Lab 1

1 实验目的

1. 掌握分治算法的设计思想与方法，
2. 熟练使用高级编程语言实现分治算法，
3. 通过对比简单算法以及不同的分治求解思想，理解算法复杂度。

2 实验问题

求解凸包问题:输入是平面上 n 个点的集合 Q，凸包问题是要输出一个 Q 的 凸包。其中，Q 的凸包是一个凸多边形 P，Q 中的点或者在 P 上或者在 P 中。

3 实验步骤

3.1 实现基于枚举方法的凸包求解算法

主要思想

遍历点集中任意4个点，然后分别判断这四个点中，某一个点是否在另外三个点组成的三角形内。如果在，那么这个点必然不是凸包上的点。遍历结束后，没有排除点集即为凸包。时间复杂度是$T(n) = C_n^4 = O(n^4)$。

```
for (size_t i = 0; i < n; ++i) {
    for (size_t j = i+1; j < n; ++j) {
        for (size_t k = j+1; k < n; ++k) {
            for (size_t l = k+1; l < n; ++l) {
                size_t counters[4] = {i, j, k, l};
                for (int m = 0; m < 4; ++m) {
                    if (indexes.find(counters[m]) == indexes.end()) {
                        Point& p = points[counters[m]];
                        Point& a = points[counters[(m + 1) % 4]];
                        Point& b = points[counters[(m + 2) % 4]];
                        Point& c = points[counters[(m + 3) % 4]];
                        if (CheckPointInTriangle(p, a, b, c)) {
                            indexes.insert(counters[m]);
                        }
                    }
                }
            }
        }
    }
}
```

判断一个点是否在三角形内

对于三角形ABC和一点P，可以有如下的向量表示

$$\overrightarrow{AP} = u\overrightarrow{AB} + v\overrightarrow{AC}$$

那么P在三角形内的充要条件是

$$0 < u < 1, \quad 0 < v < 1, \quad u + v < 1$$

可以根据点的坐标，计算出u和v的值，便可判断P是否在三角形ABC内，代码如下

```
bool CheckPointInTriangle(const Point& p, const Point& a, const Point& b, const Point&
c) const {
    Point ap{p.first - a.first, p.second - a.second};
    Point ab{b.first - a.first, b.second - a.second};
    Point ac{c.first - a.first, c.second - a.second};

    double u = (ap.first * ac.second - ac.first * ap.second) / (ab.first * ac.second -
ac.first * ab.second);
    double v = (ap.first * ab.second - ab.first * ap.second) / (ac.first * ab.second -
ab.first * ac.second);
    return u > 0 && v > 0 && u + v < 1;
}
```

3.2 实现基于 Graham-Scan 的凸包求解算法

主要思想

点集中最下方的点一定是凸包上的点，那么可以以该点为参照点，其他点按照参照点逆时针排序。然后按照这个顺序依次遍历，并在此期间维护一个栈，遇到新点时，判断其与栈顶以及次栈顶组成向量的转向。左转时将点入栈，右转时出栈。时间复杂度$T(n) = O(nlogn)$

```
// 1. find lowest point p0
auto lowest_point_iter =
    min_element(points.begin(), points.end(), [](const Point& a, const Point& b) {
return a.second < b.second; });
iter_swap(points.begin(), lowest_point_iter);

// 2. sort by angle p0 to pi
sort(next(points.begin()), points.end(), [&](const Point& a, const Point& b) { return
isRight(a, points[0], b); });

vector<Point> convex_hull{points[0], points[1], points[2]};
size_t n = points.size();

for (size_t i = 3; i < n; ++i) {
    while (isRight(points[i], convex_hull[convex_hull.size() - 2], convex_hull.back()))
{
        convex_hull.pop_back();
    }
    convex_hull.push_back(points[i]);
}
```

判断右（左）转

可以根据向量外积的正负判断一个点在一个向量的左侧或右侧， 也可以等价为三点的顺序是逆时针或顺时针。

```
// check if P is at vector AB right side(i.e. A->B->P is clockwise)
inline bool isRight(const Point& p, const Point& a, const Point& b) {
  return (b.first - a.first) * (p.second - a.second) - (b.second - a.second) * (p.first
- a.first) < 0;
};
```

