Aalto University

# Competitive Programming 2020

## 4 What can you do in 1 second?

# Today's program

- **12:15:** Lecture (Zoom)
- **13:00:** Practice contest (CSES)
  - *new: work in teams with 2–3 people*
  - create a CSES team account,
    all team members log on to CSES using it
- **16:00:** Post-contest wrap-up (Zoom)

# What is fast enough?

- **Understand orders of magnitude!**
- How many operations can you do in 1 second?
- How many operations is your program going to do with the largest inputs?
- How much space would it use?

# 1 second ≈ 2 billion cycles

< 1 clock cycle: ***independent*** elementary arithmetic operations

- **add, subtract, multiply**, and, or, xor, shift; int, long long, float, double ...

< 1 clock cycle: getting data from memory ***if easy to predict***

- **linear reading**

> 1 clock cycle: ***dependent*** elementary arithmetic operations

- a += x1; a += x2; a += x3; a += x4; ...

> 10 clock cycles: ***non-elementary*** arithmetic operations

- **division**, square root, trigonometry

> 100 clock cycles: getting data from memory ***if hard to predict***

- **following linked list**

| $n$ = | 1M | 2M | 4M | 10M | 20M | 40M | 100M | 200M | 400M | 1G | 2G |
|---|---|---|---|---|---|---|---|---|---|---|---|
| basic arithmetic: + - * \| & ^ ~ | | | | | | | | | | | x |
| linear reading | | | | | | | | | | | x |
| tiny array updates (1K) | | | | | | | | | | x | |
| division: / % | | | | | | | | | x | | |
| small array updates (1M) | | | | | | | | x | | | |
| random number generator | | | | | | | | x | | | |
| std::vector<bool> updates | | | | | | | x | | | | |
| large array updates | | | | | | x | | | | | |
| follow linked list | | | | x | | | | | | | |
| std::sort | | | | x | | | | | | | |
| std::priority_queue<int> | | | x | | | | | | | | |
| std::unordered_set<int> | | | x | | | | | | | | |
| std::unordered_map<int,int> | | x | | | | | | | | | |
| std::set<int> | x | | | | | | | | | | |
| std::map<int,int> | x | | | | | | | | | | |

≈ 1 second

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | | | |
| | | | | |
| | | | | |
| | | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | 2 000 | 7 000 | 20 000 |
| | | | | |
| | | | | |
| | | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | 2 000 | 7 000 | 20 000 |
| triple | $3 \cdot n^3$ | | | |
| | | | | |
| | | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | 2 000 | 7 000 | 20 000 |
| triple | $3 \cdot n^3$ | 150 | 300 | 700 |
| | | | | |
| | | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | 2 000 | 7 000 | 20 000 |
| triple | $3 \cdot n^3$ | 150 | 300 | 700 |
| subset | $n \cdot 2^n$ | | | |
| | | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | 2 000 | 7 000 | 20 000 |
| triple | $3 \cdot n^3$ | 150 | 300 | 700 |
| subset | $n \cdot 2^n$ | 19 | 22 | 25 |
| | | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | 2 000 | 7 000 | 20 000 |
| triple | $3 \cdot n^3$ | 150 | 300 | 700 |
| subset | $n \cdot 2^n$ | 19 | 22 | 25 |
| permutation | $n \cdot n!$ | | | |

# Algorithms and time

| Check each … | | 10M operations | 100M operations | 1G operations |
|---|---|---|---|---|
| element | $n$ | 10 000 000 | 100 000 000 | 1 000 000 000 |
| pair | $2 \cdot n^2$ | 2 000 | 7 000 | 20 000 |
| triple | $3 \cdot n^3$ | 150 | 300 | 700 |
| subset | $n \cdot 2^n$ | 19 | 22 | 25 |
| permutation | $n \cdot n!$ | 9 | 10 | 11 |

# Memory

- Example: storing a large graph in memory
  - $n$ = 1M nodes
  - $m$ = 10M edges
- **Adjacency matrix:** $n^2$ values
- **Adjacency list:** $m$ values

# Memory

- Example: storing a large graph in memory
  - $n$ = 1M nodes
  - $m$ = 10M edges
- **Adjacency matrix:** $n^2$ values > **1 TB**
- **Adjacency list:** $m$ values < **100 MB**

# I/O performance

**C++:** you can read 1M values in ≈ 0.1s

```
int main() {
    std::ios::sync_with_stdio(0);
    for (int i = 0; i < 1000000; ++i) {
        int x;
        std::cin >> x;
        ...
```

# I/O performance

**Python:** you can read 1M values in ≈ 0.4s

```
print(sum(int(x) for x in input().split()))
```

# Measure!

- Test & benchmark locally

```
time ./my-program < input > output
```

- Construct large inputs, see what happens
- Understand which part takes time (and why)