



# High-level Optimization

David Grellscheid



UNIVERSITETET I BERGEN



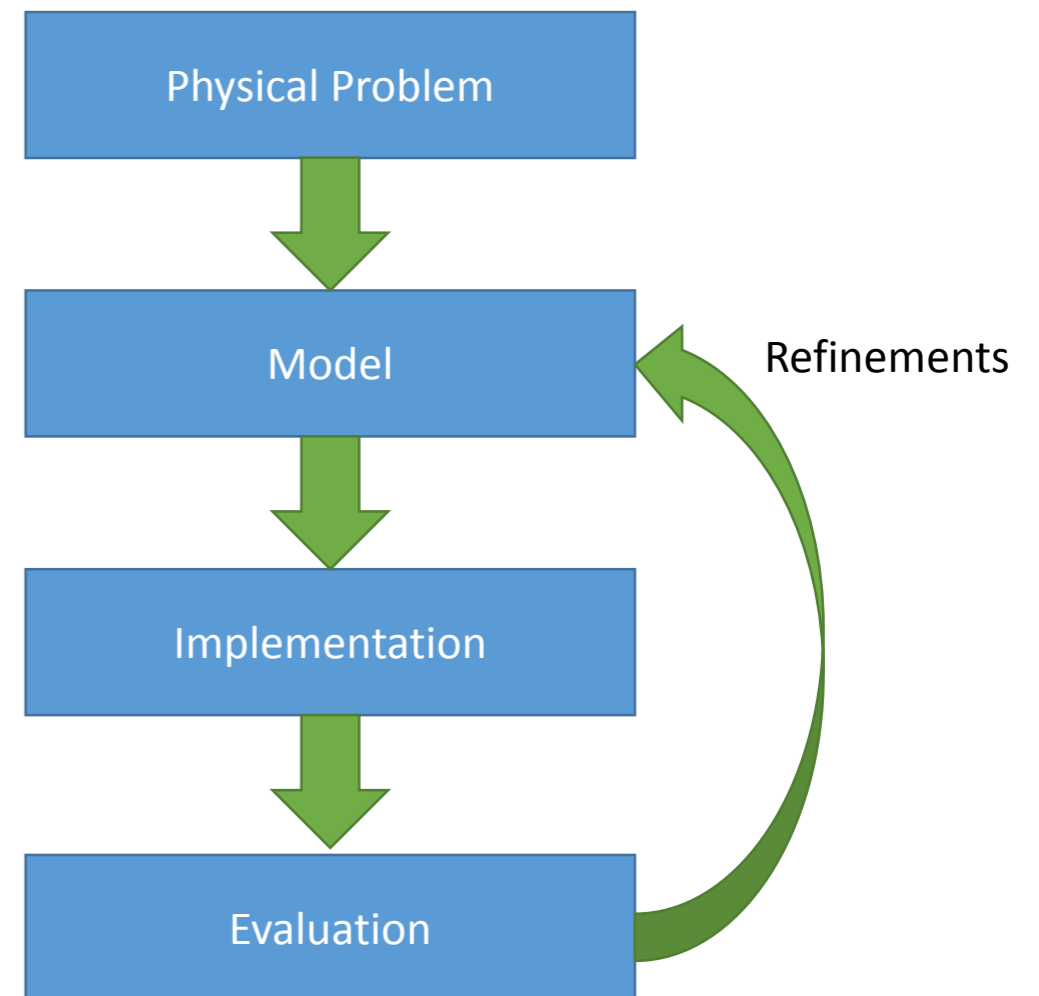
# Typical scientific workflow

Correctness is main  
concern

Start coding without  
much planning

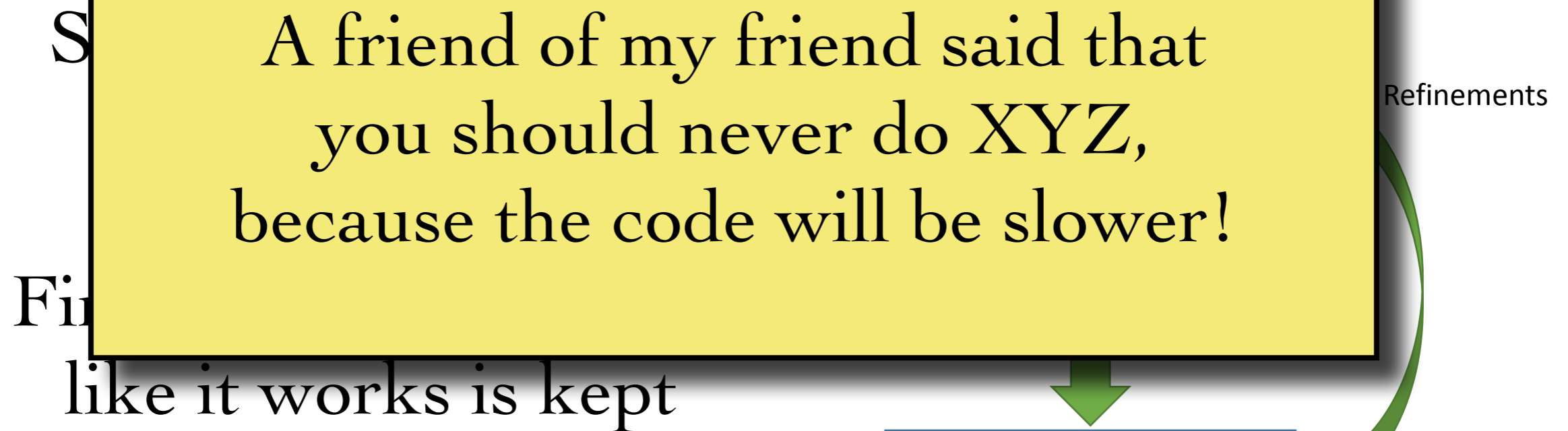
First version that looks  
like it works is kept

