Order statistics: from simple to $O(n)$

Banin, M.

January 23, 2018

When in dobut

This does what you think it does in expected $O(n)$.

```
int get_element(const vector<int>& vec, size_t pos){
  assert(pos < vec.size());
  vector<int> myvector(vec.begin(), vec.end());
  std::nth_element (myvector.begin(),
                    myvector.begin() + pos,
                    myvector.end());
  return myvector[pos];
}
```

How to do it by hand

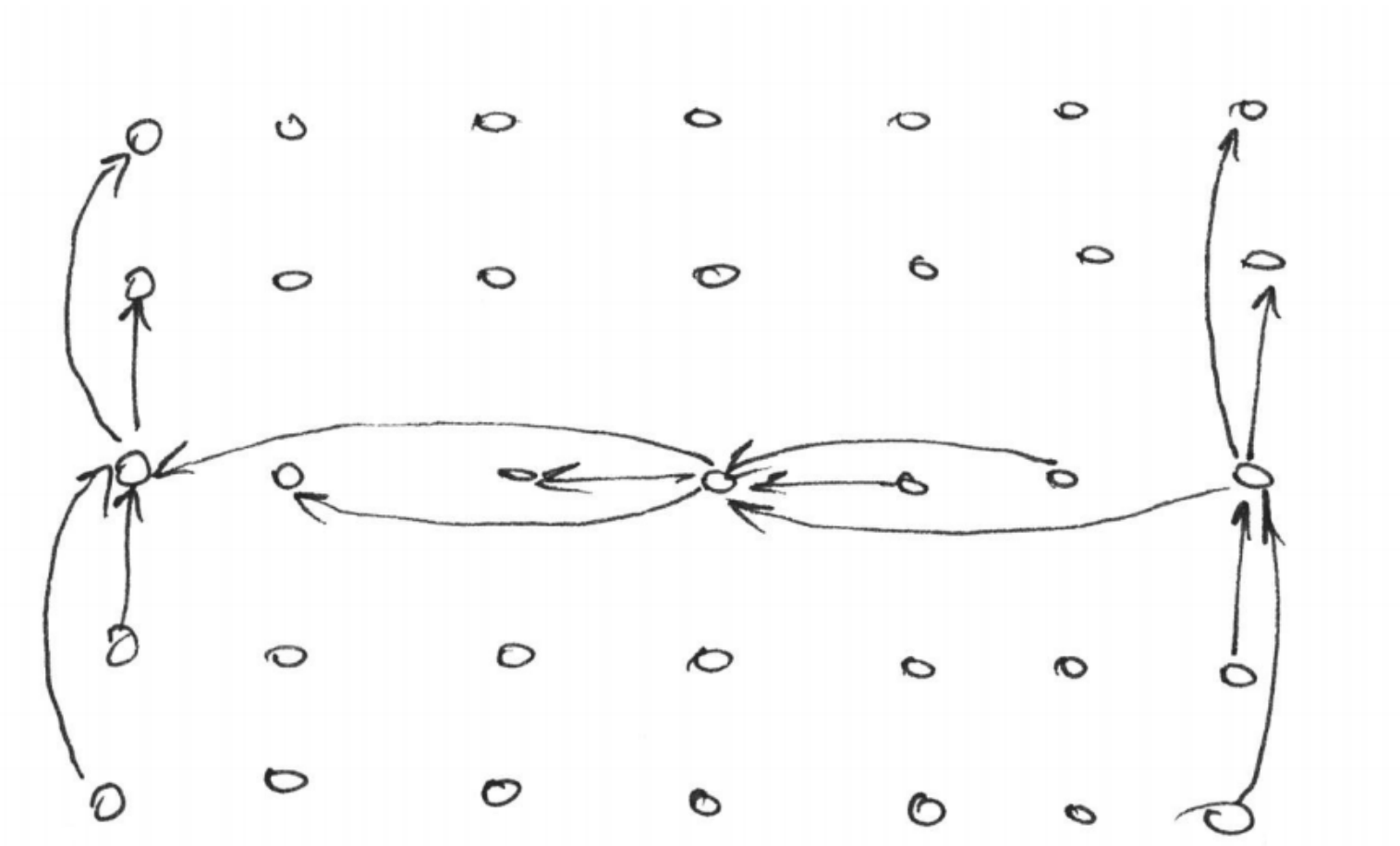
```
template <class Iter, class T>
void my_element(Iter first, Iter nth, Iter last) {
  while (last - first > 3) {
    T pivot = *(first + (last - first) / 2);

    Iter ll = partition(first, last,
      [pivot](const T& a) -> bool{ return a < pivot; });
    Iter rr = partition(ll, last,
      [pivot](const T&a) -> bool{ return a == pivot; });

    if (nth < ll) last = ll;
    else if(nth >= ll && nth < rr) return;
    else if(nth >= rr) first = rr;
  }
  std::sort(first, last);
}
```

