LEIDEN UNIVERSITY MEDICAL CENTER

Code optimisation

Jeroen F. J. Laros

Leiden Genome Technology Center

Department of Human Genetics

Center for Human and Clinical Genetics

LUMC

Introduction

Before we start

Readability comes first:

- First write *readable* and *maintainable* code.
- Make sure your code is *correct*.
- Only optimise when needed.

> Premature optimization is the root of all evil.
>
> Donald Knuth

Introduction

Before we start

Readability comes first:

- First write *readable* and *maintainable* code.
- Make sure your code is *correct*.
- Only optimise when needed.

However:

- Some things should *always* be considered.
- Inefficient code is usually not more readable.

> Premature optimization is the root of all evil.
>
> Donald Knuth

Introduction

Popular view

Buy faster computers, buy more computers.

However:

- Only basic speedup.
- Add a lot of complexity to your program.
- Expensive (€5,000 for one new node).

Introduction

Popular view

Buy faster computers, buy more computers.

However:

- Only basic speedup.
- Add a lot of complexity to your program.
- Expensive (€ 5,000 for one new node).

Related opinion: "*Don't worry about speed, everything will be solved in ten years (once computers become faster).*"

Complexity theory

Terminology

Study of the *time* and *memory* complexity of algorithms.

Correlation	Complexity
Constant	$\mathcal{O}(1)$
Logarithmic	$\mathcal{O}(\log n)$
Linear	$\mathcal{O}(n)$
Quadratic	$\mathcal{O}(n^2)$
Exponential	$\mathcal{O}(2^n)$

Table 1: Examples of different complexities.

What is the correlation between the size of the input (n) and the memory or CPU usage?

