quick-bench.com

Fred Tingaud

@FredTingaudDev

A micro-benchmarking online tool

Quick C++ Benchmark

More

```
1  static void BM_StringCreation(benchmark::State& state) {
2    while (state.KeepRunning())
3      std::string empty_string;
4  }
5  // Register the function as a benchmark
6  BENCHMARK(BM_StringCreation);
7
8  static void BM_StringCopy(benchmark::State& state) {
9    std::string x = "hello";
10   while (state.KeepRunning())
11     std::string copy(x);
12 }
13 BENCHMARK(BM_StringCopy);
14
```

compiler = clang++-3.8 | std = c++1z | optim = O1

Run Full Benchmark | Run Light Benchmark

A micro-benchmarking online tool

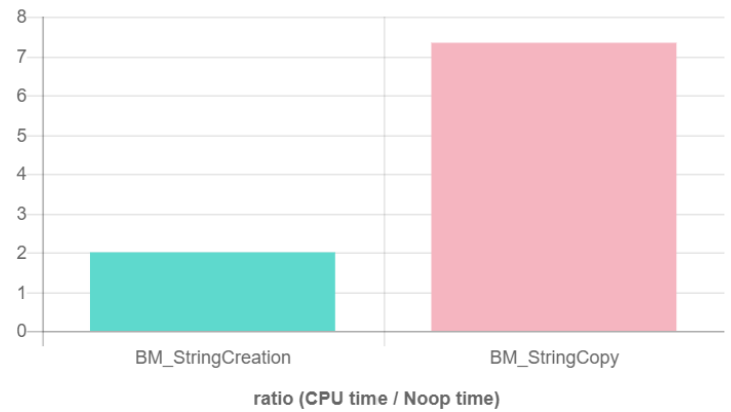
Quick C++ Benchmark

More

```
1  static void BM_StringCreation(benchmark::State& state) {
2    while (state.KeepRunning())
3      std::string empty_string;
4  }
5  // Register the function as a benchmark
6  BENCHMARK(BM_StringCreation);
7
8  static void BM_StringCopy(benchmark::State& state) {
9    std::string x = "hello";
10   while (state.KeepRunning())
11     std::string copy(x);
12 }
13 BENCHMARK(BM_StringCopy);
14
```

compiler = clang++-3.8 | std = c++1z | optim = O1

Run Full Benchmark | Run Light Benchmark | Force full recalculation



Show Noop bar

Google Benchmark

```
static void StringCopy(benchmark::State& state){
  std::string x = "hello";
  while (state.KeepRunning())
    std::string copy(x);
}
BENCHMARK(StringCopy);
```

A use case

Let's experiment on `<algorithm>` sorting tools by sorting all or part of a vector.

