High-level Optimization

David Grellscheid

UNIVERSITETET I BERGEN

Typical scientific workflow

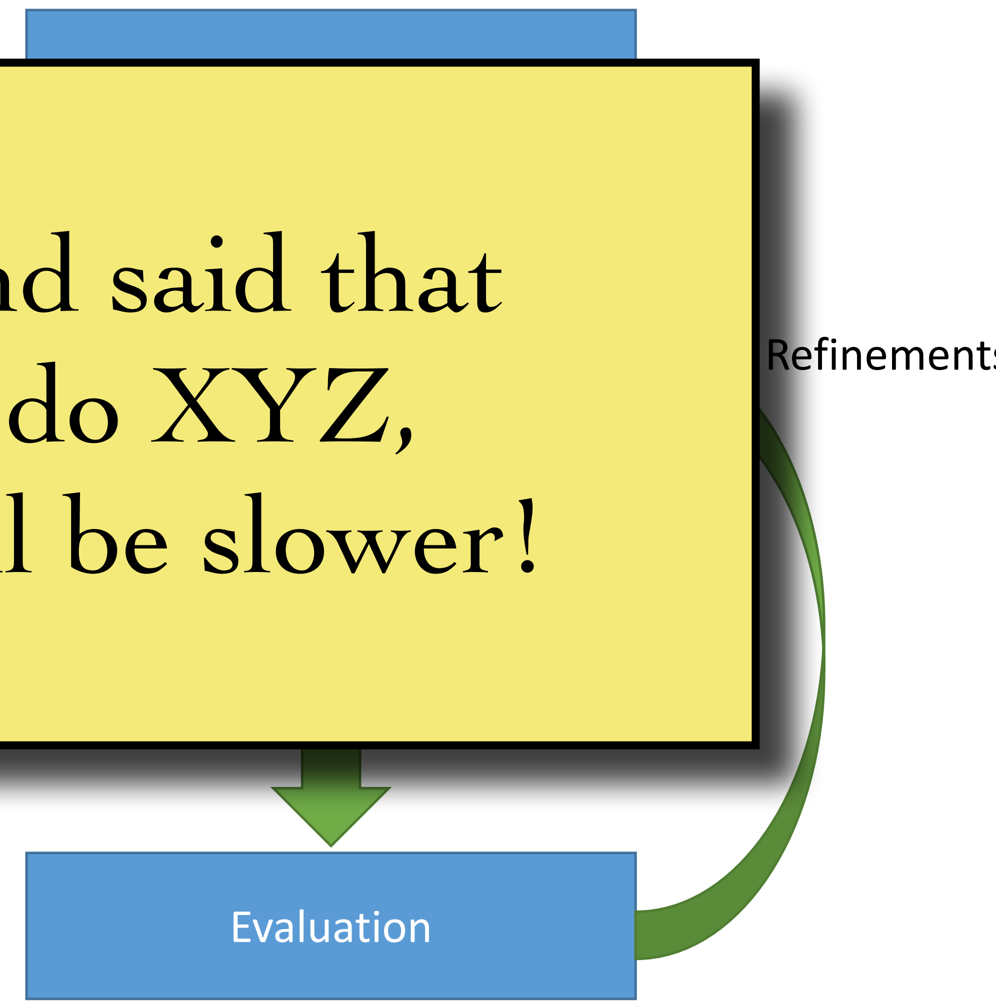
Correctness is main concern

S

Fi

like it works is kept

A friend of my friend said that you should never do XYZ, because the code will be slower!



Sub-optimal choices only noticed later on
(if at all)

Donald Knuth, December 1974:

> Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. 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%.

"Structured Programming with go to Statements", Computing Surveys, Vol 6, No 4.

Runtime is not the only factor to consider,
need to think about trade off between time spent in:

development
debugging
validation
portability
runtime in your own usage
other developers' time (now/future)
total runtime for all users

CPU time much cheaper than human time!

Reusability is an efficiency!

If the student after you has to start from zero,
all your work is wasted

Optimization points

Someone else already solved (part of) the problem:

LAPACK, BLAS
GNU scientific library
C++ Boost
Numpy, Scipy, Pandas
…

Develop googling skills, evaluate what exists.
Quality often **much** better than self-written attempts

Optimization points

Choice of programming language

Be aware of what exists

Know strengths / weaknesses

But: needs to fit rest of project

take a look at Haskell, Erlang, Prolog
to get an idea how different the approaches can be

Optimization points

```
findLongestUpTo :: Int -> (Int,Int)
findLongestUpTo mx = maximum ( map f [1 .. mx] )
  where f x = (collatzLength x, x)


collatzLength :: Int -> Int
collatzLength 1 = 1
collatzLength n = 1 + collatzLength (collatzStep n)


collatzStep :: Int -> Int
collatzStep n
  | even n    = n `div` 2
  | otherwise = 3 * n + 1
```

Optimization points

Program design

First version: understand the problems

now start again!

Second version: you know what you're doing

refactor / clean up / make reusable

Done :-)

From Python, I want to...

harder

use output from other progs

run other programs

call other functions

Use output from other progs

data files in **standard** formats

ideally plain text (txt, csv, json, xml, ...)

or self-documented binary formats (netCDF,...)

watch out for encoding issues!