Complexity theory

Terminology

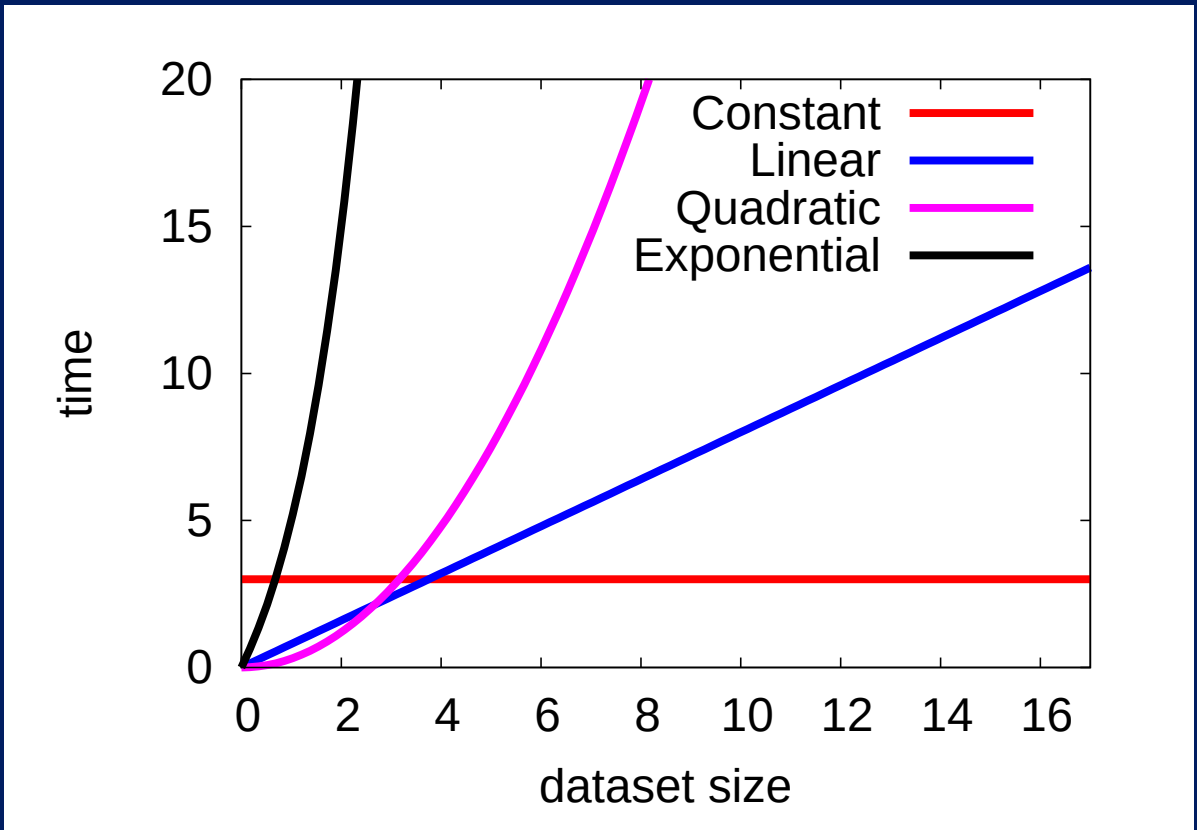

Figure 1: Different complexities.

Complexity theory

Examples

Constant complexity ($\mathcal{O}(1)$).

```
1  def first_letter(word):
2      return word[0]
```

Listing 1: Get the first letter of a word.

No dependence on the length of the word.

Complexity theory

Examples

Linear complexity ($\mathcal{O}(n)$).

```
1   def maximum_value(list_of_values):
2       maximum = 0
3
4       for value in list_of_values:
5           if value > maximum:
6               maximum = value
7
8       return maximum
```

Listing 2: Find the maximum value in a list.

If the list is twice as long, this routine will take twice as much time.

Complexity theory

Examples

Quadratic complexity ($\mathcal{O}(n^2)$).

```
1   def optimal_pair(f, list_of_values):
2       optimum = 0
3
4       for value_1 in list_of_values:
5           for value_2 in list_of_values:
6               if f(value_1, value_2) > optimum:
7                   optimum = f(value_1, value_2)
8
9       return optimum
```

Listing 3: Get the first letter of a word.

The amount of time grows faster than the length of the input.

Complexity theory

Examples

Algorithm	Complexity
Searching in a sorted list	$\mathcal{O}(\log n)$
Intersecting sorted regions	$\mathcal{O}(n)$
Sorting	$\mathcal{O}(n \log n)$
Pairwise alignment	$\mathcal{O}(n^2)$
De novo assembly	$\mathcal{O}(2^n)$

Table 2: Known complexities.

For a lot of algorithms, the complexity is known.

- If your implementation behaves differently, something is wrong.

Applications of complexity theory

Data structures and bottlenecks

Some things we should always try to do:

- Find an algorithm with minimal complexity.
- Choose the right *data structures*.

This is difficult, even for experienced programmers.

- Do not hesitate to ask.
- Discuss your solution (in detail).

These things will give you more speedup than parallelisation.

