Optimization

Optimization and Benchmarking

- **Optimization**: Modifying some aspect of your software's code to make it work more efficiently---so that it runs in less time and/or with less memory storage.

- **Benchmarking**: Performance comparison of a piece of software relative to other similar software. Comparison is usually evaluated based on the run time and memory usage.

"Premature optimization is the root of all evil."

–Sir Tony Hoare

BUT…

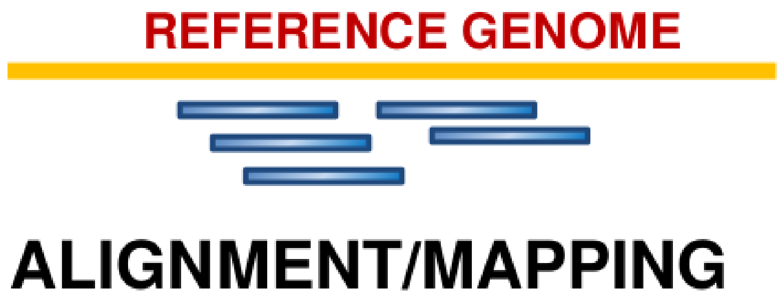
The full quote: *"We should forget about **small efficiencies**, say, 97% of the time---premature optimization is the root of all evil."*

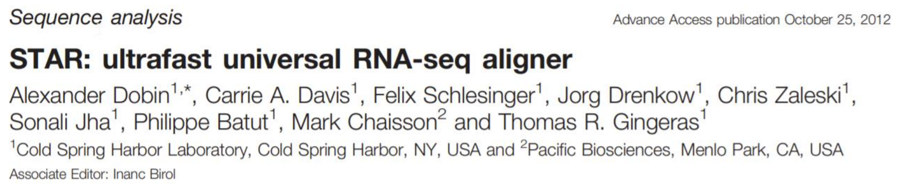
In other words…

"It's usually not worth spending a lot of time micro-optimizing code before it's obvious where the performance bottlenecks are. But, conversely, when designing software at a system level, performance issues should always be considered from the beginning. A good software developer will do this automatically, having developed a feel for where performance issues will cause problems. An inexperienced developer will not bother, misguidedly believing that a bit of fine tuning at a later stage will fix any problems."

Computing is a balance between…

A bioinformatics example: **sequence aligners**





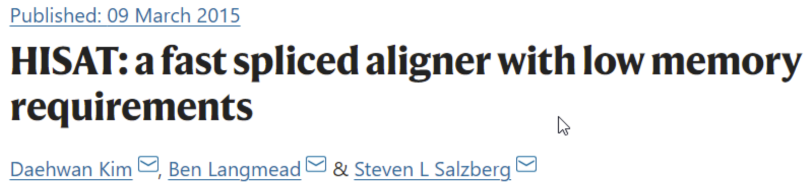
Sequence analysis Advance Access publication October 25, 2012

STAR: ultrafast universal RNA-seq aligner

Alexander Dobin[1,*], Carrie A. Davis[1], Felix Schlesinger[1], Jorg Drenkow[1], Chris Zaleski[1], Sonali Jha[1], Philippe Batut[1], Mark Chaisson[2] and Thomas R. Gingeras[1]
[1]Cold Spring Harbor Laboratory, Cold Spring Harbor, NY, USA and [2]Pacific Biosciences, Menlo Park, CA, USA
Associate Editor: Inanc Birol