Use output from other progs

Core Python: `with open(...):`

Standard libraries: csv, json, xml, ...

External libs: xarray, ...

Run other programs

https://docs.python.org/3/library/subprocess.html

```
import subprocess
result = subprocess.run(
    ["ls", "-l"],
    capture_output=True,
    text=True
)
print(result.stdout)
```

Parallel tasks: multiprocessing, queue
https://docs.python.org/3/library/concurrency.html

Call other functions

Foreign function interface (FFI)

Python-C: ctypes, cython

Python-Fortran: f2py (in numpy)

Python-Java: jython, py4j, ...

Python-R: rpy2   /   R-Python: reticulate

https://rpy2.github.io/doc/v3.5.x/html/introduction.html

Optimization points

Algorithm / data structure choice

can get orders of magnitude in savings

Local and hardware-specific optimisations

- not in this course-

What are we optimizing?

Time

Memory

Disk

Electricity

Compile time

Ease of use

Ease of deployment

Ease of development

Complexity basics

Much simplified, skipping formal derivation

```
while not is_sorted(xs):
    random.shuffle(xs)
```

$O(N\,N!)$

Scaling behaviour with size N of problem set:

$O(1)$ - constant time independent of N

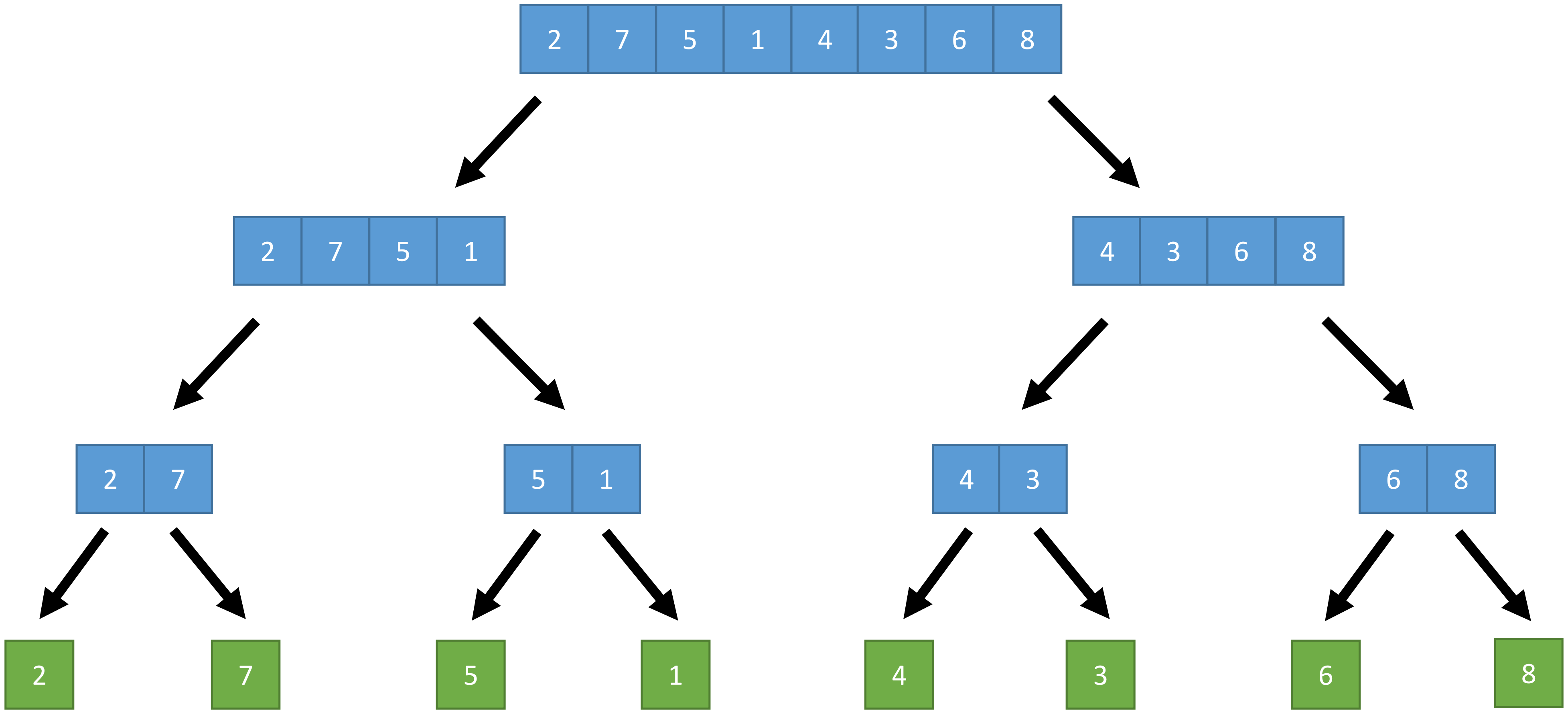
$O(N)$ - linear with N

$O(N^2)$ - quadratic in N

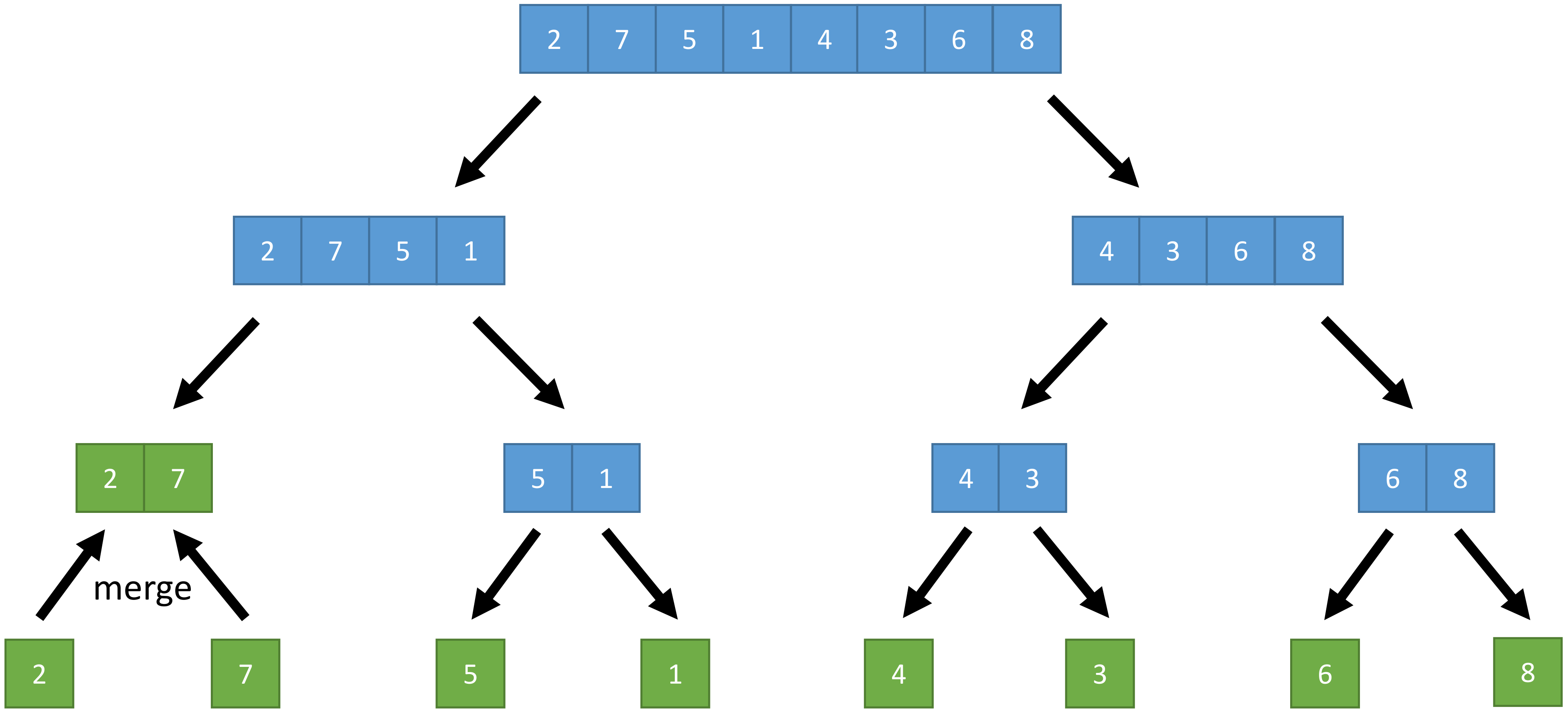
Merge Sort

2	7	5	1	4	3	6	8

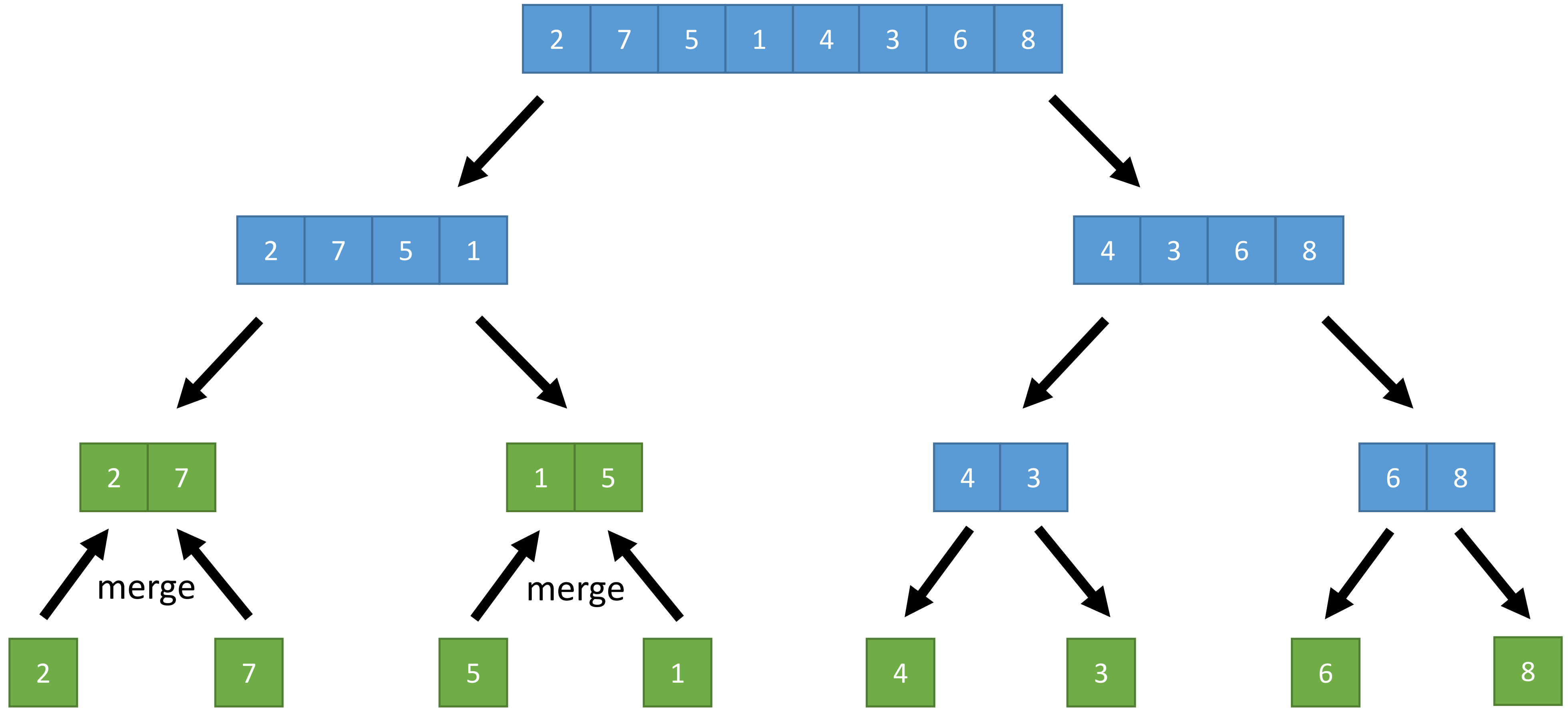
