LC744 Find Smallest Letter Greater Than Target

```
class Solution {
public:
    /**
     * @param letters: a list of sorted characters
     * @param target: a target letter
     * @return: the smallest element in the list that is larger than the given 
target
     */
    char nextGreatestLetter(string &letters, char target) {
        auto cbeg = std::cbegin(letters), cend = std::cend(letters);

        // binary search
        while (cend - cbeg > 1)
        {
            // caution: infinite loop w/o "-1"
            auto cmid = cbeg + (cend - cbeg) / 2 - 1;

            if (*cmid > target) cend = cmid + 1;
            else cbeg = cmid + 1;
        }

        // base cases
        if (cend - cbeg == 1 && *cbeg > target) return *cbeg;
        else return *std::cbegin(letters);
    }
};
```

LC475 Heaters

```
class Solution {
public:
    int findRadius(vector<int>& houses, vector<int>& heaters) {
        std::sort(heaters.begin(), heaters.end());

        int rad = 0;

        for (auto& house : houses)
        {
            auto cbeg = heaters.cbegin(), cend = heaters.cend();
            int dist = 1000000001;

            while (cbeg != cend)
            {
                auto cmid = cbeg + (cend - cbeg) / 2;

                dist = std::min(dist, std::abs(*cmid - house));

                if      (*cmid > house) cend = cmid;
                else if (*cmid < house) cbeg = cmid + 1;
                else break;
            }

            rad = std::max(rad, dist);
        }

        return rad;
    }
};
```

LC74 Search a 2D Matrix

```cpp
class Solution {
public:
    /**
     * @param matrix: matrix, a list of lists of integers
     * @param target: An integer
     * @return: a boolean, indicate whether matrix contains target
     */
    bool searchMatrix(vector<vector<int>> &matrix, int target) {
        auto begr = std::cbegin(matrix), endr = std::cend(matrix);

        // row search
       while (endr - begr > 1)
        {
            auto midr = begr + (endr - begr) / 2;

            if      (midr->front() > target) endr = midr;
            else if (midr->back()  < target)  begr = midr + 1;
            else
            {
                begr = midr;
                endr = midr + 1;
                break;
            }
        }

        if (endr == begr) return false;

        auto begc = std::cbegin(*begr), endc = std::cend(*begr);

        // column search
        while (endc != begc)
        {
            auto midc = begc + (endc - begc) / 2;

            if      (*midc > target) endc = midc;
            else if (*midc < target) begc = midc + 1;
            else return true;
        }

        return false;

    }
};
```

LC34 Find First and Last Position of Element in Sorted Array

```
class Solution {
public:
    /**
     * @param nums: the array of integers
     * @param target:
     * @return: the starting and ending position
     */
    vector<int> searchRange(vector<int> &nums, int target) {
        // Write your code here.
        auto cbeg = std::cbegin(nums), cend = std::cend(nums);

        // search the rightmost val
        while (cend - cbeg > 1)
        {
            auto cmid = cbeg + (cend - cbeg) / 2;

            if (*cmid <= target) cbeg = cmid;
            else cend = cmid;
        }

        // if val not found
        if (cbeg == cend || *cbeg != target) return {-1, -1};

        auto right = cbeg, left = cbeg;
        cbeg = std::cbegin(nums);

        // search for the left most val
        // left points to the leftmost val found so far
        while (left != cbeg)
        {
            auto cmid = cbeg + (left - cbeg) / 2;

            if (*cmid < target) cbeg = cmid + 1;
            else left = cmid;
        }

       return {left -  std::cbegin(nums), right - std::cbegin(nums)};

    }
};
```

LC719 Find K-th Smallest Pair Distance

```
class Solution {
public:
    int smallestDistancePair(vector<int>& nums, int k) {
        std::sort(nums.begin(), nums.end());

        int lo = 0, hi = nums.back() - nums.front();


        while (lo != hi){
            int md = (lo + hi) / 2;
            int count = countLessEqualPairs(nums, md);

            // shorten the search range
            // and update hi to the new possible dist
            if (count < k) lo = md + 1;
            else           hi = md;
        }

        return hi;
    }

    int countLessEqualPairs(std::vector<int>& nums, int dist) {
        int count = 0;

        for (auto i = nums.cbegin(), j = i + 1; i != nums.cend(); ++i){
            while (j != nums.cend() && *j - *i <= dist) ++j;
            count += j - i - 1;
        }

        return count;
    }
};
```