Input

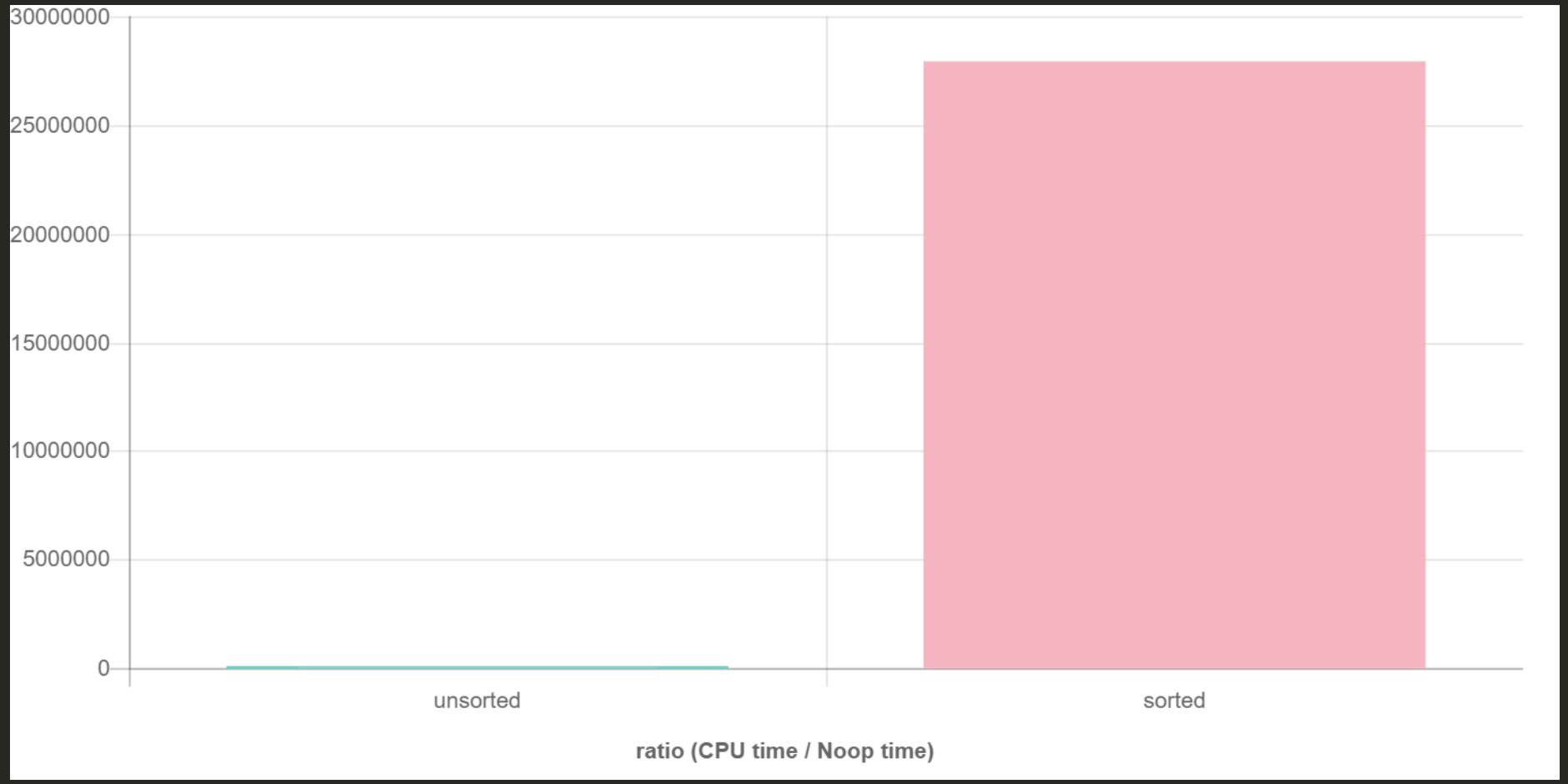
```
/**
 * returns a 1.000.000 elements std::vector<int>
 * generated by a Mersenne Twister random engine
 */
std::vector<int> getRandomVect();
```

Sorted

```
static void sorted(benchmark::State& state) {
  auto input = getRandomVect();
  while (state.KeepRunning())
  {
    std::vector<int> values(input);
    std::sort(values.begin(), values.end());
  }
}
BENCHMARK(sorted);
```