3.3 实现基于分治思想的凸包求解算法

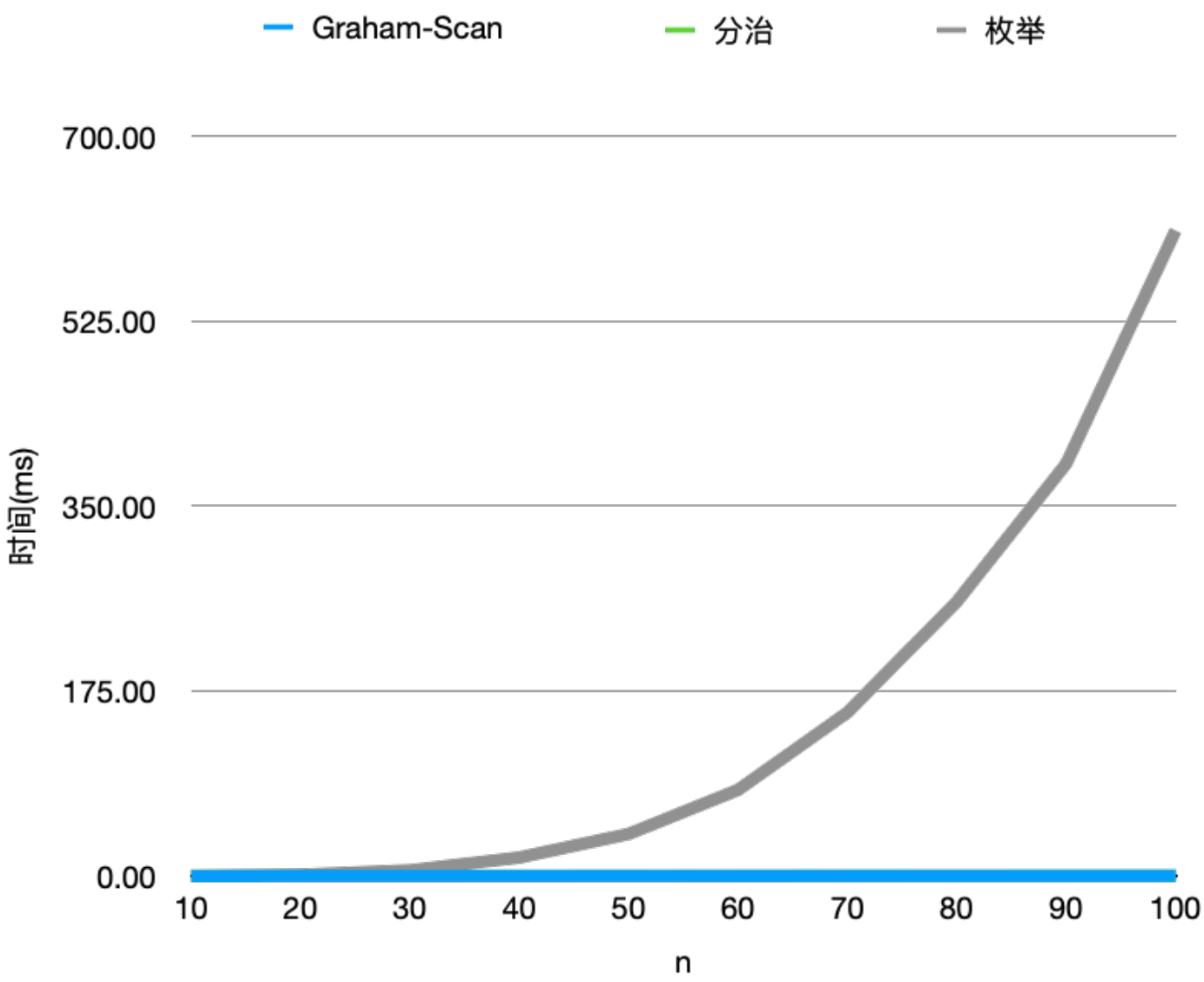
基本思想

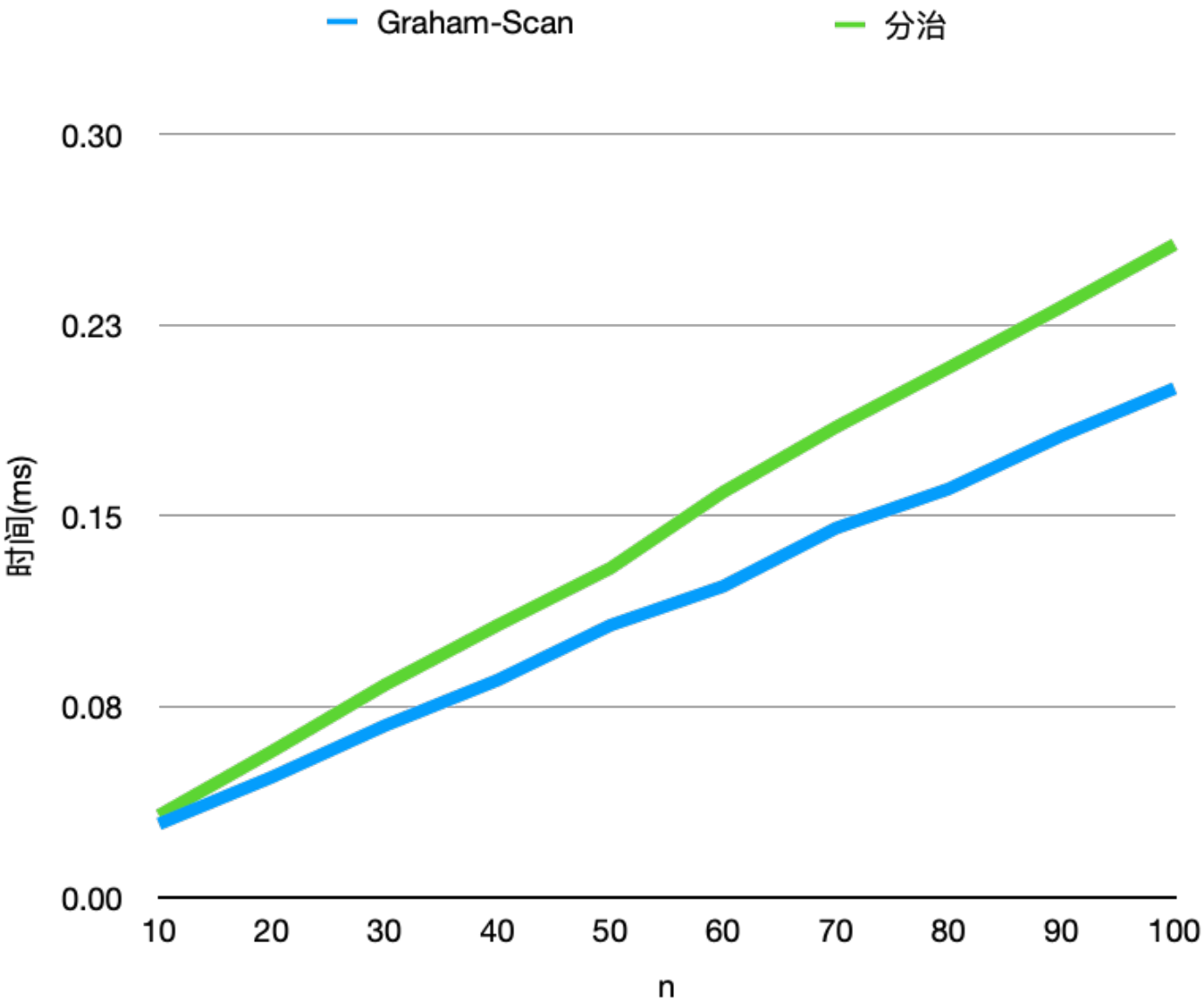
显示递归地将问题划分成最小的问题（三个及以下的点）。划分时，选择一条中线，将平面平分成左右两半。对于小于三个点的情形，直接返回本身，对于三个点逆时针排序。在划分结束后，开始合并，即将两个小的凸包的点按照逆时针顺序merge，然后使用Graham-Scan合并成大的凸包。时间复杂度$T(n) = O(nlogn)$

```
vector<Point> Divide(vector<Point>::iterator left, vector<Point>::iterator right) {
    if (right - left < 3) {
        return vector<Point>(left, right);
    }

    if (right - left == 3) {
        // Sort counterclockwise
    }

    auto m = left + (right - left) / 2;
    std::nth_element(left, m, right, [](const Point& a, const Point& b) { return
a.first < b.first; });

    vector<Point> left_ch = Divide(left, m);
    vector<Point> right_ch = Divide(m, right);

    return Merge(left_ch, right_ch);
}
```

3.4 对比三种凸包求解算法

使用google benchmark库测试性能，性能曲线如下。和理论分析一致，枚举法时间最多。而GS和分治法时间差不多，GS更优。原因可能是stl库函数sort的性能对于随机生成的数据，性能要优于分治法（可以看作归并排序）。


Graham-Scan
分治
枚举
700.00
525.00
350.00
175.00
0.00
时间(ms)
10
20
30
40
50
60
70
80
90
100
n


Graham-Scan
分治
0.30
0.23
0.15
0.08
0.00
时间(ms)
10
20
30
40
50
60
70
80
90
100
n