Runtime: ~ 1 hour
Memory: ~ 100 gb

VS



Published: 09 March 2015

HISAT: a fast spliced aligner with low memory requirements

Daehwan Kim, Ben Langmead & Steven L Salzberg

Runtime: ~ 10 hours
Memory: ~ 10 gb

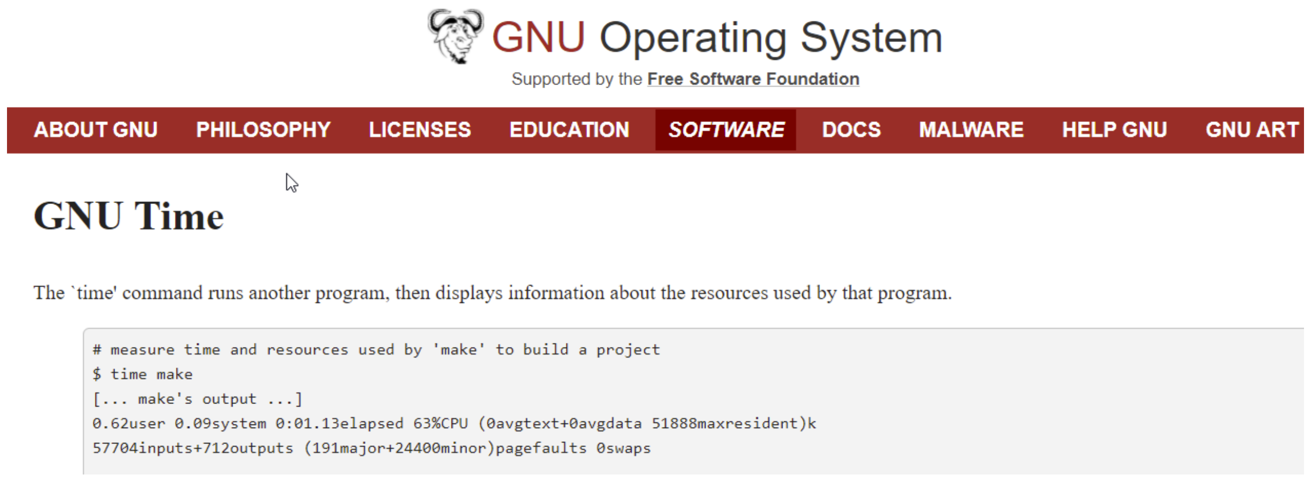
Two steps to learning how to optimize…

1. Learn how to use the available tools to evaluate the resources consumed by your code.

2. Use those tools to evaluate different ways of writing your code to learn what data types and algorithms are more efficient for each application.

Tools for evaluating runtime and memory usage

- At the system level: **GNUtime**
  - https://www.gnu.org/software/time/

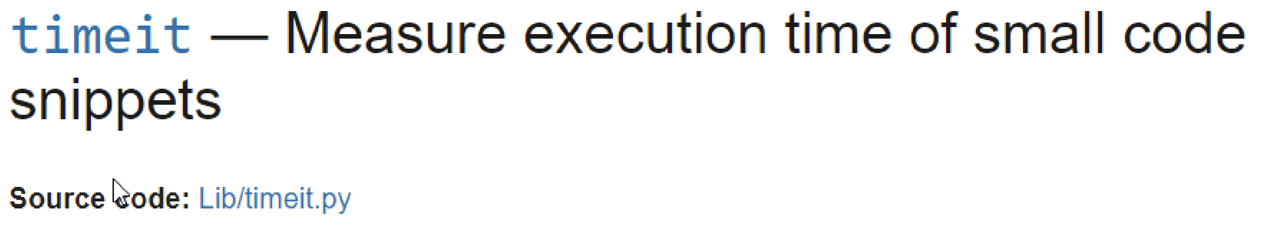


GNU Time

The `time' command runs another program, then displays information about the resources used by that program.

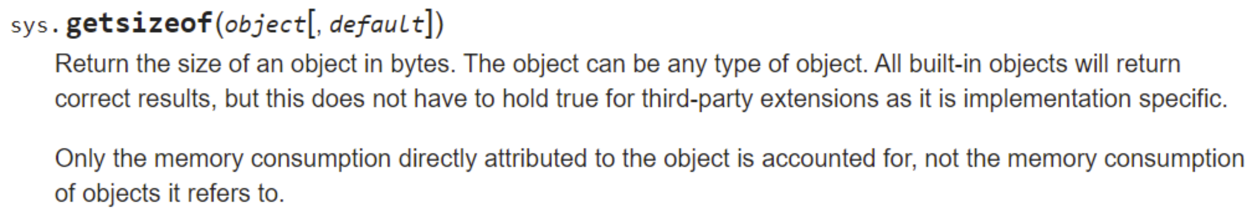
```
# measure time and resources used by 'make' to build a project
$ time make
[... make's output ...]
0.62user 0.09system 0:01.13elapsed 63%CPU (0avgtext+0avgdata 51888maxresident)k
57704inputs+712outputs (191major+24400minor)pagefaults 0swaps
```

- Within Python: **`timeit` and `getsizeof`**
  - https://docs.python.org/3/library/timeit.html
  - https://docs.python.org/3/library/sys.html#sys.getsizeof



`timeit` — Measure execution time of small code snippets

Source code: Lib/timeit.py



sys.**getsizeof**(*object*[, *default*])

Return the size of an object in bytes. The object can be any type of object. All built-in objects will return correct results, but this does not have to hold true for third-party extensions as it is implementation specific.

Only the memory consumption directly attributed to the object is accounted for, not the memory consumption of objects it refers to.