Applications of complexity theory

Complexity reduction

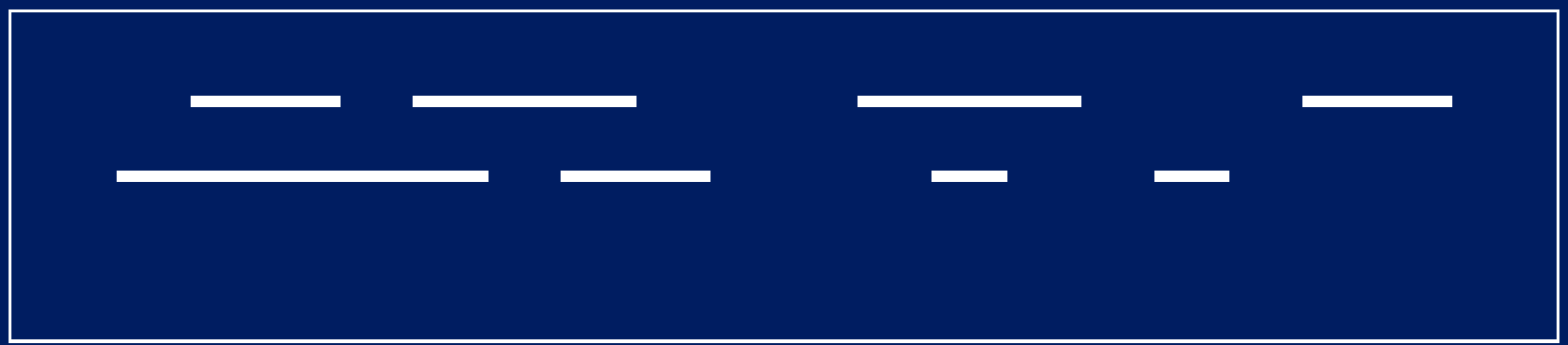

Figure 2: Intersection of two sets of regions.

Calculate the *overlap* of two lists of regions.

- Pops up in many different forms.

Applications of complexity theory

Complexity reduction

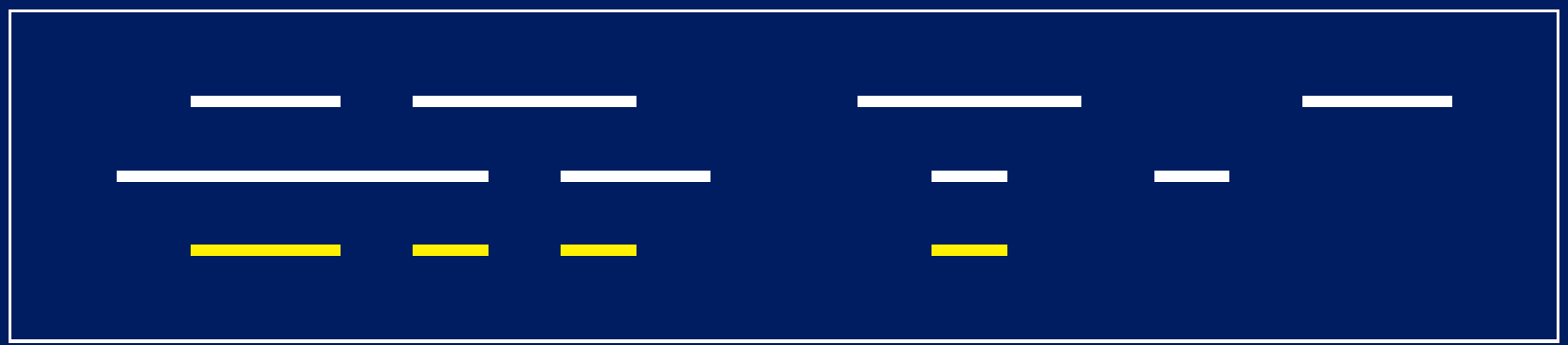

Figure 2: Intersection of two sets of regions.

Calculate the *overlap* of two lists of regions.

- Pops up in many different forms.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
$\rightarrow$	$(10, 20)$	$(5, 30)$	$\leftarrow$	
	$(25, 40)$	$(35, 45)$		
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
$\rightarrow$	$(10, 20)$	$(5, 30)$	$\leftarrow$	$(10, 20)$
	$(25, 40)$	$(35, 45)$		
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
$\rightarrow$	$(10, 20)$	$(5, 30)$		$(10, 20)$
	$(25, 40)$	$(35, 45)$	$\leftarrow$	
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
$\rightarrow$	$(10, 20)$	$(5, 30)$		$(10, 20)$
	$(25, 40)$	$(35, 45)$		
	$(55, 70)$	$(60, 65)$	$\leftarrow$	
	$(85, 95)$	$(75, 80)$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
$\rightarrow$	$(10, 20)$	$(5, 30)$		$(10, 20)$
	$(25, 40)$	$(35, 45)$		
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$	$\leftarrow$	

