Commented Reference Solution to “Quick Select”

Adel Gavranović

```cpp
#include <random>
#include <vector>

  using Iterator = std::vector<int>::iterator;

  Iterator partition(Iterator begin, Iterator end, int pivot)
  {
      auto left = begin;
      auto right = end - 1;
      bool do_left = true;

      while(left <= right)
      {
          while(*left < pivot)
              ++left;
          while(pivot < *right)
              --right;
          if(*left == *right)
          {
              if(left == right || do_left)
                  ++left;
              else
                  --right;
              do_left = !do_left;
          }
          else
              std::swap(*left, *right);
      }
      return left - 1;
  }
```

```
// - - - IGNORE ALL OF THE ABOVE - - -


// Note that this implementation of selection
// uses an iterative approach in contrast to the pseudocode
// that we presented in class.
// Of course, the recursive solution is also valid,
// albeit it can lead to a stack overflow
// if we are unlucky enough to choose a bad pivot too often
// in a very large array.
int selectRandomized(std::vector<int> v, int k)
{
    Iterator begin = v.begin();
    Iterator end = v.end();

    std::mt19937 gen(123);       // create a Random Numbe Generator
    int n = end - begin;         // calculate size of the array (vector, really)

    if(k >= n){                  // because that's impossible
      return 0;
    }

    while(n > 1)                 // "iterative approach"
    {
        std::uniform_int_distribution<int> dis(0, n - 1); // pick a random number from U([0, n-1])

        Iterator pivot = begin + dis(gen);                // pick that element as a pivot

        Iterator pos = partition(begin, end, *pivot);     // call partition function and
                                                          // get iterator pointing to
                                                          // the pivot's position

        int offset = pos - begin;                         // calculate the offset, i.e. number
                                                          // of elements smaller than pivot

        // - - - - the "either look at left or right subarray" part- - - - -
        if (offset < k) {                 // answer is in right subarray,
            begin = pos + 1;              // so change pointers accordingly
            k -= offset + 1;
        } else if (offset > k) {          // answer is in left subarray
            end = pos;                    // so change pointers accordingly
        } else {
            return *pos;                  // offset = k, i.e. we got it!
        }
        // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        n = end - begin;                  // calculate next size `n`
    }

    return *begin;                // = `A[k]`, i.e. the value with rank `k`
}

// - - - IGNORE ALL OF THE BELOW - - -
```

```
int selectBruteForce(std::vector<int> v, int k)
{
    for(auto current = v.begin(); current != v.end(); ++current)
    {
        int less = 0;
        int equal = 0;
        for(auto other = v.begin(); other != v.end(); ++other)
        {
            if(*other < *current)
                ++less;
            else if(*current == *other)
                ++equal;
        }
        if(less <= k && less + equal > k)
            return *current;
    }
    return 0; // not reached
}

// PRE: 0 <= k < std::ssize(v)
// POST: return k-smallest element in vector v
int select(std::vector<int> v, int k){
  return selectRandomized(v,k);
}
```