Санкт-Петербургский политехнический университет Петра великого
Институт машиностроения, материалов и транспорта

Курсовая работа

По дисциплине: «Объектно-ориентрованное программирование»

Выполнил:
студент гр. 3331506/90401 Козюра К.К.

Преподаватель Ананьевский М.С.

«___»_____2022 г.

Санкт-Петербург
2022 г.

Оглавление

1. ВВЕДЕНИЕ 3
2. ИДЕЯ АЛГОРИТМА 4
2.1. Задача поиска минимальной выпуклой оболочки 4
2.2. Алгоритм Джарвиса 4
2.3. Анализ метода 8
3. ИССЛЕДОВАНИЕ АЛГОРИТМА 9
3.1. Методика исследования 9
3.2. Результаты исследований 10
3.2.1. Равномерное распределение 10
3.2.2. Нормальное распределение 11
3.3. Анализ результатов 12
4. ЗАКЛЮЧЕНИЕ 13
СПИСОК ЛИТЕРАТУРЫ 14
ПРИЛОЖЕНИЯ 15

1. ВВЕДЕНИЕ

Задача поиска выпуклых оболочек является одной из базовых задач вычислительной геометрии. Она имеет большое количество практических приложений, связанных с задачами распознавания образов и кластеризации. Несмотря на свою простоту, данная задача имеет различные решения, являющиеся оптимальными в той или иной ситуации.

Одним из методов решения данной задачи является алгоритм Джарвиса или алгоритм заворачивания подарка [1], который рассмотрен в данной работе.

В главе 2 описана задача, которую решает алгоритм, его идея и конкретная реализация.

В главе 3 обозреваются проведенные исследования алгоритма.

2. ИДЕЯ АЛГОРИТМА

2.1. Задача поиска минимальной выпуклой оболочки

Выпуклой оболочкой (ВО) множества X называется наименьшее выпуклое множество, содержащее X. «Наименьшее множество» здесь означает наименьший элемент по отношению к вложению множеств, то есть такое выпуклое множество, содержащее данную фигуру, что оно содержится в любом другом выпуклом множестве, содержащем данную фигуру.

Главной особенностью ВО множества точек X является то, что эта оболочка представляет собой выпуклый многоугольник, вершинами которого являются некоторые точки из X. Поэтому задача поиска ВО сводится к отбору и упорядочиванию нужных точек из X.

Пример выпуклой оболочки приведен на рисунке 2.1.

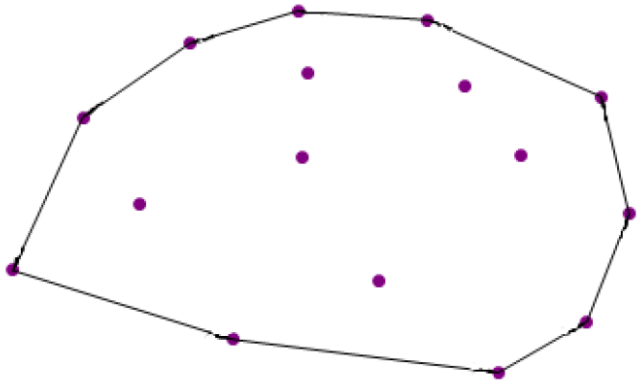

Рисунок 2.1 – Выпуклая оболочка.

2.2. Алгоритм Джарвиса

Алгоритм Джарвиса определяет последовательность элементов множества, образующих выпуклую оболочку для этого множества. Метод можно представить как обтягивание верёвкой множества вбитых в доску гвоздей.

Пусть дано множество точек $P = \{p_1, p_2, \dots, p_n\}$. В качестве начальной берётся самая левая нижняя точка p_1(её можно найти за $O(n)$ обычным проходом по всем точкам), она точно является вершиной выпуклой оболочки. Следующей точкой p_2 берём такую точку, которая имеет наименьший положительный

полярный угол относительно точки p_1 как начала координат. После этого для каждой точки p_i (2 < i <= $|P|$) против часовой стрелки ищется такая точка p_{i+1}, путём нахождения за $O(n)$ среди оставшихся точек (+ самая левая нижняя), в которой будет образовываться наибольший угол между прямыми $p_{i-1}p_i$ и p_ip_{i+1}. Она и будет следующей вершиной выпуклой оболочки. Сам угол при этом не ищется, а ищется только его косинус через скалярное произведение между лучами $p_{i-1}p_i$ и $p_ip'_{i+1}$, где p_i — последний найденный минимум, p_{i-1} — предыдущий минимум, а p'_{i+1} — претендент на следующий минимум. Новым минимумом будет та точка, в которой косинус будет принимать наименьшее. Нахождение вершин выпуклой оболочки продолжается до тех пор, пока $p_{i+1} \neq p_1$. В тот момент, когда следующая точка в выпуклой оболочке совпала с первой, алгоритм останавливается — выпуклая оболочка построена.