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



# Typical scientific workflow

Correctness is main  
concern



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

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, Go, 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 :-)**

# Optimization points

Algorithm / data structure choice

can get orders of magnitude in speed

Local and hardware-specific optimisations

*- next lecture -*

# Complexity basics

Much simplified, skipping formal derivation

# Complexity basics

Much simplified, skipping formal derivation

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

# Complexity basics

Much simplified, skipping formal derivation

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

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  $N$

# Complexity basics

Much simplified, skipping formal derivation

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

$O(NN!)$

Scaling behaviour with size  $N$  of problem set:

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

$O(N)$  - linear with  $N$

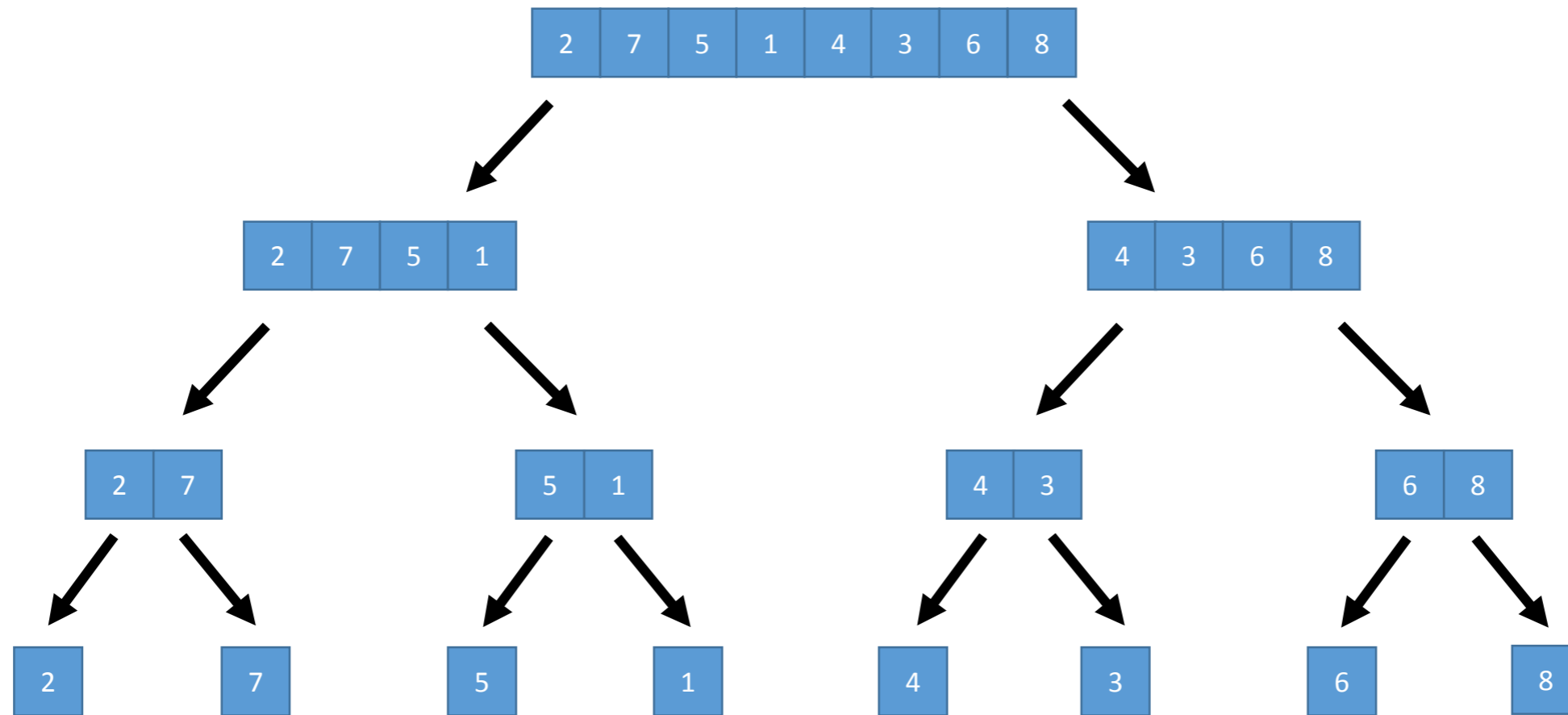
$O(N^2)$  - quadratic in  $N$