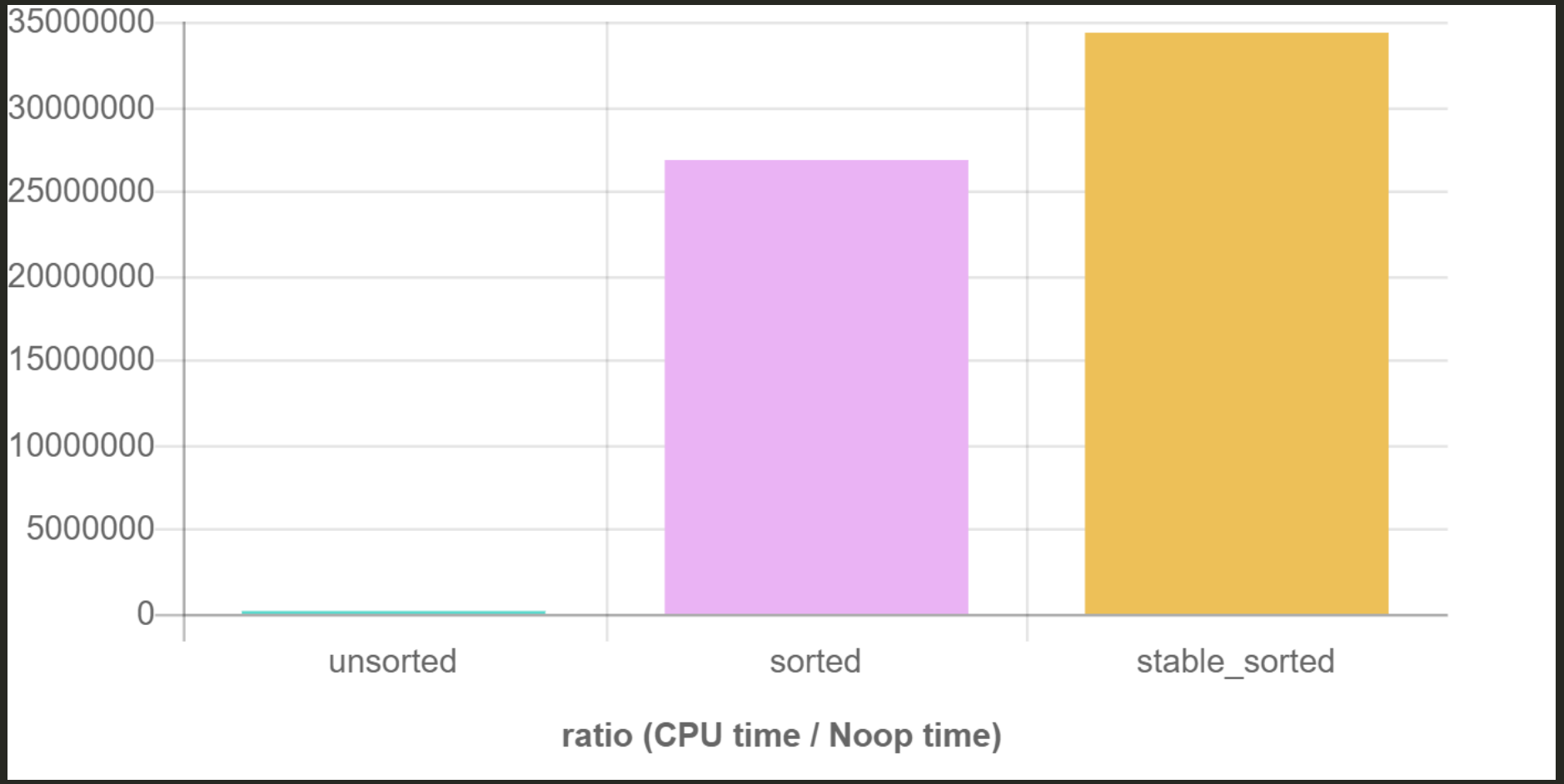
The Reference

```
static void unsorted(benchmark::State& state) {
  auto input = getRandomVect();
  while (state.KeepRunning())
  {
    std::vector<int> values(input);
    benchmark::DoNotOptimize(values);
  }
}
BENCHMARK(unsorted);
```


30000000
25000000
20000000
15000000
10000000
5000000
0
unsorted
sorted
ratio (CPU time / Noop time)