Данный алгоритм, представленный в виде псевдокода:

```
Jarvis(P)
    1) p[1] = самая левая нижняя точка множества P;
    2) p[2] = соседняя точка от p[1] справа (находится через минимальный положительный полярный угол)
    3) i = 2;
    4) do:
            (a)for для каждой точки j от 1 до |P|, кроме уже попавших в выпуклую оболочку, но включая p[1]
                p[i+1] = point_with_min_cos(p[i-1], p[i], P[j]); //точка, образующая минимальный косинус с прямой p[i-1]p[i],
            (b)i = i + 1;
       while p[i] != p[1]
    5) return p;
```

На рисунке 2.2 приведены несколько шагов, наглядно иллюстрирующих построение выпуклой оболочки.

На рисунке 2.3 приведен обход Джарвиса, на каждом шаге мы получаем одну вершину.

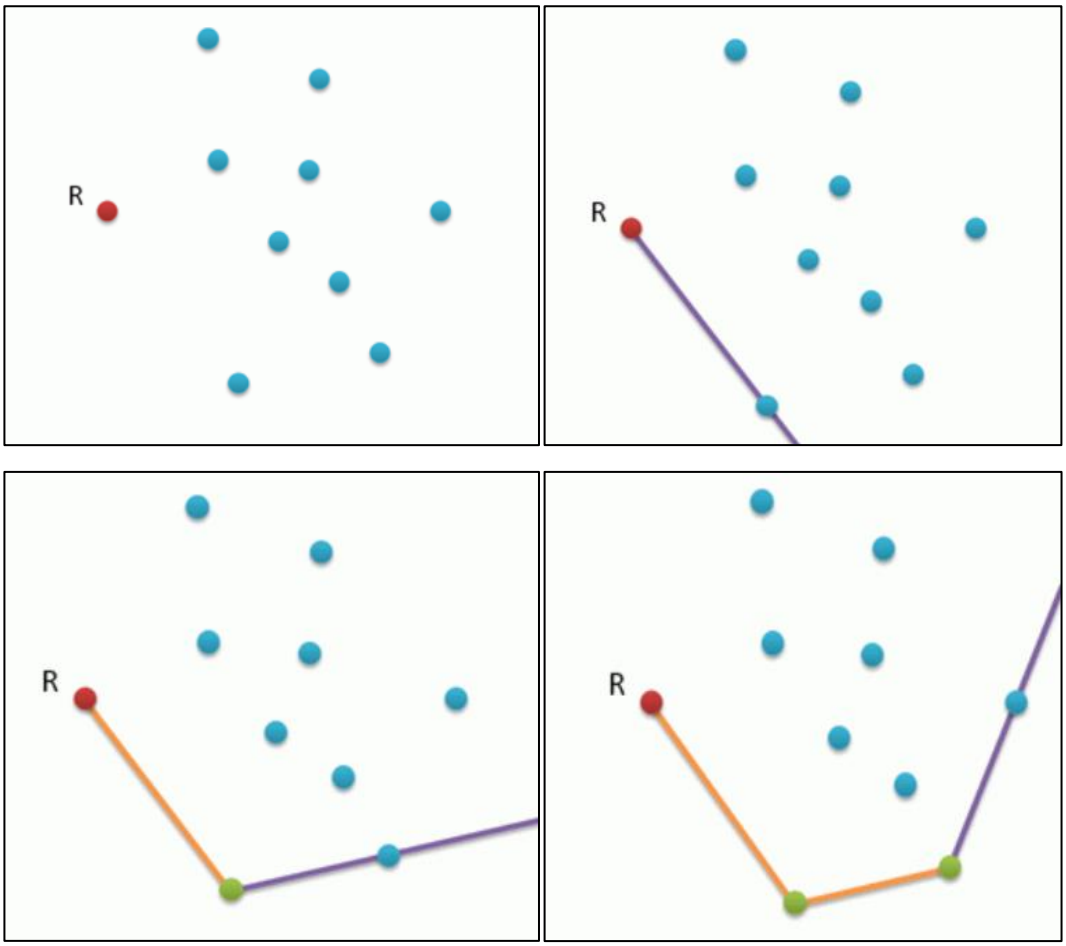

Рисунок 2.2 – Несколько шагов построения выпуклой оболочки

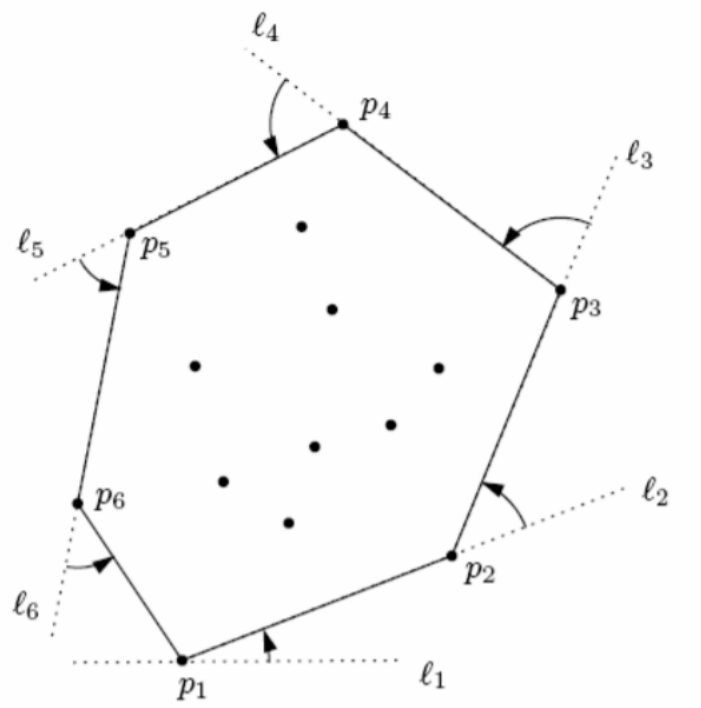

Рисунок 2.3 – Построение выпуклой оболочки методом Джарвиса

Ниже приведен код алгоритма Джарвиса на языке C++. Полный код, используемый в работе приведен в приложении А.

```
void ConvexHullJarvis(const std::vector<point> &mass, std::vector<int>
&convex_hul, int count) {

    if (count < 1) {
        return;
    }
    int base = 0;
    for (int i=1;i<count;i++) {
        if ((mass[i].y < mass[base].y) || (mass[i].y == mass[base].y &&
mass[i].x < mass[base].x)) {
            base = i;
        }
    }
    convex_hul.push_back(base);

    int first = base;
    int cur = base;
    do {
        int next = (cur + 1) % count;
        for (int i=0;i<count;i++) {
            double sign = OrientTriangl2(mass[cur], mass[next], mass[i]);
            if (sign < 0)
                next = i;
            else if (sign == 0) {
                if (isInside(mass[cur],mass[next],mass[i]))
                    next = i;
            }
        }
        cur = next;
        convex_hul.push_back(next);
    }
    while (cur != first);

}
```

2.3. Анализ метода