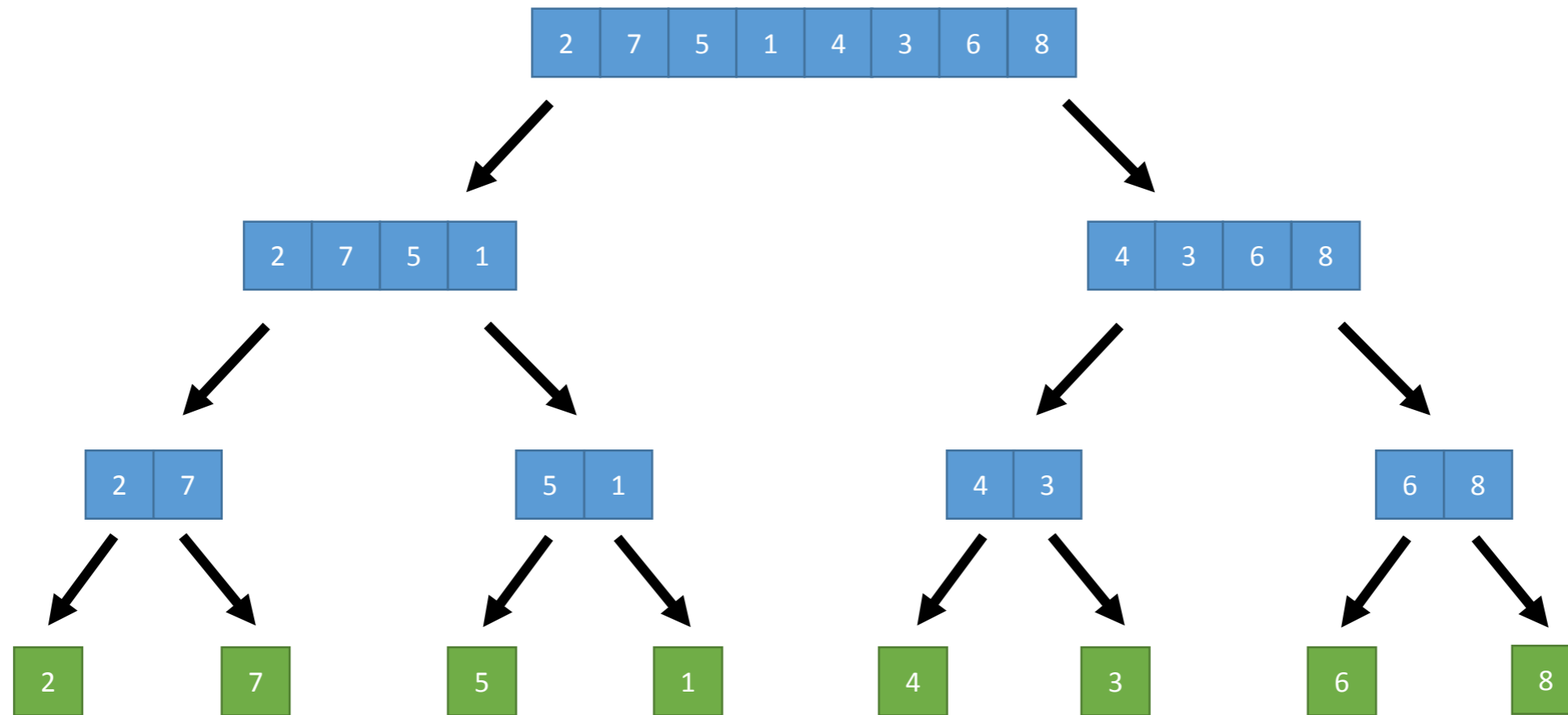
# Merge Sort



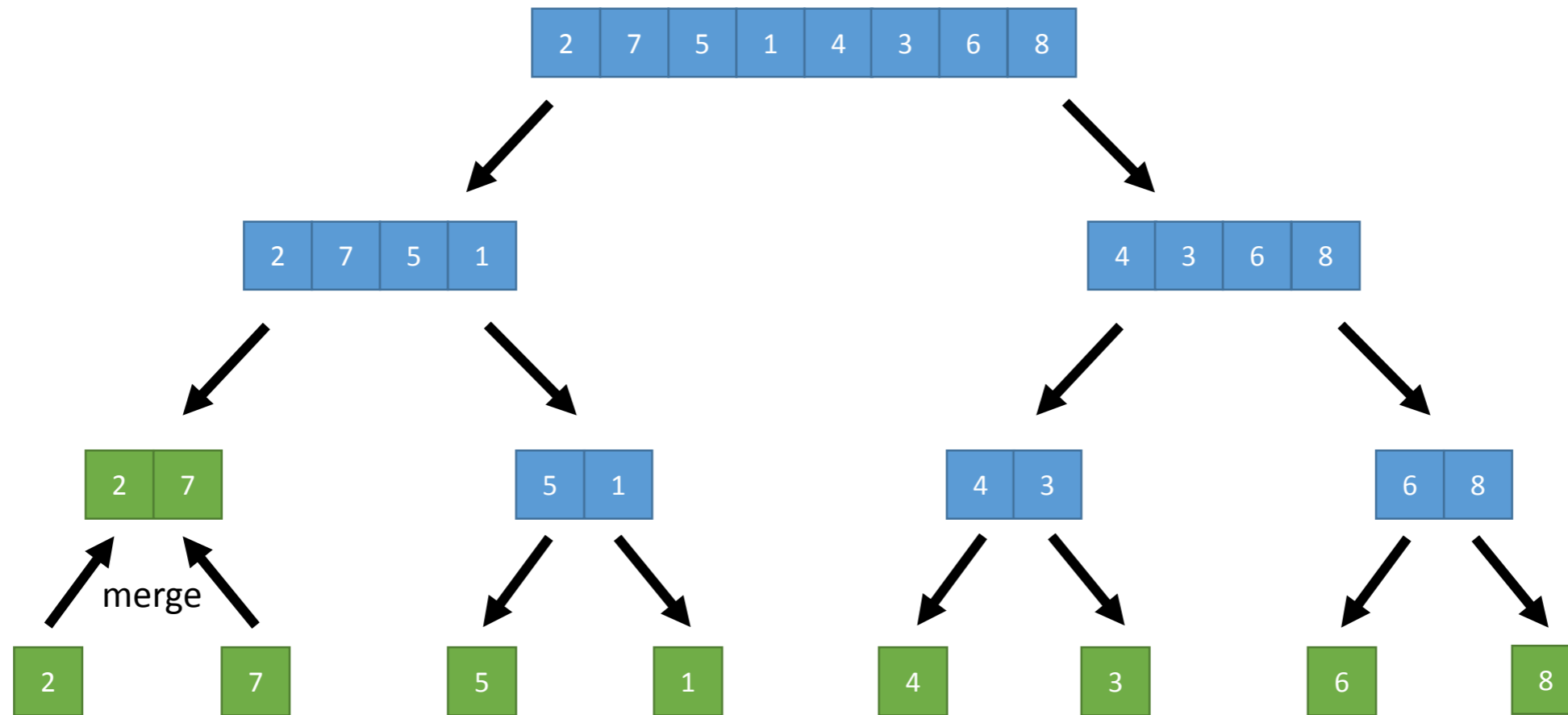
# Merge Sort



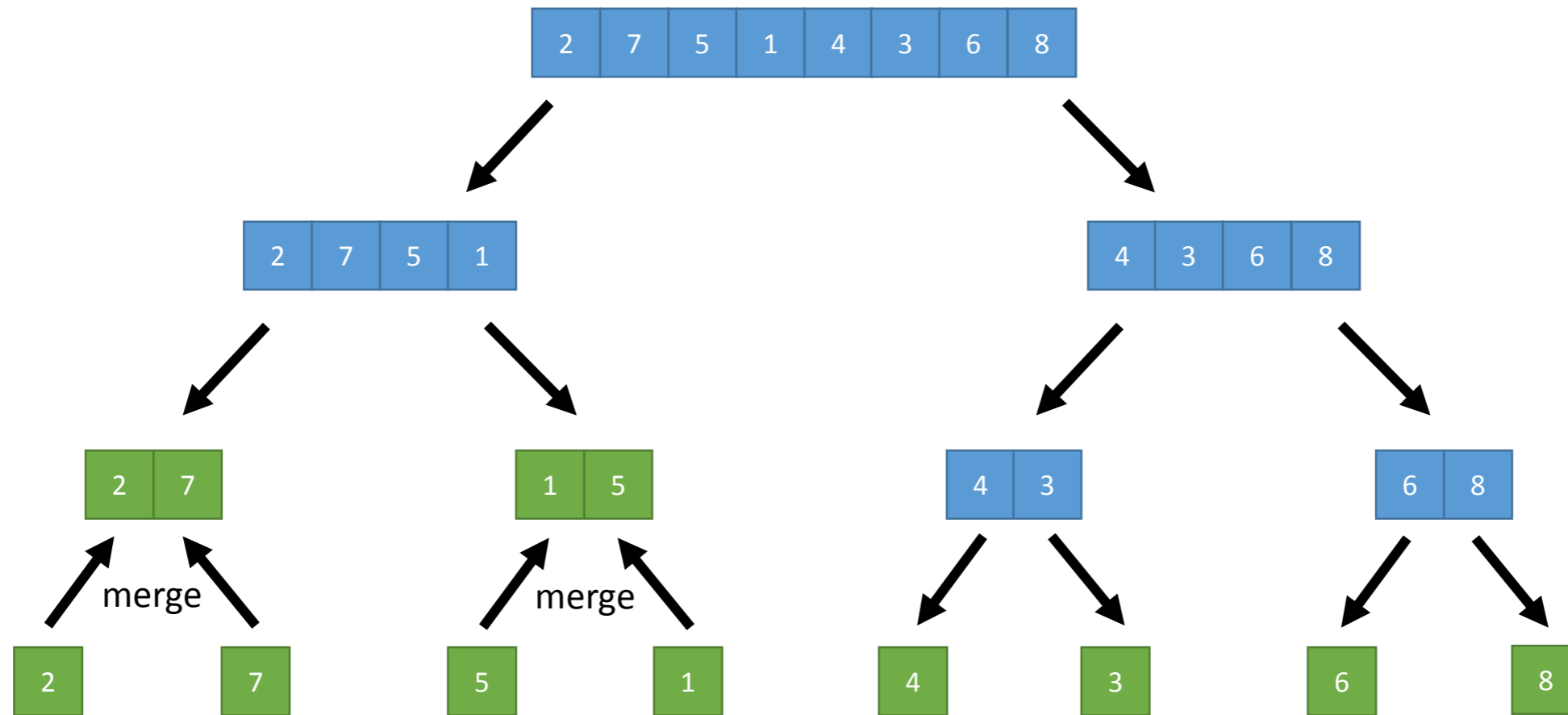
# Merge Sort



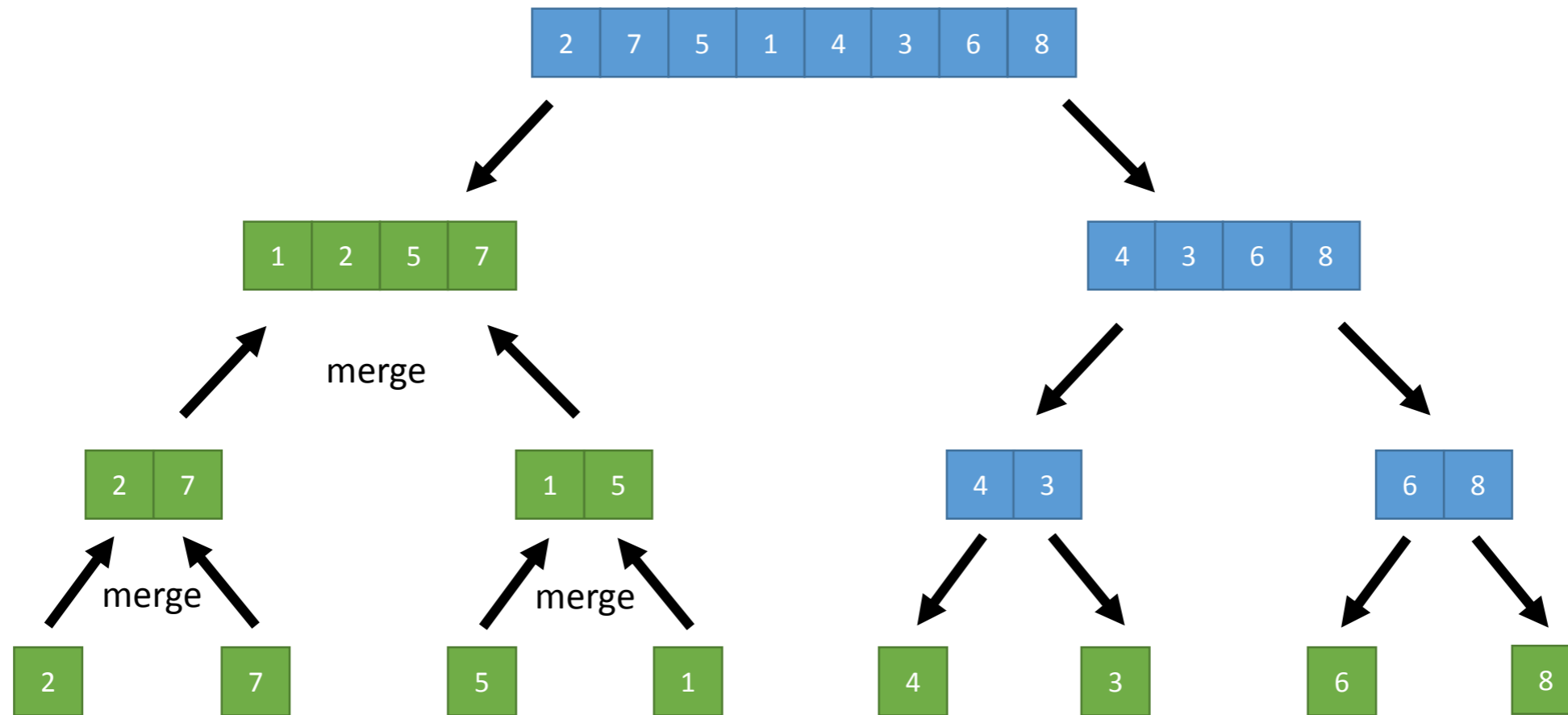
# Merge Sort



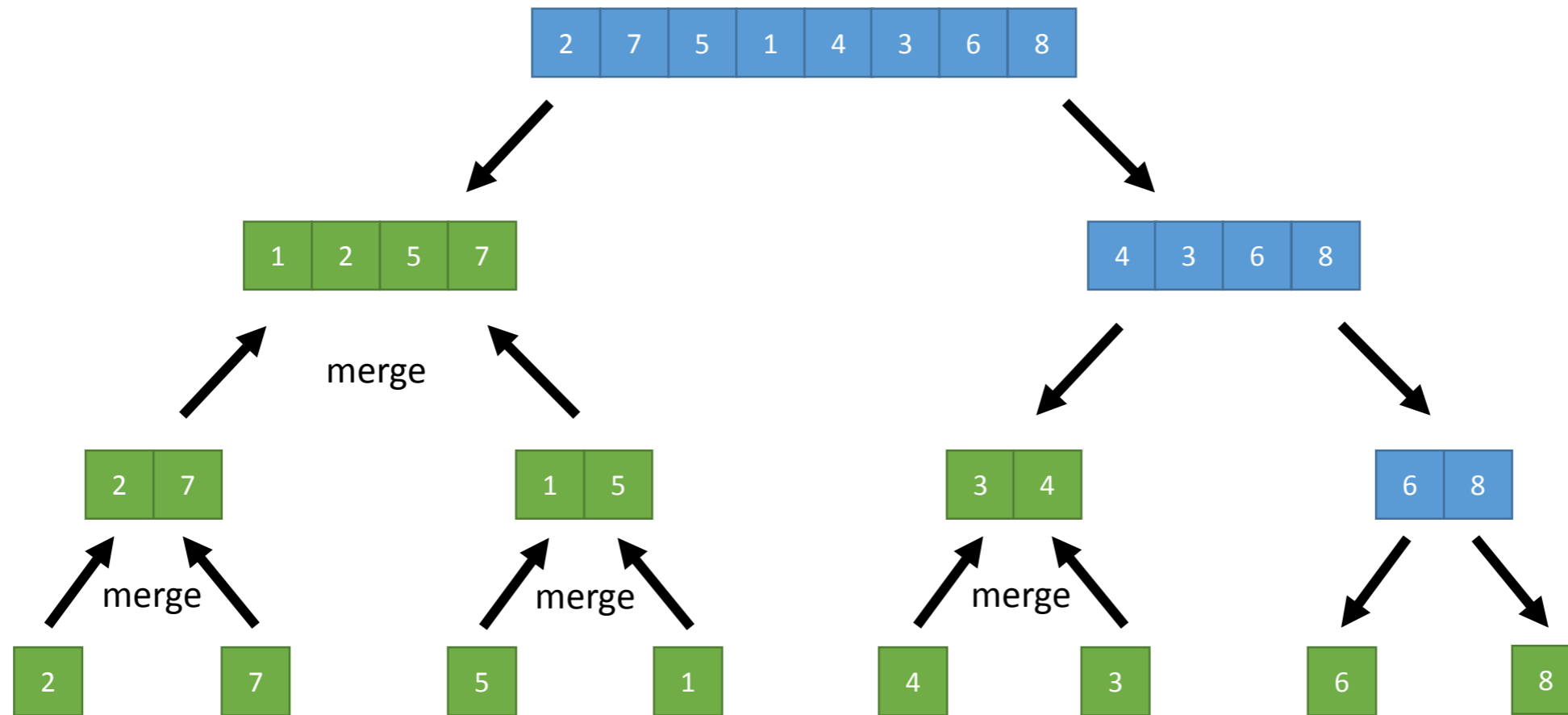
# Merge Sort



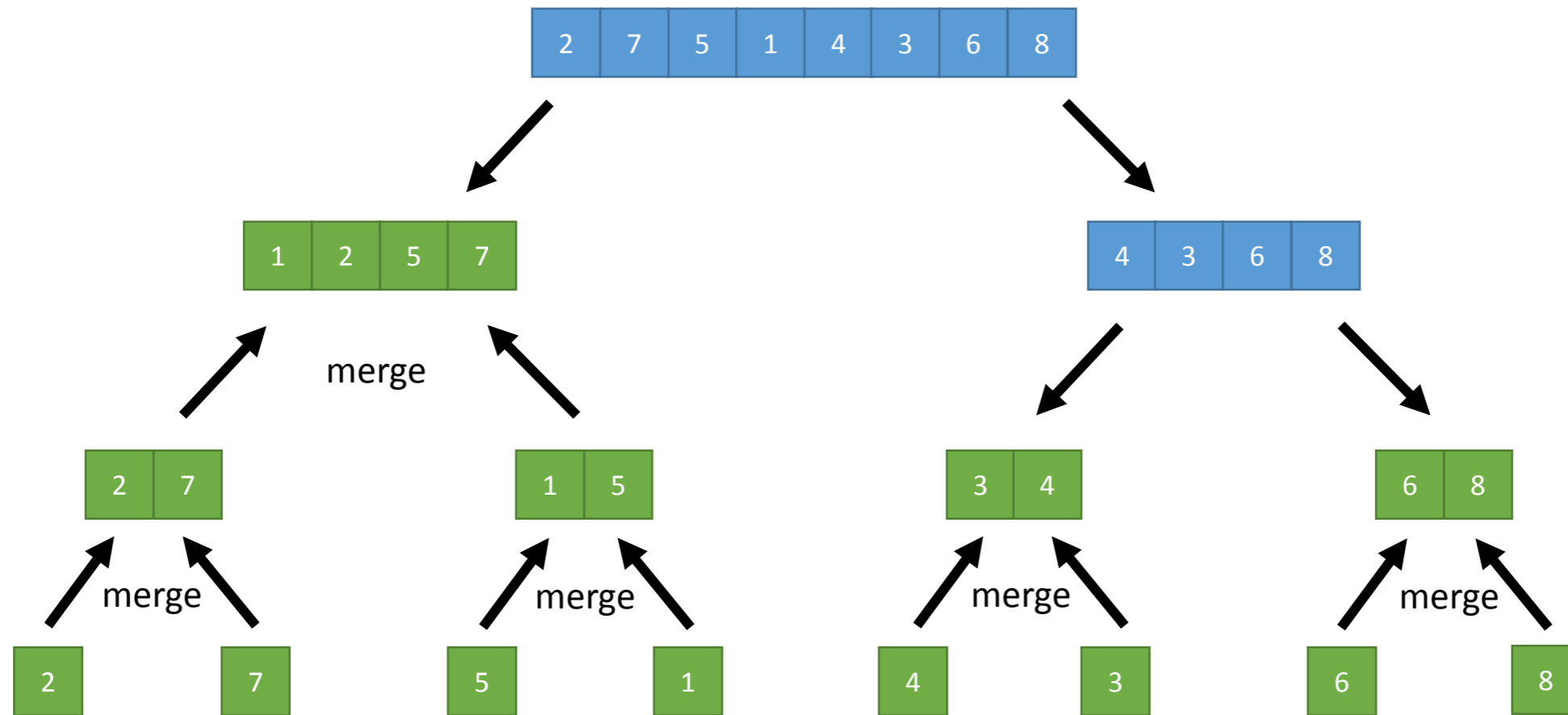