Merge Sort



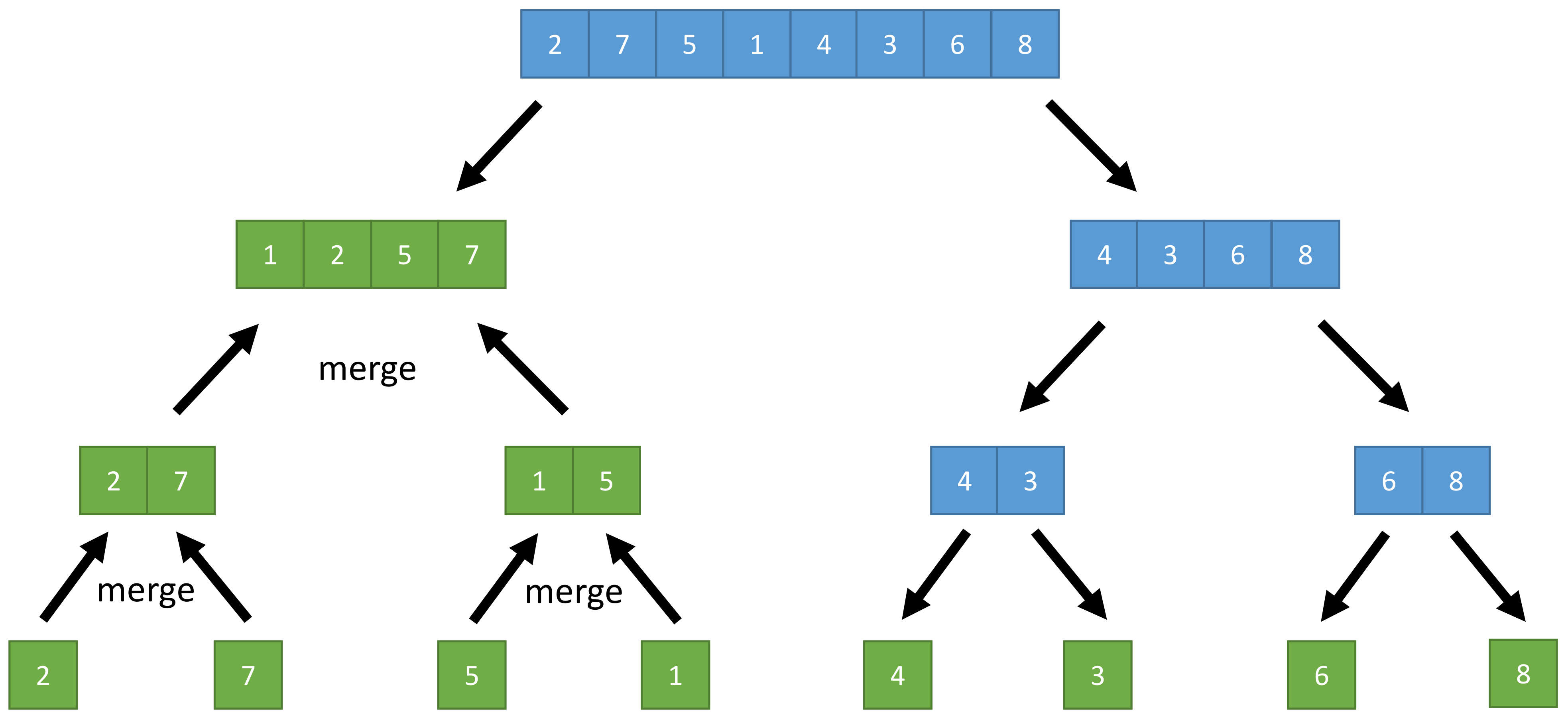
Merge Sort



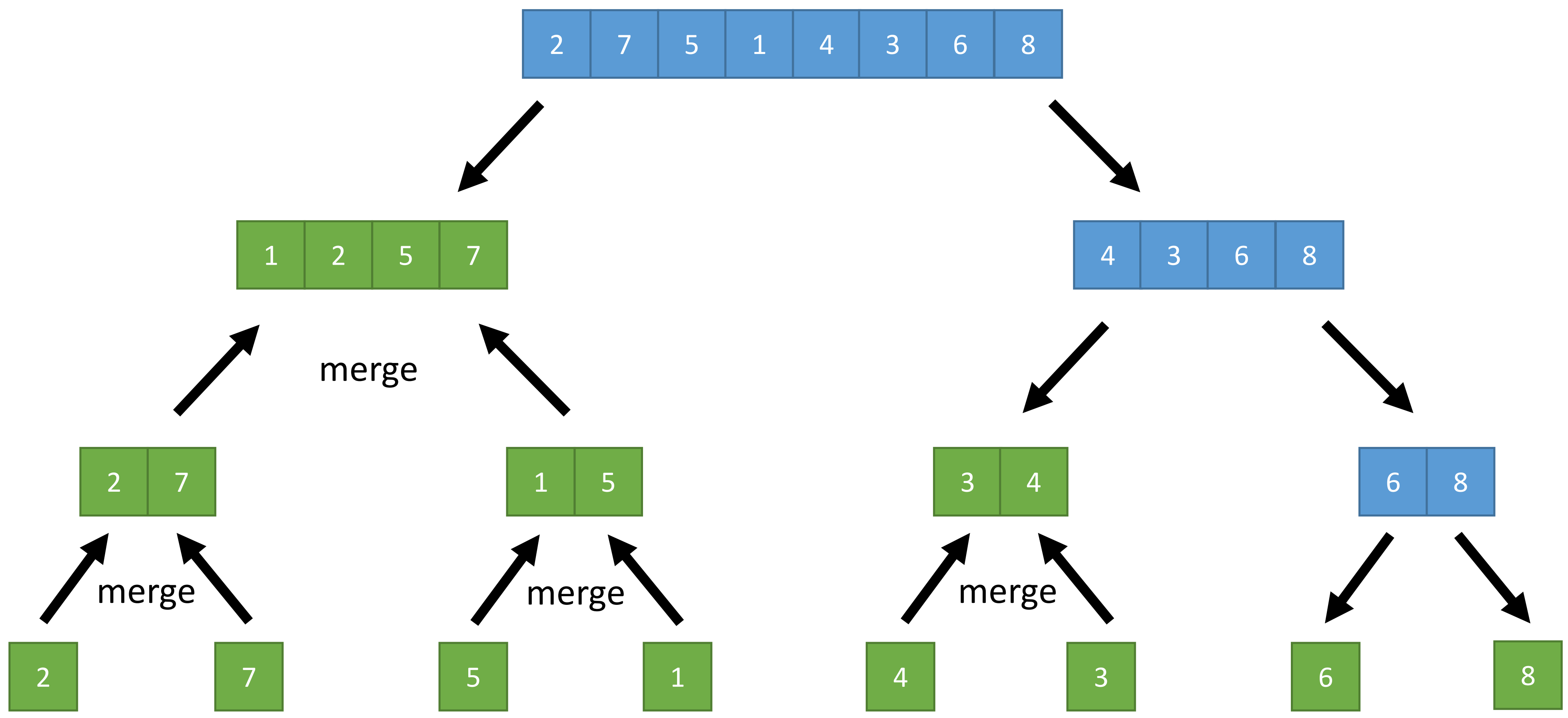
Merge Sort



Merge Sort

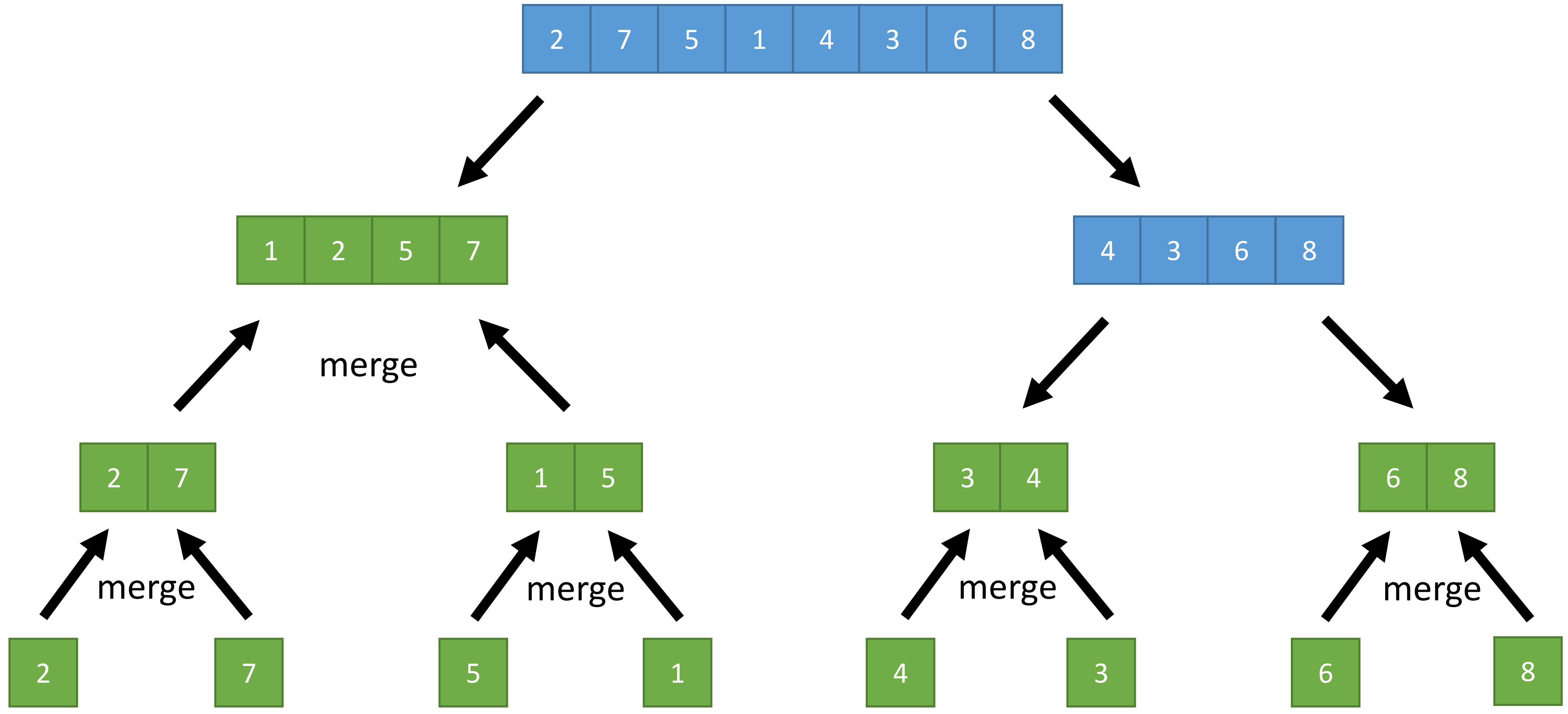


Merge Sort



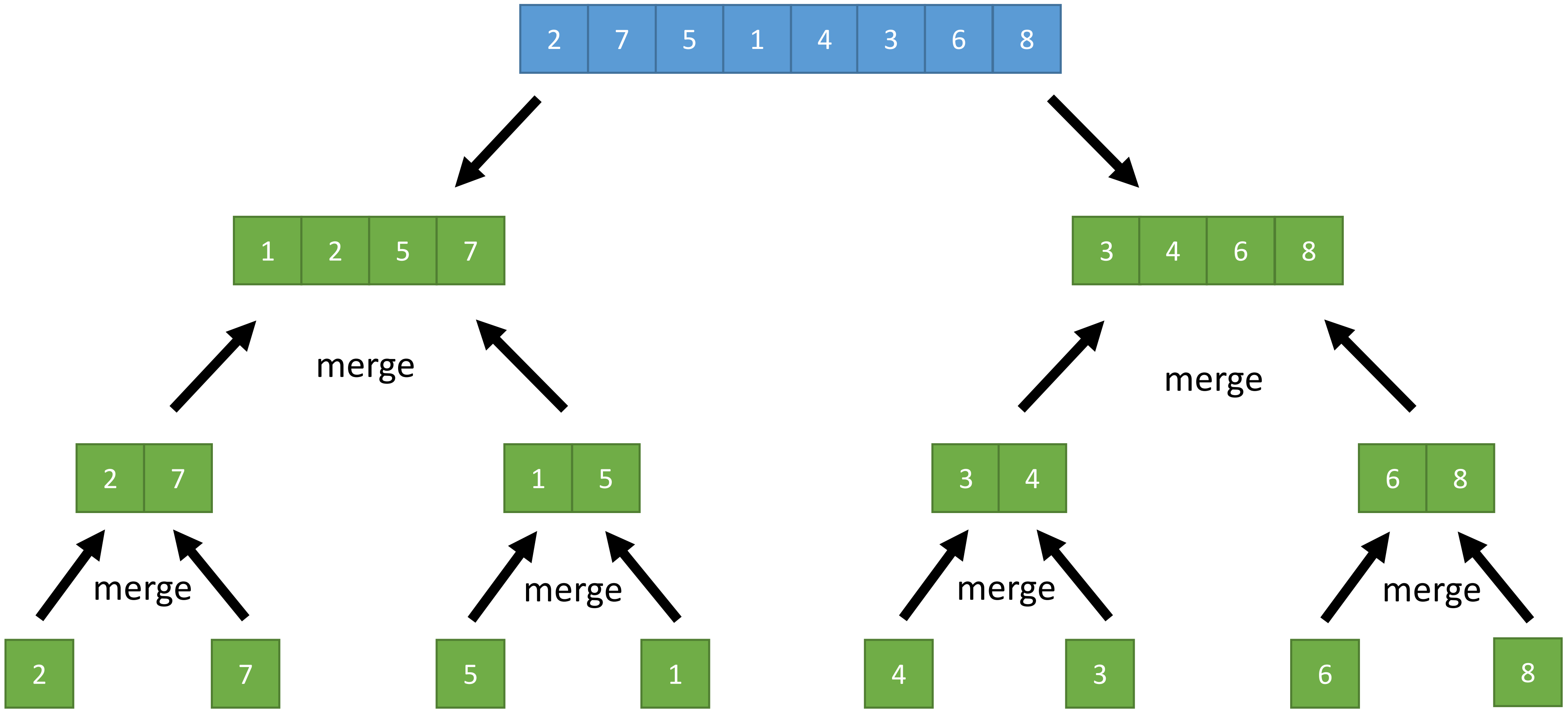
2 7 5 1 4 3 6 8
1 2 5 7
4 3 6 8
merge
2 7
1 5
3 4
6 8
merge
merge
merge
2
7
5
1
4
3
6
8

