Software Performance Engineering

Recitation 1.4

Sophia Sun
Tuesday, Sep 23, 2025

Some due dates

- Homework 3 is released, due Sep 29, next Monday.

- Project 1 beta is due today!
- Submit your code to the GitHub repo
- Submit your writeup to **both Gradescope and your repo**

- Reach tier 20 to get at least a B grade for beta.

Beta Writeup

- A brief overview of your design.
- The general state of completeness and expected performance of your implementation.
- Any additional information that you feel would be helpful to the staff in understanding your submission.
- An acknowledgment of any help received from course staff, classmates on Piazza, or publicly available materials.

Weekly Report

- Please remember to submit your weekly report every week.
- Class contribution: 5 points towards final

- Time adjustment (starting this Friday):
  - Release: Friday morning, 10am
  - Due: next Monday evening, 5pm

Optimization tips

- What we covered in recitations so far:
  - recitation 1.1: from bit rotation to block rotation
  - recitation 1.2: in-block rotation (row-column-row algorithm)
  - recitation 1.3: divide-and-conquer approach for RCR
- Bit hacks
- Bently rules:
  - Caching
  - Precomputation
  - Sub-expression elimination
  - Loop unrolling
  - Function inlining

And so many more!

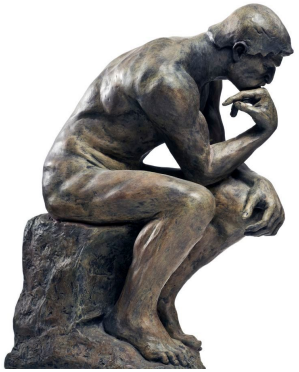
Resources

- Go to the course calendar to find all the lecture and recitation slides, and the project due dates.

Performance Engineering

Performance Engineering

think,



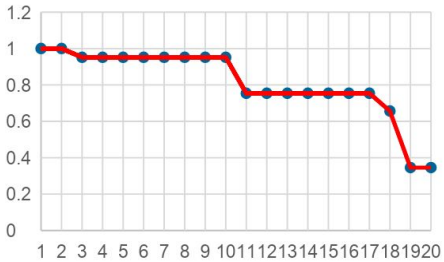
code,

run,

measure,

Basic Performance-Engineering Workflow

1. Measure the performance of Program `A`.
2. Make a change to Program `A` to produce a hopefully faster Program `A′`.
3. Measure the performance of Program `A′`.
4. If `A′` beats `A`, set `A = A′`.
5. If `A` is still not fast enough, go to Step 2.



If you can't measure performance **reliably**,
it is hard to make many small changes that add up

Example: Timing a Code for Sorting

```
#include <stdio.h>
#include <time.h>

void my_sort(double *A, int n);
void fill(double *A, int n);

int main() {
  int max = 4 * 1000 * 1000;
  int min = 1;
  int step = 20 * 1000;
  double A[max];
  struct timespec start, end;

  for (int n=min; n<max; n+=step) {
    fill(A, n);

    clock_gettime(CLOCK_MONOTONIC, &start
    my_sort(A, n);
    clock_gettime(CLOCK_MONOTONIC, &end);

    double tdiff = (end.tv_sec -
start.tv_sec)
        + 1e-9*(end.tv_nsec -
start.tv_nsec);
    printf("size %d, time %f\n", n, tdiff);
  }
  return 0;
}
```

Library for `clock_gettime()`

Sorting routine to be timed.

Auxiliary routine for filling array with random numbers.

Used by `clock_gettime()`:

```
struct timespec {
  time_t tv_sec; /* seconds */
  long tv_nsec;  /* nanoseconds */
};
```

Inspired by a study due to Sivan Toledo.

Example: Timing a Code for Sorting

```
#include <stdio.h>
#include <time.h>

void my_sort(double *A, int n);
void fill(double *A, int n);

int main() {
  int max = 4 * 1000 * 1000;
  int min = 1;
  int step = 20 * 1000;
  double A[max];
  struct timespec start, end;

  for (int n=min; n<max; n+=step) {
    fill(A, n);

    clock_gettime(CLOCK_MONOTONIC, &start);
    my_sort(A, n);
    clock_gettime(CLOCK_MONOTONIC, &end);

    double tdiff = (end.tv_sec -
start.tv_sec)
        + 1e-9*(end.tv_nsec -
start.tv_nsec);
    printf("size %d, time %f\n", n, tdiff);
  }
  return 0;
}
```

Loop over arrays of increasing length.

Array randomly filled.