Now we can start comparing

sort VS stable_sort

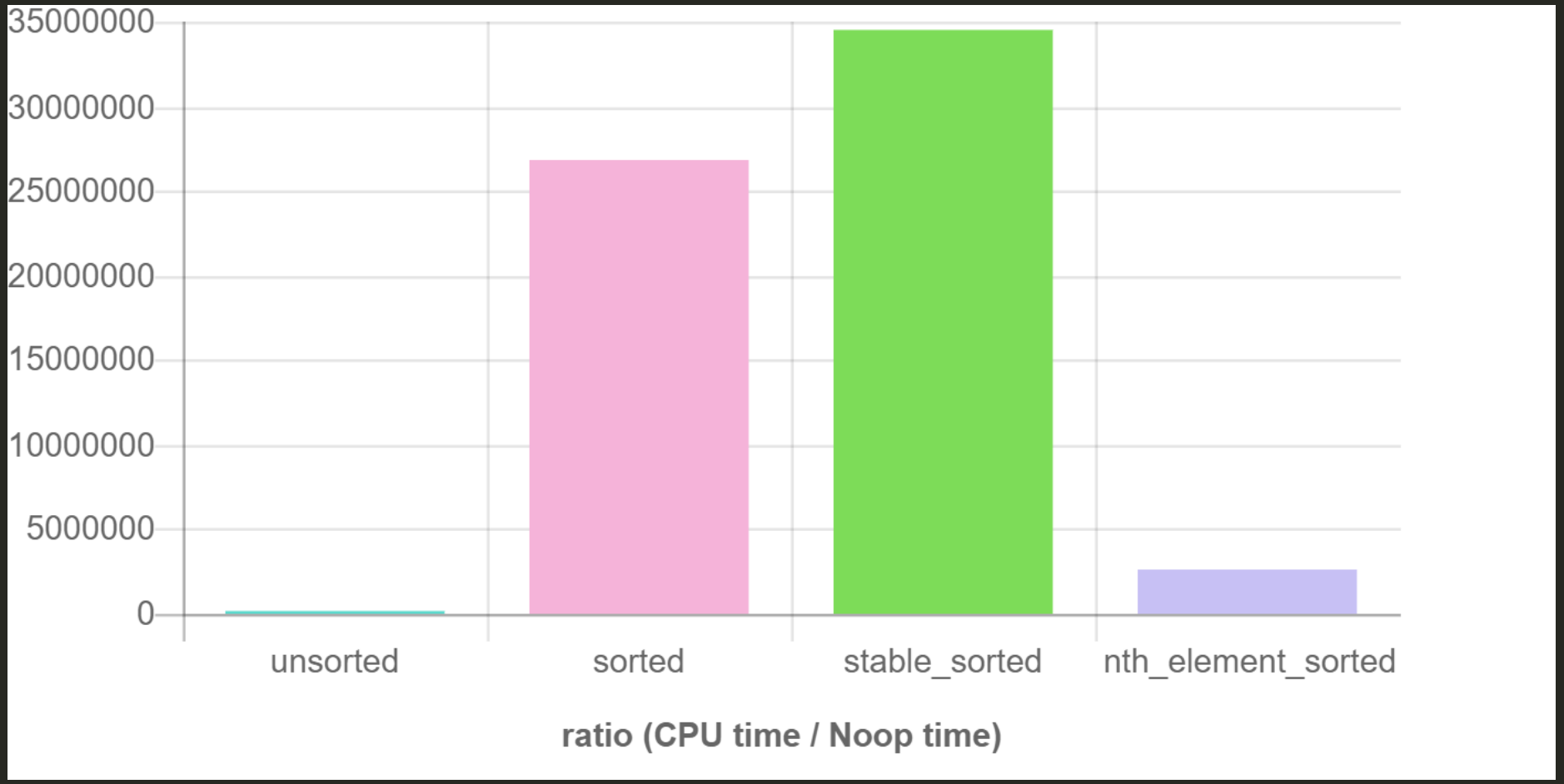

35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
unsorted
sorted
stable_sorted
ratio (CPU time / Noop time)

Nth element

If we are looking for the median, we don't need to sort everything.

We can use `std::nth_element`.

