total <- t_wide$a + t_wide$b + t_wide$c
  t <- tidyr::pivot_longer(t_wide, cols = c("a", "b", "c", "total"))
  colnames(t) <- c("n", "sub", "t")
  t
}
t <- create_big_o_example(n = seq(0, 100))
ggplot2::ggplot(t, ggplot2::aes(x = n, y = t, color = sub)) +
  ggplot2::geom_line(size = 4) +
  ggplot2::theme(text = ggplot2::element_text(size = 20))
```

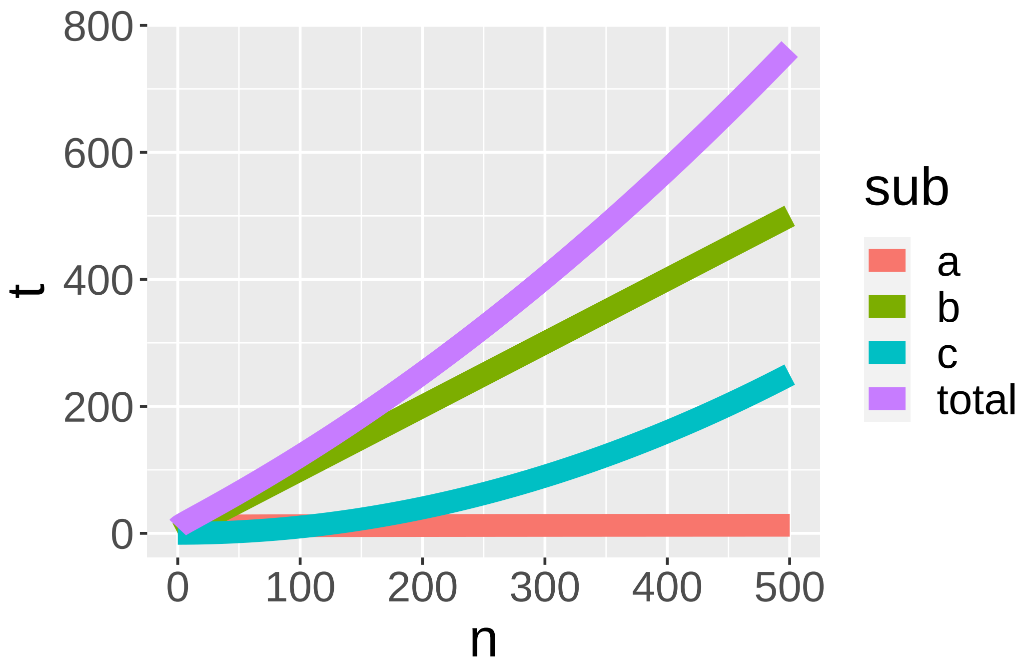
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



Work on B?

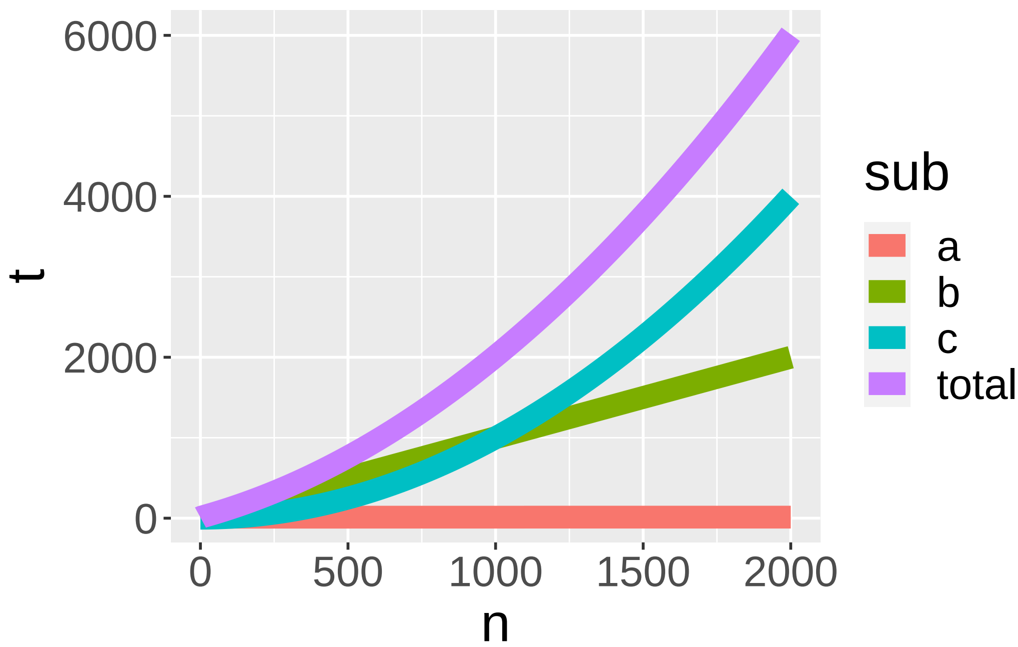
6.7 Example