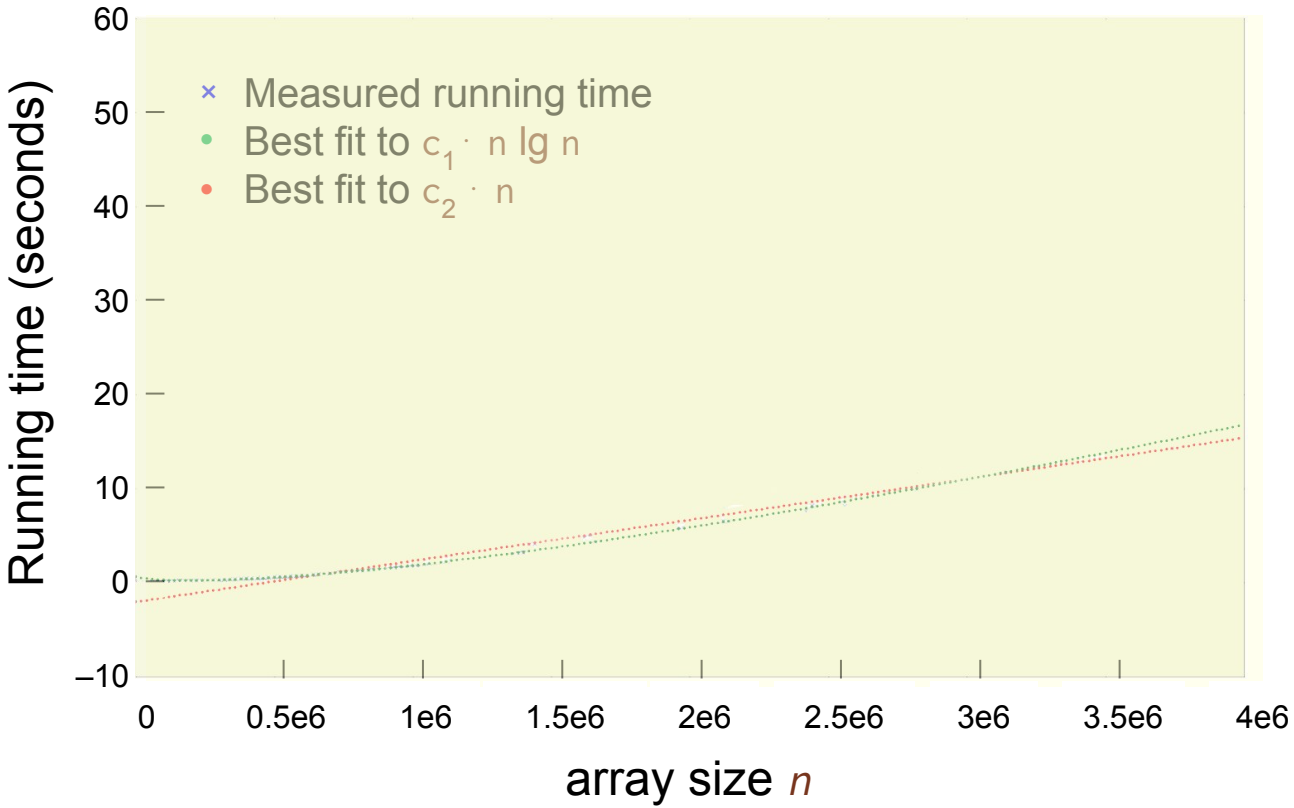
Measure time before sorting.

Sort.

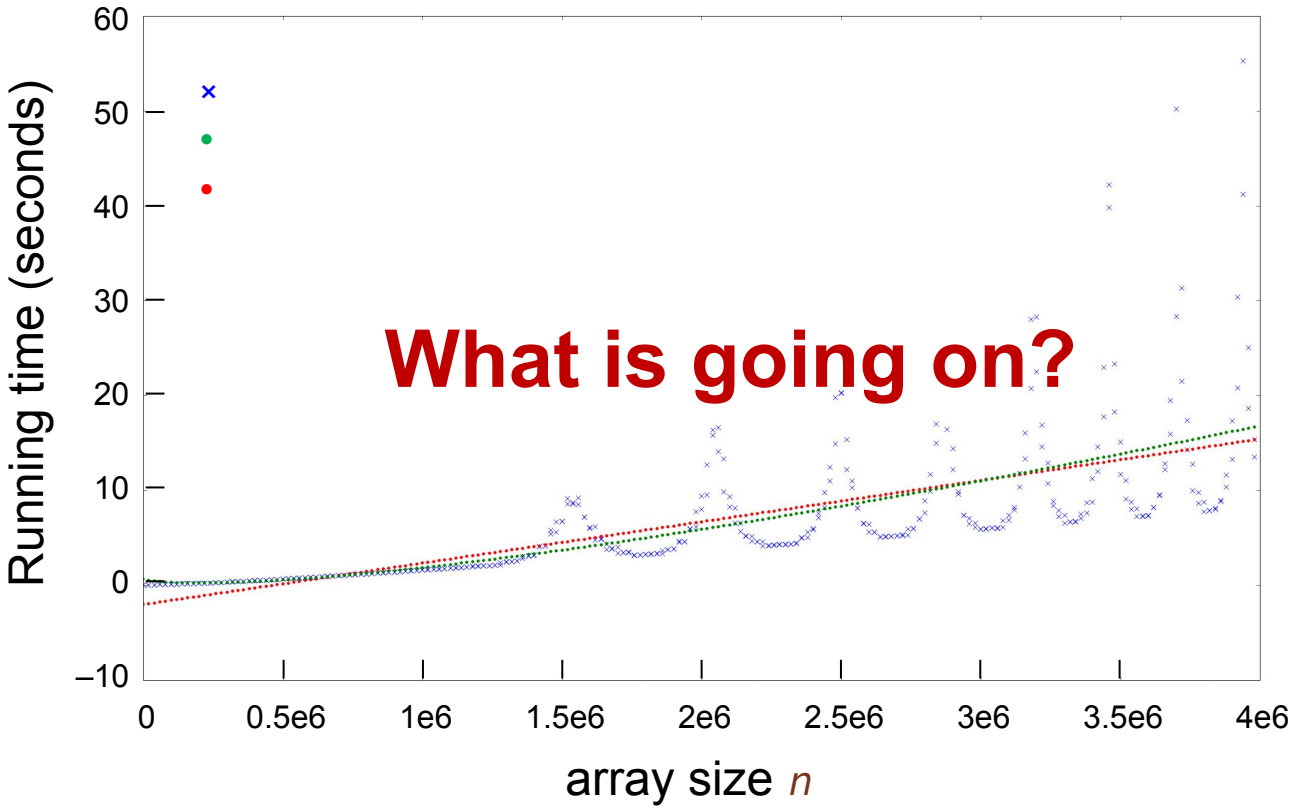
Measure time after sorting.