What STL does

```
template<typename _RandomAccessIterator, typename _Size, typename _Compare>
  void
  __introselect(_RandomAccessIterator __first, _RandomAccessIterator __nth,
                _RandomAccessIterator __last, _Size __depth_limit,
                _Compare __comp)
{
  while (__last - __first > 3)
      {
        if (__depth_limit == 0)
          {
            std::__heap_select(__first, __nth + 1, __last, __comp);
            // Place the nth largest element in its final position.
            std::iter_swap(__first, __nth);
            return;
          }
        --__depth_limit;
        _RandomAccessIterator __cut =
          std::__unguarded_partition_pivot(__first, __last, __comp);
        if (__cut <= __nth)
          __first = __cut;
        else
          __last = __cut;
      }
    std::__insertion_sort(__first, __last, __comp);
}
```

```
/// This is a helper function...
template<typename _RandomAccessIterator, typename _Compare>
  _RandomAccessIterator
  __unguarded_partition(_RandomAccessIterator __first,
                        _RandomAccessIterator __last,
                        _RandomAccessIterator __pivot, _Compare __comp)
  {
    while (true)
      {
        while (__comp(__first, __pivot))
          ++__first;
        --__last;
        while (__comp(__pivot, __last))
          --__last;
        if (!(__first < __last))
          return __first;
        std::iter_swap(__first, __last);
        ++__first;
      }
  }
```

Median-of medians (Blum, Floyd, Pratt, Rivest, Tarjan)



A bit of compexlity theory

- $\min x = 5^i : x > n = O(n)$
- $\frac{2}{10} + \frac{7}{10} < 1$

Makeshift strict O(n) implementation

```
template <class Iter, class T>
void lintime_element(Iter first, Iter nth, Iter last) {
    while (last - first > 5) {
        std::vector<T> medians;
        for(Iter it = first; it < last; it += 5){
            sort(it, std::min(it + 5, last));
            medians.push_back(*std::min(it + 2, last - 1));
        }

        T padding = *max_element(medians.begin(), medians.end());
        while(medians.size() % 5 != 0)
            medians.push_back(padding);

        Iter mid = medians.begin() + (medians.end() - medians.begin()) / 2;
        lintime_element<typename std::vector<T>::iterator, T>(medians.begin(),
                                                              mid,
                                                              medians.end());

        T pivot = *mid;

        ...
```

Benchmark

On 256K-element arrays, we see:

Implementation	Average runtime
sorting in $O(nlogn)$	52 μsec
makeshift strict $O(n)$	34 μsec
makeshift expected $O(n)$	17 μsec
nth_element call	15 μsec
generating the array	11 μsec

References

- `http://www.cplusplus.com/reference/algorithm/nth_element/`
- `https://en.wikipedia.org/wiki/Quickselect`
- Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. (August 1973). "Time bounds for selection"
- Noriyuki Kurosawa. (2016) Quicksort with median of medians considered practic