Merge Sort



2 7 5 1 4 3 6 8
1 2 5 7
4 3 6 8
merge
2 7
1 5
3 4
6 8
merge
merge
merge
merge
2
7
5
1
4
3
6
8

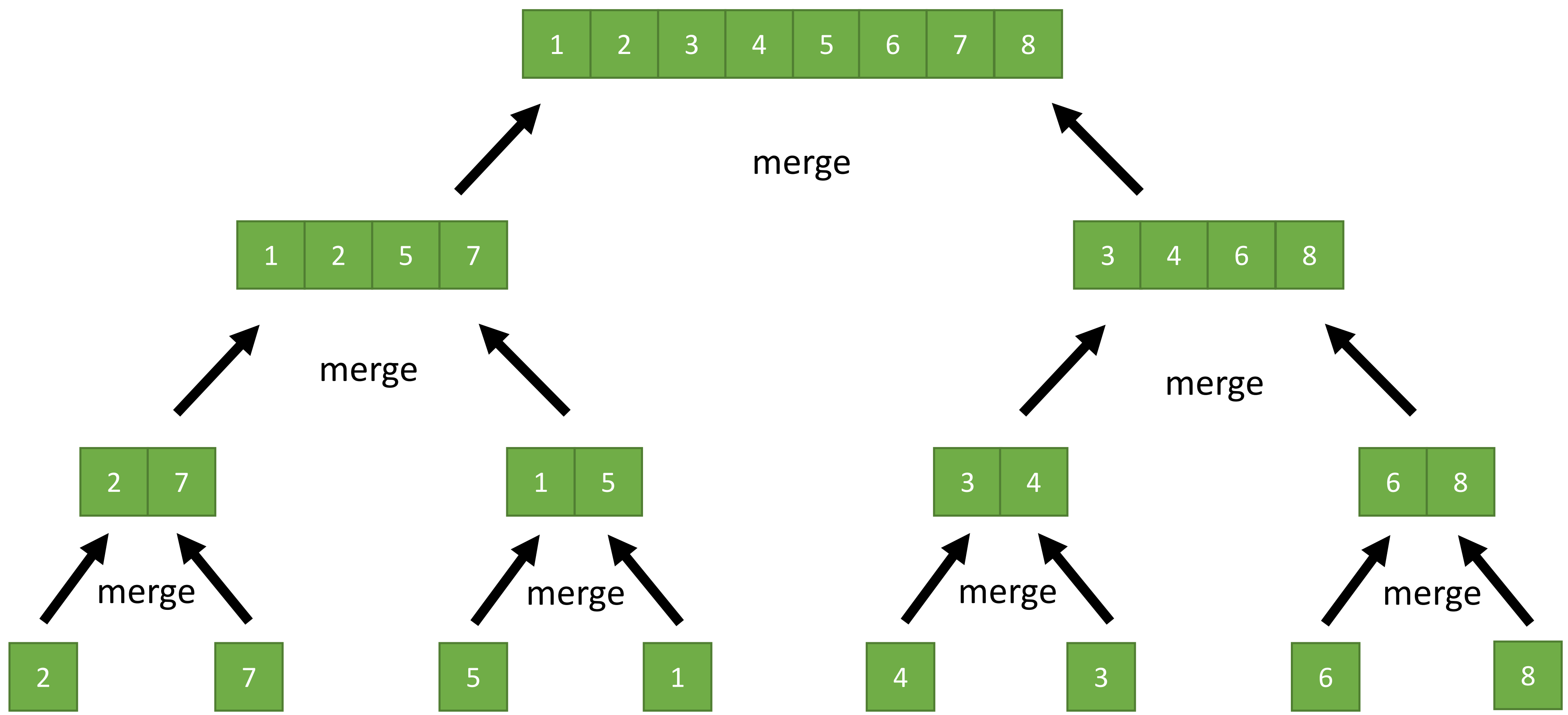
Merge Sort



2 7 5 1 4 3 6 8
1 2 5 7
3 4 6 8
merge
merge
2 7
1 5
3 4
6 8
merge
merge
merge
merge
2
7
5
1
4
3
6
8