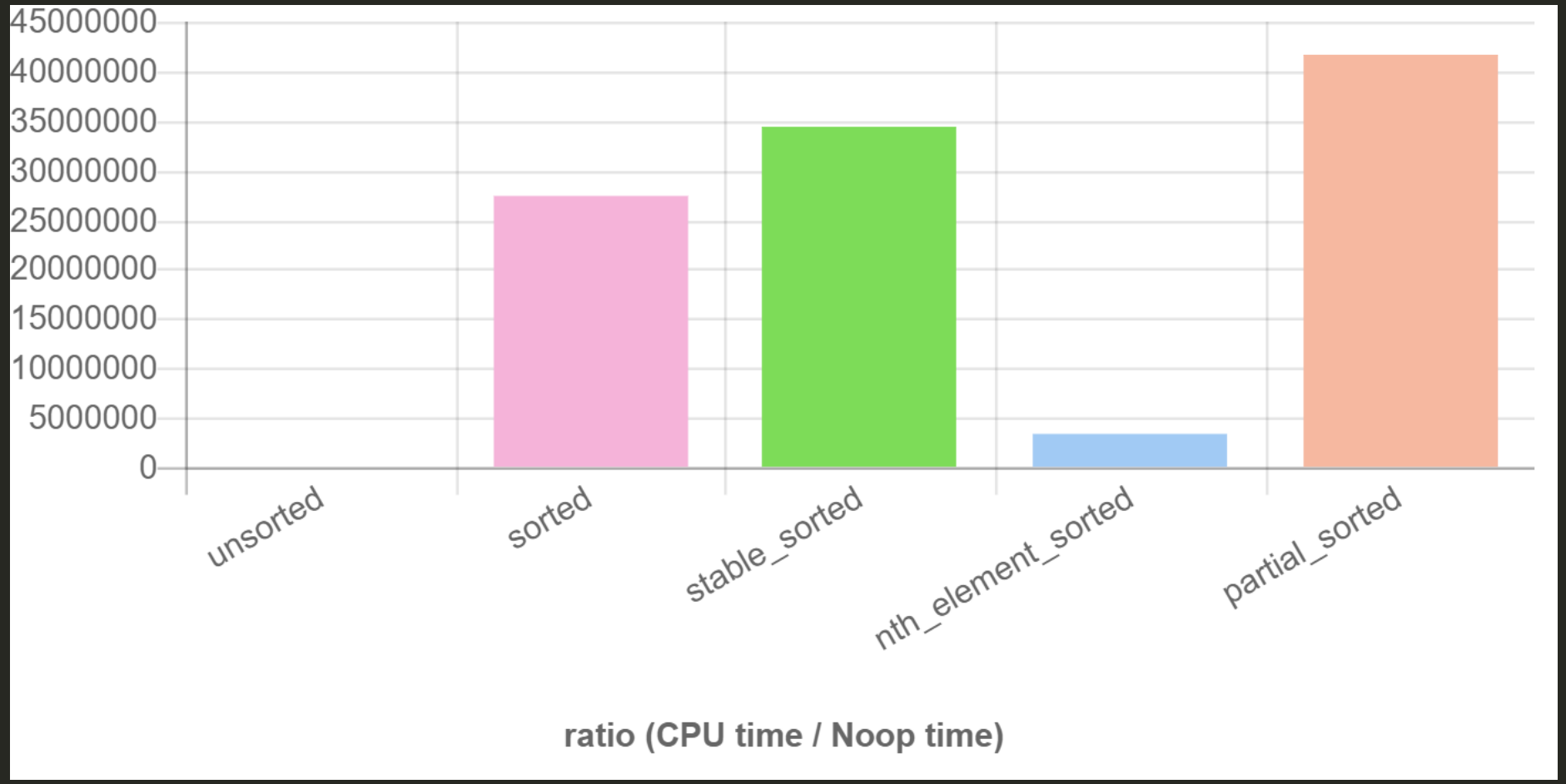
It does a partition around the given position.