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
	$(10, 20)$	$(5, 30)$	$\leftarrow$	$(10, 20)$
$\rightarrow$	$(25, 40)$	$(35, 45)$		$(25, 30)$
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
$\rightarrow$	$(25, 40)$	$(35, 45)$	$\leftarrow$	$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$			
	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
$\rightarrow$	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$	$\leftarrow$		
	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
$\rightarrow$	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$			
	$(85, 95)$	$(75, 80)$	$\leftarrow$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$	$\leftarrow$	$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
$\rightarrow$	$(55, 70)$	$(60, 65)$			
	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$	$\leftarrow$	$(25, 30)$	$(35, 40)$
$\rightarrow$	$(55, 70)$	$(60, 65)$			
	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
$\rightarrow$	$(55, 70)$	$(60, 65)$	$\leftarrow$	$(60, 65)$	
	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
$\rightarrow$	$(55, 70)$	$(60, 65)$		$(60, 65)$	
	$(85, 95)$	$(75, 80)$	$\leftarrow$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$	$\leftarrow$	$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$		$(60, 65)$	
$\rightarrow$	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$	$\leftarrow$	$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$		$(60, 65)$	
$\rightarrow$	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$	$\leftarrow$	$(60, 65)$	
$\rightarrow$	$(85, 95)$	$(75, 80)$			

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$		$(60, 65)$	
$\rightarrow$	$(85, 95)$	$(75, 80)$	$\leftarrow$		

Table 3: Intersection algorithm of $\mathcal{O}(n^2)$.

Notice that interesting things happen only when the arrows are close.

Applications of complexity theory

Complexity reduction

Based on the previous observation:

- Remote regions will never give an overlap.

Algorithm that tries to “keep close”.

1. Consider the first regions in both lists.
2. See whether the two regions have an overlap.
3. Take the list which region has the smallest end coordinate and go to the next region.
4. Proceed to Rule 2.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
$\rightarrow$	$(10, 20)$	$(5, 30)$	$\leftarrow$	
	$(25, 40)$	$(35, 45)$		
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$		

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
$\rightarrow$	$(10, 20)$	$(5, 30)$	$\leftarrow$	$(10, 20)$
	$(25, 40)$	$(35, 45)$		
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$		

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap
	$(10, 20)$	$(5, 30)$	$\leftarrow$	$(10, 20)$
$\rightarrow$	$(25, 40)$	$(35, 45)$		$(25, 30)$
	$(55, 70)$	$(60, 65)$		
	$(85, 95)$	$(75, 80)$		

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
$\rightarrow$	$(25, 40)$	$(35, 45)$	$\leftarrow$	$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$			
	$(85, 95)$	$(75, 80)$			

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$	$\leftarrow$	$(25, 30)$	$(35, 40)$
$\rightarrow$	$(55, 70)$	$(60, 65)$			
	$(85, 95)$	$(75, 80)$			

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
$\rightarrow$	$(55, 70)$	$(60, 65)$	$\leftarrow$	$(60, 65)$	
	$(85, 95)$	$(75, 80)$			

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
$\rightarrow$	$(55, 70)$	$(60, 65)$		$(60, 65)$	
	$(85, 95)$	$(75, 80)$	$\leftarrow$		

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$		$(60, 65)$	
$\rightarrow$	$(85, 95)$	$(75, 80)$	$\leftarrow$		

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

Applications of complexity theory

Complexity reduction

	Region 1	Region 2		Overlap	
	$(10, 20)$	$(5, 30)$		$(10, 20)$	
	$(25, 40)$	$(35, 45)$		$(25, 30)$	$(35, 40)$
	$(55, 70)$	$(60, 65)$		$(60, 65)$	
$\rightarrow$	$(85, 95)$	$(75, 80)$	$\leftarrow$		

Table 4: Intersection algorithm of $\mathcal{O}(n)$.

This optimisation will outweigh parallelisation.

- One computer running this algorithm will outperform a cluster running the other algorithm.

Applications of complexity theory

Data structures

Closely related to previous discussion.

Use the simplest data structure that suits your needs.

- Complicated data structures have more overhead.

Note that simplest is not always the *easiest*.

This can give (way) more speedup than parallelisation.

