

LC167. Two Sum II - Input array is sorted

```
class Solution {
public:
    std::vector<int> twoSum(std::vector<int>& numbers, int target) {
        auto l = numbers.cbegin(), r = numbers.cend() - 1;

        while (true) {
            auto sum = *l + *r;
            if (sum < target) ++l;
            else if (sum > target) --r;
            else return {l - numbers.cbegin() + 1, r - numbers.cbegin() + 1};
        }
    }
};
```

LC441. Arranging Coins

```
class Solution {
public:
    int arrangeCoins(int n) {
        return std::sqrt(.125 + n) * std::sqrt(2) - .5;
    }
};
```

LC973. K Closest Points to Origin

```
class Solution {
public:
    std::vector<std::vector<int>> kClosest(std::vector<std::vector<int>>& points, int K) {
        // Preprocessing
        std::random_device rd;
        std::mt19937_64 mt(rd());
        std::shuffle(points.begin(), points.end(), mt);

        auto lt = [](const std::vector<int>& a,
                    const std::vector<int>& b)
            {return a[0]*a[0] + a[1]*a[1] < b[0]*b[0] + b[1]*b[1];};

        nth_point(points.begin(), points.end(), lt, K);

        return std::vector<std::vector<int>>(points.cbegin(), points.cbegin() + K) ;
    }

    template<typename Iterator, typename Comp>
    void nth_point(Iterator lo, Iterator hi, Comp lt, int n) {
        // Base case
        if (hi - lo == 1) return;

        auto i = lo, j = lo + 1, k = hi;

        while (j != k) {
            if (lt(*i, *j)) std::iter_swap(j, --k);
            else if (lt(*j, *i)) std::iter_swap(j++, i++);
            else ++j;
        }

        if (n <= i - lo) nth_point(lo, i, lt, n);
        else if (n > j - lo) nth_point(j, hi, lt, n - (j - lo));
    }
};
```

LC327. Count of Range Sum

```
class Solution {
public:
    int countRangeSum(std::vector<int>& nums, int lower, int upper) {
        // Corner case
        if (nums.empty()) return 0;

        // Compute prefix sums
        std::vector<long> pfs(nums.size(), nums[0]);

        for (std::size_t i = 1; i != nums.size(); ++i)
            pfs[i] = pfs[i - 1] + nums[i];

        auto aux = pfs;

        return mergeCount(aux, pfs, 0, pfs.size(), lower, upper);
    }

    int mergeCount(std::vector<long>& src,
                  std::vector<long>& dst,
                  std::size_t lo, std::size_t hi, int lower, int upper) {
        // hi - lo > 0
        // Base case
        if (hi - lo == 1) return src[lo] < lower || src[lo] > upper ? 0 : 1;

        auto md = lo + (hi - lo) / 2;

        int ct = mergeCount(dst, src, lo, md, lower, upper)
            + mergeCount(dst, src, md, hi, lower, upper);

        ct += count(src, lo, md, hi, lower, upper);

        merge(src, dst, lo, md, hi);

        return ct;
    }

    int count(const std::vector<long>& src,
              std::size_t lo, std::size_t md, std::size_t hi,
              int lower, int upper) {

        int ct = 0;
```

```

    for (auto i = lo, j = lo, k = md; k != hi; ++k) {
        while (i != md && src[k] - src[i] > upper) ++i;

        // Trim
        if (i == md) return ct;
        if (j - i < 0) j = i;

        while (j != md && src[k] - src[j] >= lower) ++j;

        ct += j - i;
    }

    return ct;
}

void merge(const std::vector<long>& src,
           std::vector<long>& dst,
           std::size_t lo, std::size_t md, std::size_t hi) {

    auto i = lo, j = md, k = lo;

    while (i != md && j != hi) {
        if (src[i] <= src[j]) dst[k++] = src[i++];
        else                dst[k++] = src[j++];
    }

    while (i != md) dst[k++] = src[i++];
    while (j != hi) dst[k++] = src[j++];
}
};

```

LC315. Count of Smaller Numbers After Self