nth_element is 10x faster than sort

Partial sort

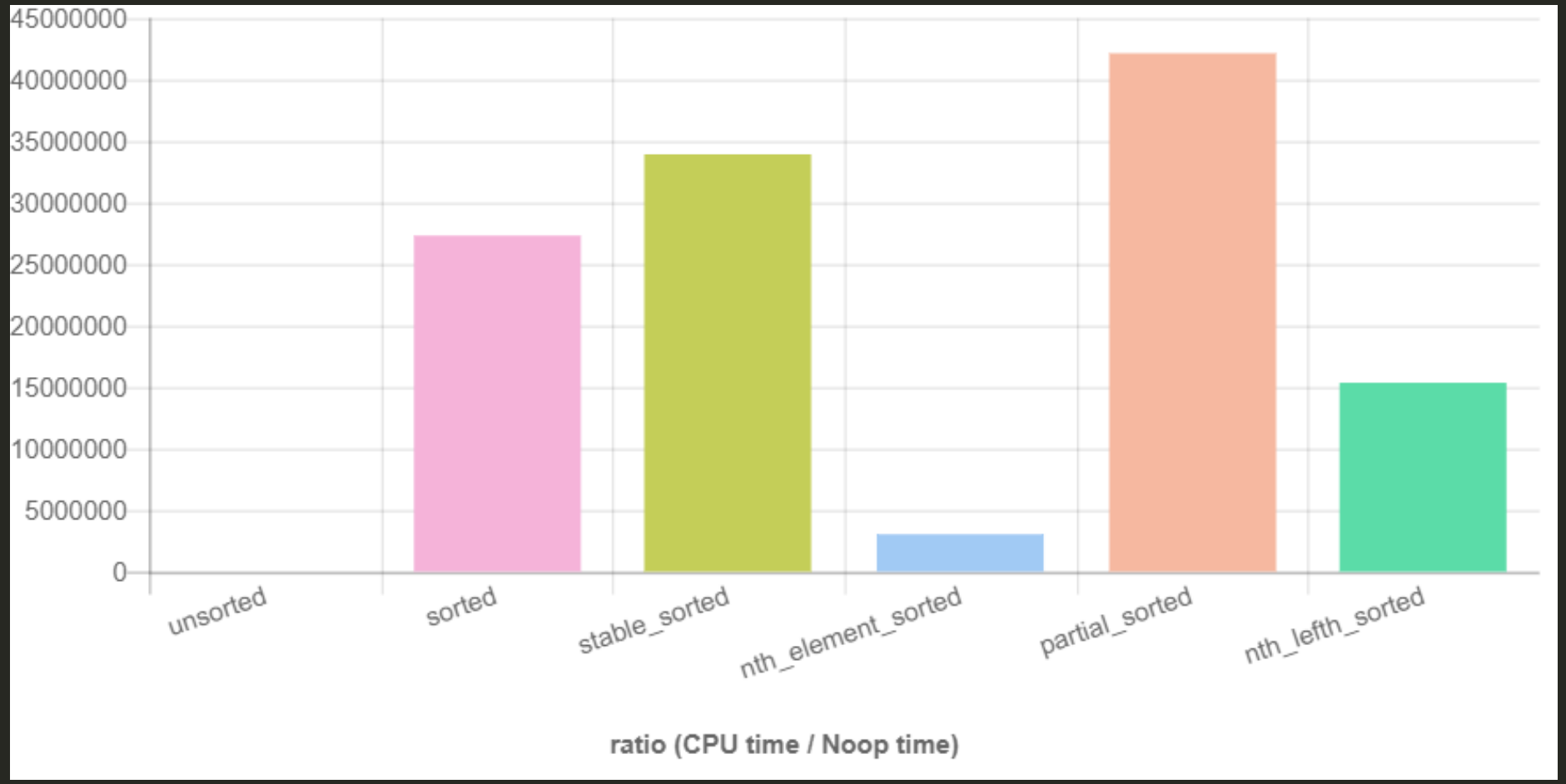
Another way to get the median is to use `std::partial_sort` and sort half of the elements.


45000000
40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
unsorted
sorted
stable_sorted
nth_element_sorted
partial_sorted
ratio (CPU time / Noop time)

Home-cooked partial sort

We can probably do better than this last one.

Let's just std::sort the result of nth_element.


45000000
40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
unsorted
sorted
stable_sorted
nth_element_sorted
partial_sorted
nth_lefth_sorted
ratio (CPU time / Noop time)

Please go on

`quick-bench.com`

and fiddle with it!