Applications of complexity theory

Data structures

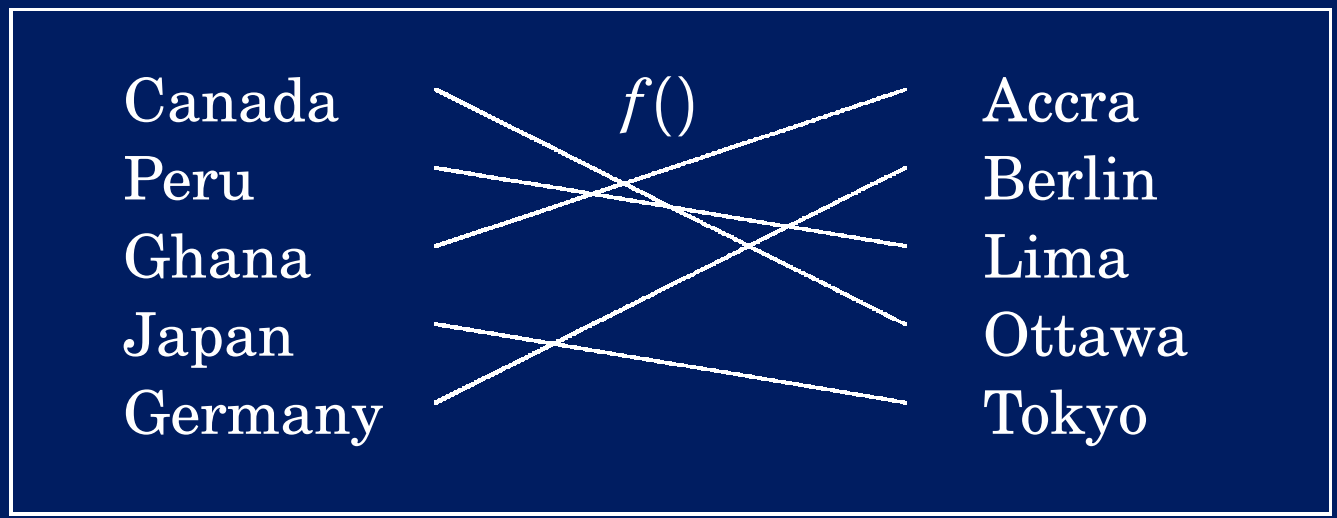

Figure 3: Hashing function.

A *dictionary* or *hash*.

- Very fast data structure for *key-value pairs*.

Applications of complexity theory

Data structures

4	5	8	1	0	7	2	3

Figure 4: Direct memory indexing.

But if you have keys ranging from 0 to n, use a simple *list* or *array*.

Linear speedup.

Applications of complexity theory

Data structures

Using a *set* data structure.

```
1   for word in text:
2     for gene in genes:
3       if word == gene:
4         print word
```

Listing 4: Find gene names in some text.

```
1   print set(text) & set(genes)
```

Listing 5: Find gene names in some text using sets.

Quadratic speedup.

Applications of complexity theory

Bottlenecks

The 90/10 law: 90% *of the execution time of a computer program is spent executing* 10% *of the code.*

Finding *bottlenecks* in your code (focus on the 10%).

You might have a good idea already.

- The most complicated part of your program.
- The most used part of your program.

But often it is hard to find.

Applications of complexity theory

Profilers

Use a *profiler* to see which part takes up the most time.

```
1        ncalls  tottime  cumtime  filename:line(function)
2             1    0.002   35.276  tssv:3(<module>)
3             1    0.000   34.776  tssv.py:437(main)
4             1    0.058   34.767  tssv.py:336(tssv)
5          7272    0.185   34.599  tssv.py:121(alignPair)
6         14544    0.100   34.408  sg_align.py:71(align)
7         14544   25.350   33.433  sg_align.py:22(_align)
8      11092934    7.192    7.192  {min}
9        895601    0.958    0.958  {range}
```

Listing 6: Profiling output.

The culprit in this example is the function “_align”.

Combining languages

Compiled languages

Rewrite bottleneck in a *compiled language*.

- Still readable.
- Linear speedup.
- Requires compilation of your code.

When I find my code in tons of trouble,
Friend and colleagues come to me,
Speaking words of wisdom:
"Write in C."