Merge Sort

$O(N \log N)$



15 Sorting Algorithms in 6 Minutes
http://youtu.be/kPRA0W1kECg

Data structure complexity

Array
Vector

Linked list

Ordered map

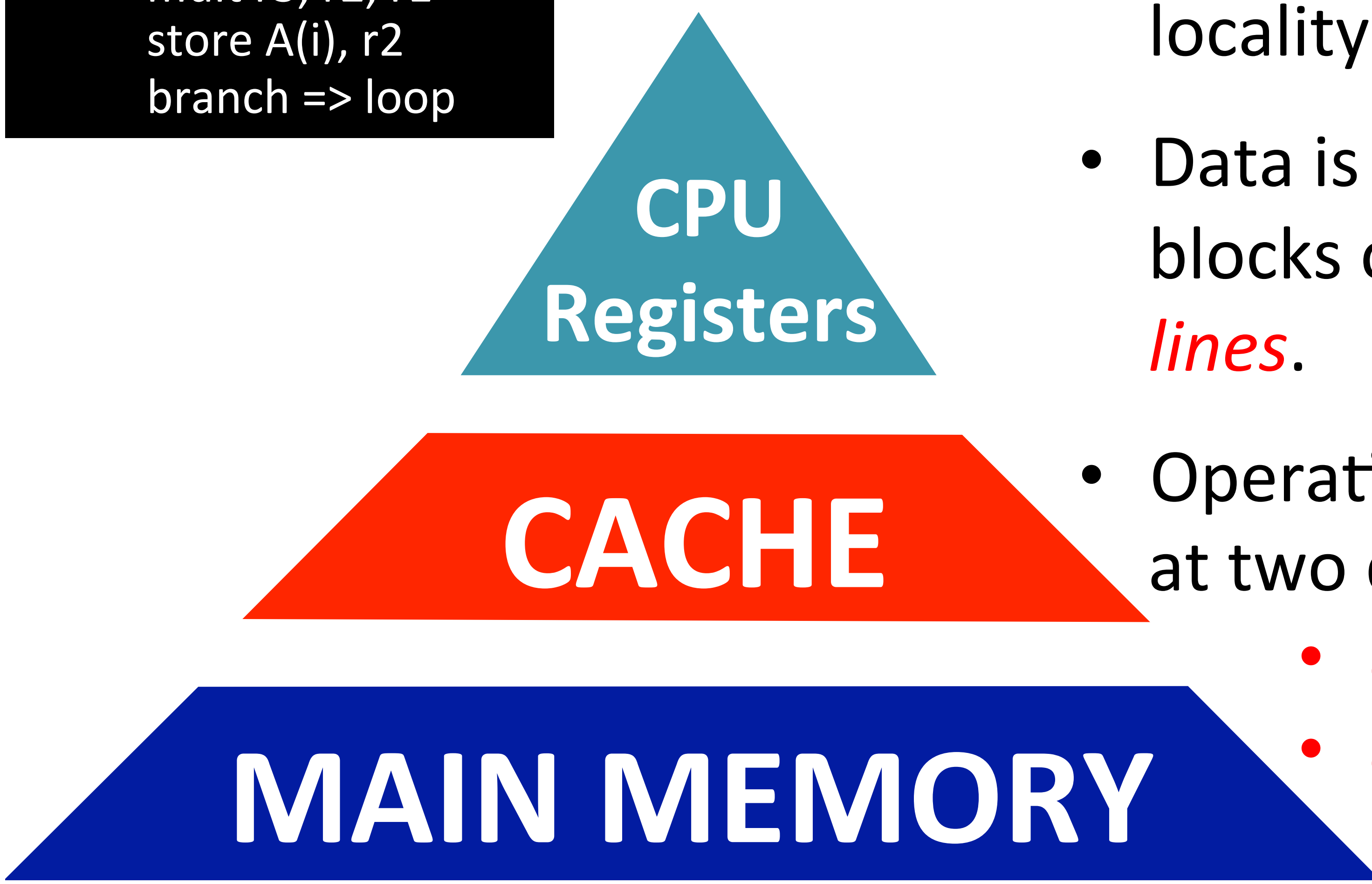
Hash table

http://bigocheatsheet.com/

Nicolai Josuttis, “The C++ Standard Library”

Cache Memory

```
Loop: load r1, A(i)
      load r2, s
      mult r3, r2, r1
      store A(i), r2
      branch => loop
```



- Designed for temporal/spatial locality
- Data is transferred to cache in blocks of fixed size, called *cache lines*.
- Operation of LOAD/STORE can lead at two different scenario:
  - *cache hit*
  - *cache miss*

Manual computer

L1 cache reference	0.5 s
Branch mispredict	5 s
L2 cache reference	7 s
Mutex lock/unlock	25 s
Main memory reference	100 s
SSD random read	1.7 days
Read 1 MB sequentially from RAM	2.9 days
Read 1 MB sequentially from SSD	11.6 days
HDD seek	16.5 weeks
Read 1 MB sequentially from HDD	7.8 months
Internet data packet EU -> USA -> EU	4.8 years

Optimization strategy

Don't optimize the whole code

Profile the code, find the bottlenecks
They may not always be where you thought they were

Break the problem down

Try to run the shortest possible test you can to get meaningful results
Isolate serial kernels

Keep a working version of the code!

Getting the wrong answer faster is not the goal.

Optimize on the architecture on which you intend to run

Optimizations for one architecture will not necessarily translate

The compiler is your friend!

If you find yourself coding in machine language, you are doing to much.

This is the most important slide in the talk

Never, ever optimize unless you have good reason to.

- Why do you need to optimize?
- Do you have a clear plan of action?
- What do you expect to gain?
- How long will it take?
- Are you still sure it's worth it?