# Merge Sort



# Merge Sort

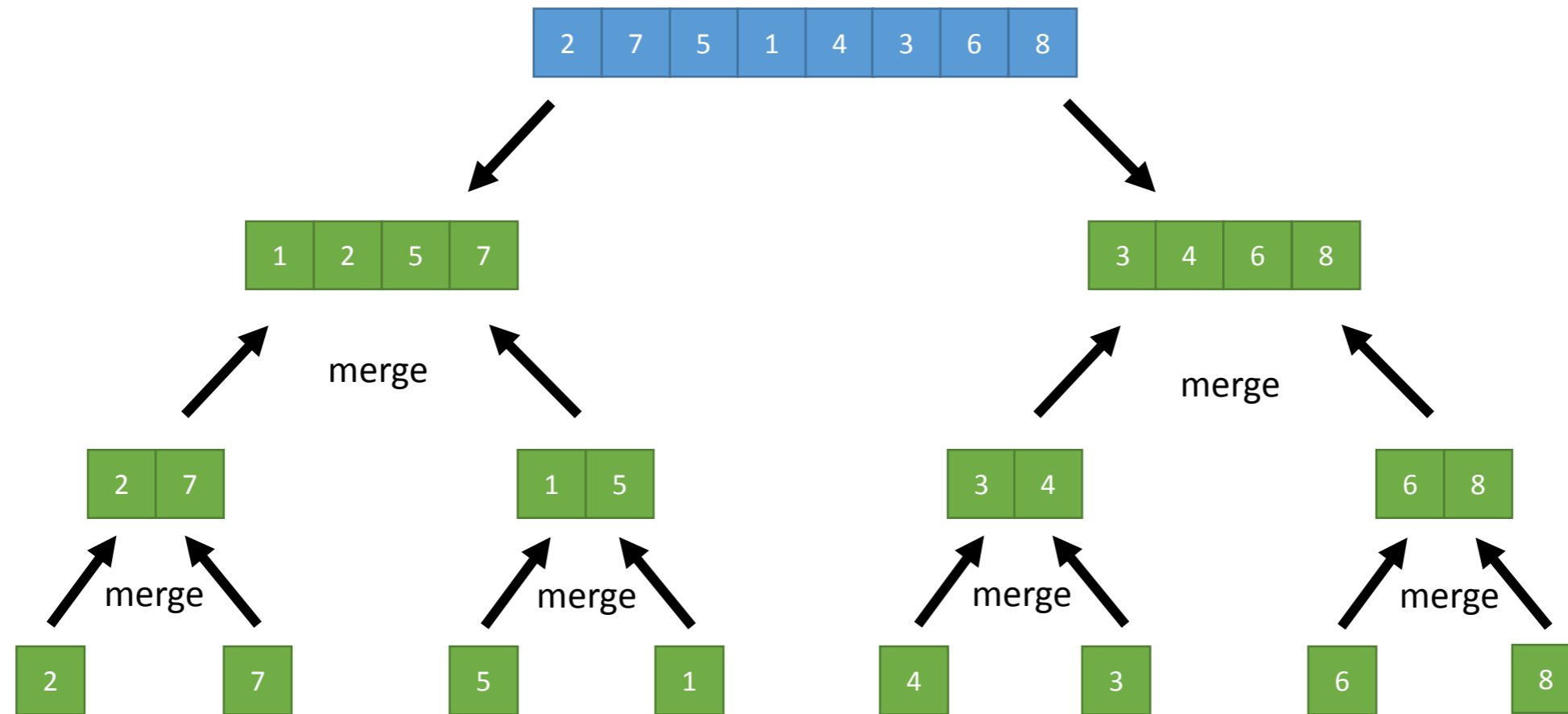


# Merge Sort

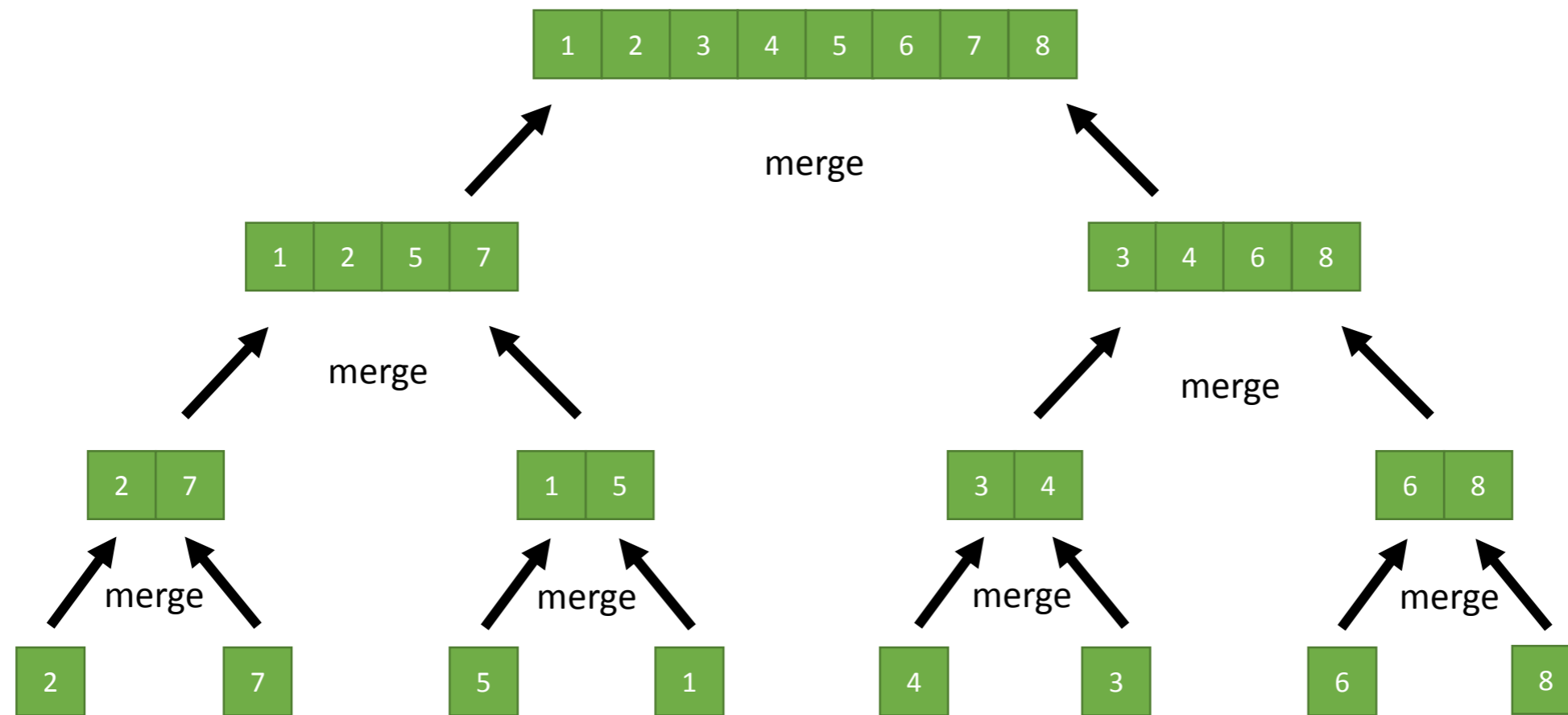




# Merge Sort

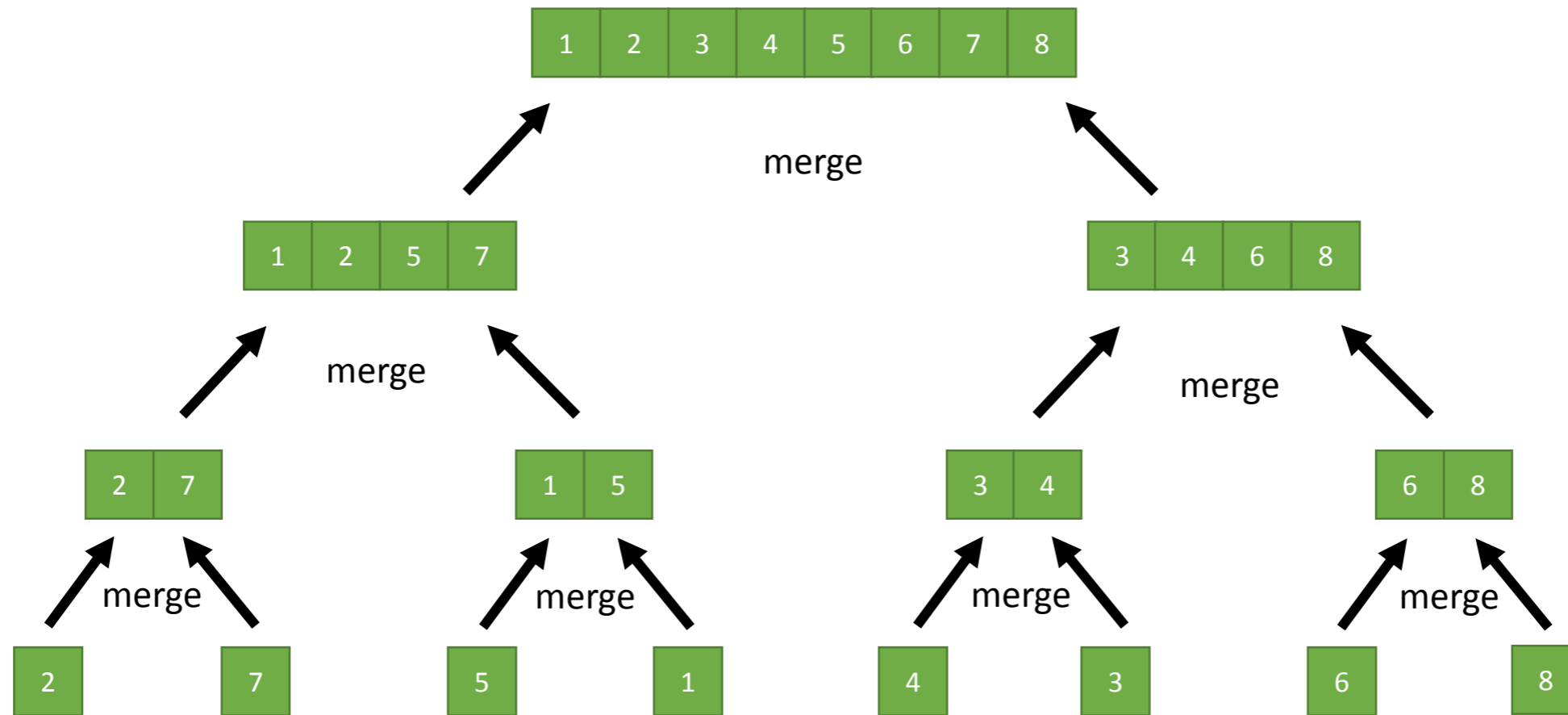


# Merge Sort



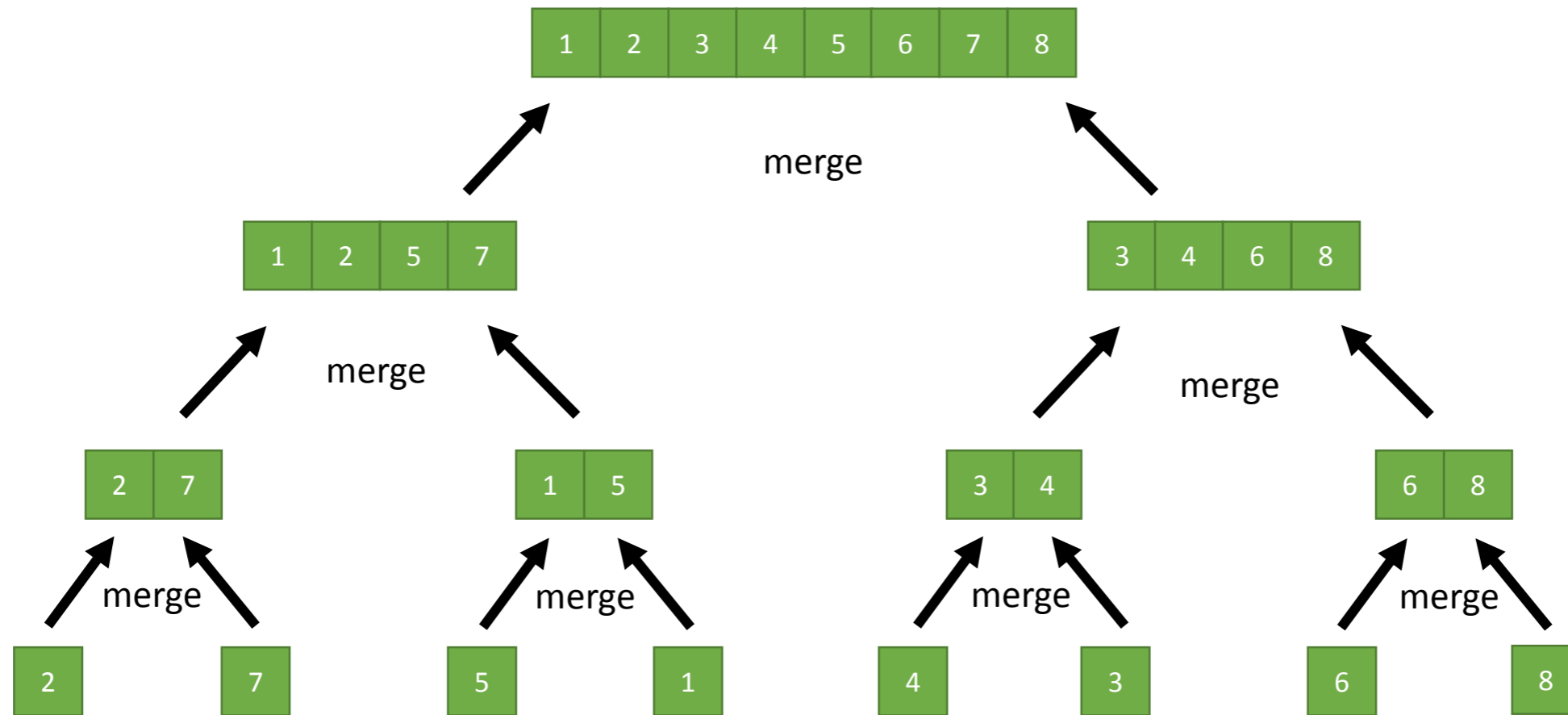
# Merge Sort

$O(N \log N)$



# Merge Sort

$O(N \log N)$



15 Sorting Algorithms in 6 Minutes

<http://youtu.be/kPRA0W1kECg>



# Data structure complexity

`std::array`  
`std::vector`

`std::list`

`std::map`

`std::unordered_map` (hash table)

<http://bigocheatsheet.com/>

Nicolai Josuttis, “The C++ Standard Library”