Anonymous

Combining languages

Example: TSSV

```
1   def _align(matrix, xSize, ySize, seq1, seq2):
2       for x in range(1, xSize):
3           for y in range(1, ySize):
4               matrix[x][y] = min(
5                   matrix[x -1][y] + 1,
6                   matrix[x][y - 1] + 1,
7                   matrix[x - 1][y - 1] +
8                       int(seq1[x - 1] != seq2[y - 1]))
```

Listing 7: Semi-global alignment in Python.

The complexity is already minimal.

Combining languages

Example: TSSV

```
1   void _align(int **matrix, int x_size, int y_size,
2       char *seq1, char *seq2) {
3     int x,
4         y;
5
6     for (x = 1; x < x_size; x++)
7       for (y = 1; y < y_size; y++)
8         matrix[x][y] = _min(
9           _min(
10            matrix[x - 1][y] + 1,
11            matrix[x][y - 1] + 1),
12          matrix[x - 1][y - 1] +
13            (int)(seq1[x - 1] != seq2[y - 1]));
14  }
```

Listing 8: Semi-global alignment in C.

Combining languages

Example: TSSV

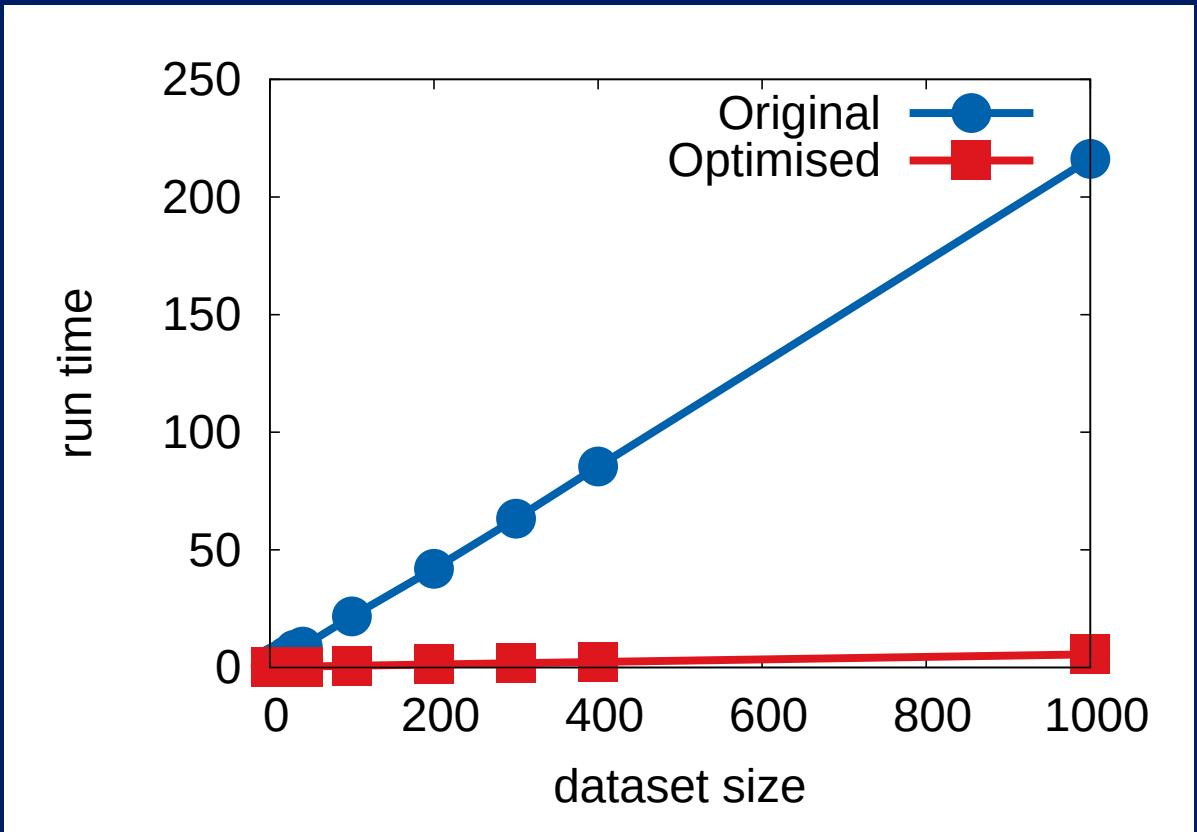

Figure 5: Run times for original and optimised versions.