```
t <- create_big_o_example(n = seq(0, 500))
ggplot2::ggplot(t, ggplot2::aes(x = n, y = t, color = sub)) +
  ggplot2::geom_line(size = 4) +
  ggplot2::theme(text = ggplot2::element_text(size = 20))
```



6.8 Example

```
t <- create_big_o_example(n = seq(0, 2000))
ggplot2::ggplot(t, ggplot2::aes(x = n, y = t, color = sub)) +
  ggplot2::geom_line(size = 4) +
  ggplot2::theme(text = ggplot2::element_text(size = 20))
```



No, work on C instead

6.9 Discussion

Big-O helps to:

- find algorithm to profile
- make predictions

Agree yes/no

6.10 Exercise 1 [SKIP]

- Measure big-O complexity of https://www.pythonpool.com/check-if-number-is-prime-in-python/

```
def isprime(num):
  for n in range(
    2, int(num**0.5)+1
  ):
```

```
    if num%n==0:
      return False
  return True
```

```
def isprime(num):
    if num> 1:
        for n in range(2,num):
            if (num % n) == 0:
                return False
        return True
    else:
        return False
```

6.11 Exercise 1 [SKIP]

- Measure big-O complexity of https://www.pythonpool.com/check-if-number-is-prime-in-python/

```
def isprime(num):
  for n in range(
    2, int(num**0.5)+1
  ):
    if num%n==0:
      return False
  return True
```

```
def Prime(no, i = 2):
    if no == i:
        return True
    elif no % i == 0:
        return False
    return Prime(no, i + 1)
```

6.12 Exercise 2 [SKIP]

- Measure big-O complexity of DNA alignment at https://johnlekberg.com/blog/2020-10-25-seq-align.html

```
ACGTACGTACGTACGTACGTACGT
ACGTACGTACGTCGTACGTACGT

ACGTACGTACGTACGTACGTACGT
ACGTACGTACGT-CGTACGTACGT
```

7 Measurement 2: Run-time speed profile

- See which code is spent most time in
- Use an input of suitable complexity
  - Note to self: next example should take at least 10 seconds!
- Consider using CI to obtain a speed profile every push!

7.1 Run-time speed profile: code

- [ ] Show R code in repo
- [ ] Run R code from RStudio
- [ ] Show Python code in repo
- [ ] Run Python code from command line

7.2 Myth 1

```
def slow_tmp_swap(x, y):
    tmp = x
    x = y
    y = tmp
    return x, y

def superfast_xor_swap(x, y):
    x ^= y
    y ^= x
    x ^= y
    return x, y
```

. . .

- C++ Core Guidelines: Per.4: Don't assume that complicated code is necessarily faster than simple code
- C++ Core Guidelines: Per.5: Don't assume that low-level code is necessarily faster than high-level code

7.3 Exercise 1 [30 mins]

Create speed profile of any function you like.

- [ ] Remind Python and R code on learner's repo

7.4 Exercise 2 [SKIP]

Create speed profile of https://www.pythonpool.com/check-if-number-is-prime-in-python/

7.5 Exercise 3 [SKIP]

Create speed profile of DNA alignment

8 Step 3: Think

- How to achieve the same with less calculations?
  - Aim to change big-O, not some micro-optimization
  - For example, store earlier results in a sorted look-up table

  Feynman Problem Solving Algorithm:

  1. Write down the problem.
  2. Think very hard.
  3. Write down the answer

9 Step 4: Measure again

In TDD, this test would have been present already:

```
assert 10.0 * get_t_runtime_b() < get_t_runtime_a()
```

Adapt the constant to reality.

- C++ Core Guidelines: Per.6: Don't make claims about performance without measurements

9.1 Recap quote

> It is far, far easier to make a correct program fast, than it is to make a fast program correct.
>
> Herb Sutter

Figure 3: Herb Sutter

Source Wikimedia

9.2 Case study

- ☐ Show ProjectRampal

9.3 Discussion

- Be critical on speed optimization solutions
- Tested and clean code always comes first
- Measure correctly, at the right complexity, before and after
- Prefer changing big-O over micro-optimizations (but see first point!)

Agree yes/no?

9.4 The End

9.5 Links

- Lecture of 2022: here:

1. Newport C. Deep work: Rules for focused success in a distracted world. Hachette UK; 2016.
2. Prechelt L. An empirical comparison of c, c++, java, perl, python, rexx and tcl. IEEE Computer. 2000;33(10):23–9.
3. Rodgers DP. Improvements in multiprocessor system design. ACM SIGARCH Computer Architecture News. 1985;13(3):225–31.
4. Sutter H, Alexandrescu A. C++ coding standards: 101 rules, guidelines, and best practices. Pearson Education; 2004.
5. Chellappa S, Franchetti F, Püschel M. How to write fast numerical code: A small introduction. Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007 Revised Papers. 2008;196–259.
6. Bartz-Beielstein T, Doerr C, Berg D van den, Bossek J, Chandrasekaran S, Eftimov T, et al. Benchmarking in optimization: Best practice and open issues. arXiv preprint arXiv:200703488. 2020;