ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 3: Sorting and Search Algorithms
Lecture 9: Sorting: Guidelines for standard libraries

Maxim Buzdalov
Saint Petersburg 2016

General words

This video is dedicated to standard libraries of various languages.

General words

This video is dedicated to standard libraries of various languages.
The Most Important Primary Advice:

General words

This video is dedicated to standard libraries of various languages.
The Most Important Primary Advice:

NEVER, EVER WRITE YOUR OWN SORTING CODE
if a standard one will do

General words

This video is dedicated to standard libraries of various languages.
The Most Important Primary Advice:

NEVER, EVER WRITE YOUR OWN SORTING CODE
if a standard one will do

- Your code has more bugs than the standard library's

General words

This video is dedicated to standard libraries of various languages.
The Most Important Primary Advice:

NEVER, EVER WRITE YOUR OWN SORTING CODE
if a standard one will do

- Your code has more bugs than the standard library's
- Your code is less tuned $\rightarrow$ performs worse

This video is dedicated to standard libraries of various languages.
The Most Important Primary Advice:

NEVER, EVER WRITE YOUR OWN SORTING CODE
if a standard one will do

- Your code has more bugs than the standard library's
- Your code is less tuned $\rightarrow$ performs worse
- You may write your own sorting if:
  - it is fast because it is problem-specific (e.g. radix sort)

General words

This video is dedicated to standard libraries of various languages.
The Most Important Primary Advice:

NEVER, EVER WRITE YOUR OWN SORTING CODE
if a standard one will do

- Your code has more bugs than the standard library's
- Your code is less tuned $\to$ performs worse
- You may write your own sorting if:
  - it is fast because it is problem-specific (e.g. radix sort)
  - the standard one eats too much memory (Java and objects)

This video is dedicated to standard libraries of various languages.
The Most Important Primary Advice:

NEVER, EVER WRITE YOUR OWN SORTING CODE
if a standard one will do

- Your code has more bugs than the standard library's
- Your code is less tuned $\rightarrow$ performs worse
- You may write your own sorting if:
  - it is fast because it is problem-specific (e.g. radix sort)
  - the standard one eats too much memory (Java and objects)
- If you write quicksort, write it randomized

- Avoid using `qsort` from the C library:
  - Too simple to be killed by special tests
  - Comparator is never inlined $\rightarrow$ slower than C++ sorts

- Avoid using `qsort` from the C library:
  - Too simple to be killed by special tests
  - Comparator is never inlined $\rightarrow$ slower than C++ sorts
- `std::sort`: a quicksort with hacks
  - If spent too much time, switches to a heapsort (slow, in-place, $\Theta(N \log N)$)
  - Guaranteed $O(N \log N)$ on any input data, not a stable sorting

- Avoid using `qsort` from the C library:
  - Too simple to be killed by special tests
  - Comparator is never inlined $\to$ slower than C++ sorts
- `std::sort`: a quicksort with hacks
  - If spent too much time, switches to a heapsort (slow, in-place, $\Theta(N \log N)$)
  - Guaranteed $O(N \log N)$ on any input data, not a stable sorting
- `std::stable_sort`: a mergesort with tuning
  - Generally slower than `std::sort`, but nevertheless $O(N \log N)$
  - Stable, as follows from the name

- Avoid using `qsort` from the C library:
  - Too simple to be killed by special tests
  - Comparator is never inlined $\rightarrow$ slower than C++ sorts
- `std::sort`: a quicksort with hacks
  - If spent too much time, switches to a heapsort (slow, in-place, $\Theta(N \log N)$)
  - Guaranteed $O(N \log N)$ on any input data, not a stable sorting
- `std::stable_sort`: a mergesort with tuning
  - Generally slower than `std::sort`, but nevertheless $O(N \log N)$
  - Stable, as follows from the name
- Comparator for `std::sort` and `std::stable_sort`:
  - An optional argument x which can be called as $x(a, b)$ and returns whether $a < b$
    - Pointer to function (old style, not inlinable, so not recommended)
    - A class with `bool operator ()(T const &, T const &) const`

- Java has three families of sorting algorithms:

- Java has three families of sorting algorithms:
  - `java.util.Arrays.sort(primitive[])`: a dual-pivot quicksort
    - Not stable, but nobody misses it, because it…
    - Does not support comparators: only natural ordering
    - As majority of quicksorts, can be degraded to $\Theta(N^2)$. Use with care.

Java

- Java has three families of sorting algorithms:
  - `java.util.Arrays.sort(primitive[])`: a dual-pivot quicksort
    - Not stable, but nobody misses it, because it. . .
    - Does not support comparators: only natural ordering
    - As majority of quicksorts, can be degraded to $\Theta(N^2)$. Use with care.
  - `java.util.Arrays.sort(Object[])`: used to be a tuned mergesort, now TimSort
    - Stable and fool-proof, supports comparators
    - TimSort: mergesort with various “best-case” improvements, originally from Python

- Java has three families of sorting algorithms:
  - `java.util.Arrays.sort(primitive[])`: a dual-pivot quicksort
    - Not stable, but nobody misses it, because it…
    - Does not support comparators: only natural ordering
    - As majority of quicksorts, can be degraded to $\Theta(N^2)$. Use with care.
  - `java.util.Arrays.sort(Object[])`: used to be a tuned mergesort, now TimSort
    - Stable and fool-proof, supports comparators
    - TimSort: mergesort with various “best-case” improvements, originally from Python
  - `java.util.Collections.<T>sort(Collection<T>)`
    - Same algorithm as the previous one, maybe differently tuned
    - Often reduces to copying data to an array and calling `Arrays.sort`
    - …so even more extra memory needed!

- Java has three families of sorting algorithms:
  - `java.util.Arrays.sort(primitive[])`: a dual-pivot quicksort
    - Not stable, but nobody misses it, because it…
    - Does not support comparators: only natural ordering
    - As majority of quicksorts, can be degraded to $\Theta(N^2)$. Use with care.
  - `java.util.Arrays.sort(Object[])`: used to be a tuned mergesort, now TimSort
    - Stable and fool-proof, supports comparators
    - TimSort: mergesort with various “best-case” improvements, originally from Python
  - `java.util.Collections.<T>sort(Collection<T>)`
    - Same algorithm as the previous one, maybe differently tuned
    - Often reduces to copying data to an array and calling `Arrays.sort`
    - …so even more extra memory needed!
  - Comparator: an implementation of `java.util.Comparator<T>` interface
    - Implement `public int compare(T, T)` with result types “$<$”, “$=$”, “$>$”
    - In Java 8, can be done by a lambda