Combining languages

Example: TSSV

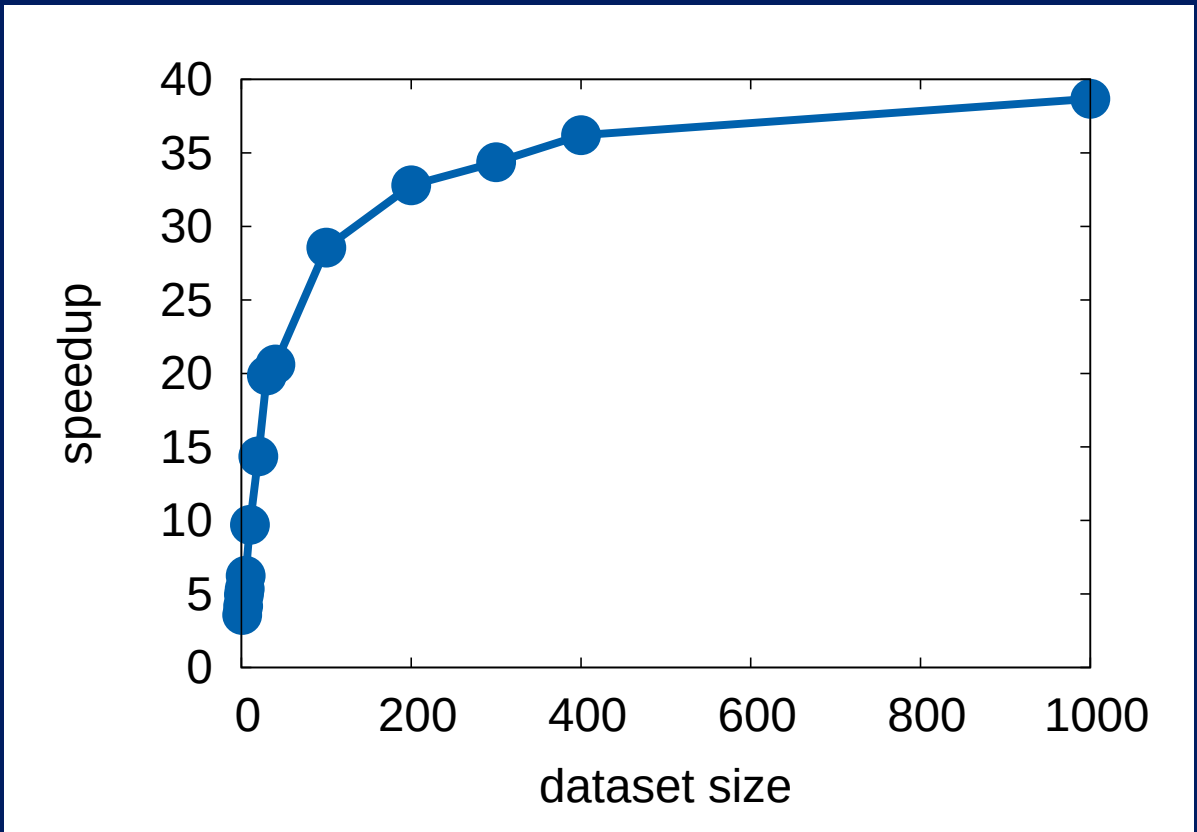

Figure 6: Speedup.

Conclusions

Think before you buy

Design outperforms brute force.

- Analyse the bottlenecks.
- Do not hesitate to ask for help.

Small investments can matter a lot.

- Choosing correct data structures.
- Rewriting in an other language.

Questions?

Acknowledgements:

Johan den Dunnen