Compute elapsed time.

Running Times for Sorting



Running Times for Sorting



Dynamic Voltage and Frequency Scaling

DVFS is a technique to dynamically trade power for performance by **adjusting** the **clock frequency** and **supply voltage** to transistors

- Reduce operating frequency if chip is too hot or otherwise to save power.
- Reduce voltage if frequency is reduced.
- Turbo Boost increases frequency if the chip is cool.

$$\text{Power} \propto CV^2f$$

C = dynamic capacitance ≈ roughly area × activity (how many bits toggle)
V = supply voltage
f = clock frequency

Changing frequency and voltage has a cubic effect on power (and heat)

Today's Topic

How can one **reliably** measure the performance of software?

We'll start with **What Statistics and Metrics To Measure**

What Statistics and Metrics To Measure

Summary Statistics and Noise

Suppose that you measure the performance of a **deterministic program 100 times*** with the same input on a computer with some **interfering background noise**.

What statistic best represents the raw performance of the software?

- ❑ mean
- ❑ median
- ❑ maximum
- ❑ minimum

* we start it cold 100 times to eliminate any kind of caching effect from previous runs

Summary Statistics and Noise

Suppose that you measure the performance of a **deterministic program 100 times*** with the same input on a computer with some **interfering background noise**.

What statistic best represents the raw performance of the software?

- ❑ mean
- ❑ median
- ❑ maximum
- ☑ minimum

Minimum does the best at **noise rejection**, because we expect that any measurements higher than the minimum are due to noise.

Summarizing Ratios

Trial	Program A	Program A’
1	9	3
2	8	2
3	2	20
4	10	2
Mean	7.25	6.75

Summarizing Ratios

Trial	Program A	Program A'	A/A'
1	9	3	3.00
2	8	2	4.00
3	2	20	0.10
4	10	2	5.00
Mean	7.25	6.75	3.03

Conclusion

Program A' is > 3 times better than A

WRONG!

Turn the Comparison Upside-Down

Trial	Program A	Program A’	A/A’	A’/A
1	9	3	3.00	0.33
2	8	2	4.00	0.25
3	2	20	0.10	10.00
4	10	2	5.00	0.20
Mean	7.25	6.75	3.03	2.70

Paradox

A’ is 3.03x faster than A **&** A is 2.70x faster than A’

Observation

The arithmetic mean of A/A’ is **NOT** the inverse of the arithmetic mean of A’/A

Geometric Mean

Trial	Program A	Program A'	A/A'	A'/A
1	9	3	3.00	0.33
2	8	2	4.00	0.25
3	2	20	0.10	10.00
4	10	2	5.00	0.20
Mean	(a) 7.25	(a) 6.75	(g) 1.57	(g) 0.64

Formula

$$\left(\prod_{i=1}^{n} a_i\right)^{1/n} = \sqrt[n]{a_1 a_2 \cdots a_n}$$

Observation

The geometric mean of A/A' **is** the inverse of the geometric mean of A'/A

Selecting among Summary Statistics

Given server, service as many requests as possible

- Arithmetic mean
- CPU utilization

Most cloud service requests are satisfied within 100 ms

- 90th percentile
- Wall-clock time

Best game-playing AI

- Arithmetic mean
- Win rate

Fit into a machine with 100 MB of memory

- Maximum
- Memory use

Support frequent use on a mobile device

- Arithmetic mean
- Energy use or CPU utilization

Most environmentally friendly

- Arithmetic mean
- Carbon footprint

Meet a customer service-level agreement (SLA)

- Weighted combo of statistics
- Multiple metrics

Coding time before beta due!

Optimization tips

- Always take a shot if you have an idea!
- Decide for yourself: what works for others may not work for you (even an idea from a course staff!)
  - a course staff's suggestion is also not the only solution.
- It's not guaranteed that an optimization idea will actually make your code run faster.
  - if that happens, think about why.
- Take one step at a time to make debugging easier.
- Write comments, so you won't forget what you're doing or planning to do a week later.