Оценим сложность алгоритма Джарвиса. Первый шаг имеет сложность $O(n)$. Второй шаг содержит вложенный цикл, число внешних итераций равно числу вершин h в ВО, число внутренних итераций не превышает n. Следовательно, сложность всего алгоритма равна $O(hn)$. Необычным в этой формуле является то, что сложность определяется не только длиной входных данных, но и длиной выхода (output-sensitive algorithm). В худшем случае все точки из А входят в ВО, тогда h=n и сложность равна $O(n^2)$. В лучшем случае (при условии, что точки из А не лежат на одной прямой) h=3 и сложность становится $O(n)$.

Достоинства:

• Алгоритм Джарвиса может быть применен в пространствах размерности больше двух.

• При априорном знании малости числа вершин оболочки в теории даёт оптимальную трудоемкость.

Недостатки:

• Высокая квадратичная трудоемкость в худшем случае.

3. ИССЛЕДОВАНИЕ АЛГОРИТМА

3.1. Методика исследования

Для количественного анализа описанного алгоритма построения выпуклых оболочек на плоскости было проведено моделирование работы алгоритма построения выпуклых оболочек на различных распределениях точек. При этом анализировалась скорость работы алгоритмов.

Для сравнения алгоритмов в различных условиях в экспериментах строились наборы данных, располагавшимися в единичном квадрате $[0,1]^2$ (или близко к этому) со следующими характеристиками:

1. Равномерное распределение. Всех точки были распределены равномерно и независимо в единичном интервале.
2. Нормальное распределение. Все точки размещались по нормальному распределению с центром в точке (0,5; 0,5) и среднеквадратическим отклонением 0,1.

Для каждого числа точек эксперимент повторялся 10 раз, графики построены по медианным значениям полученных наборов данных.

Значения, использованные для построения графиков приведены в приложении Б.

3.2. Результаты исследований

3.2.1. Равномерное распределение

На рисунке 3.1 изображено равномерное распределение точек на единичной плоскости.

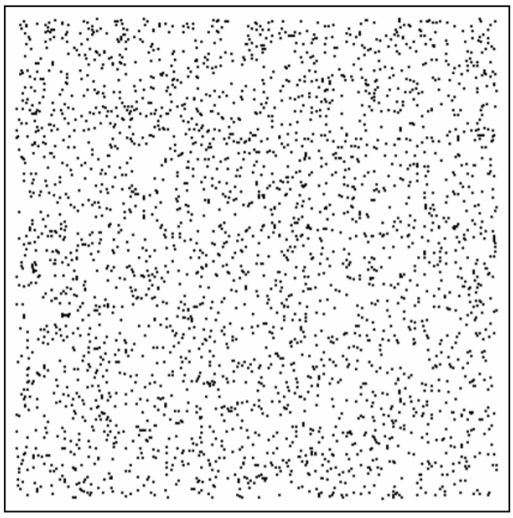

Рисунок 3.1 – Равномерное распределение

На рисунке 3.2 изображен график медианного времени выполнения алгоритма от количества точек в случае их равномерного распределения.

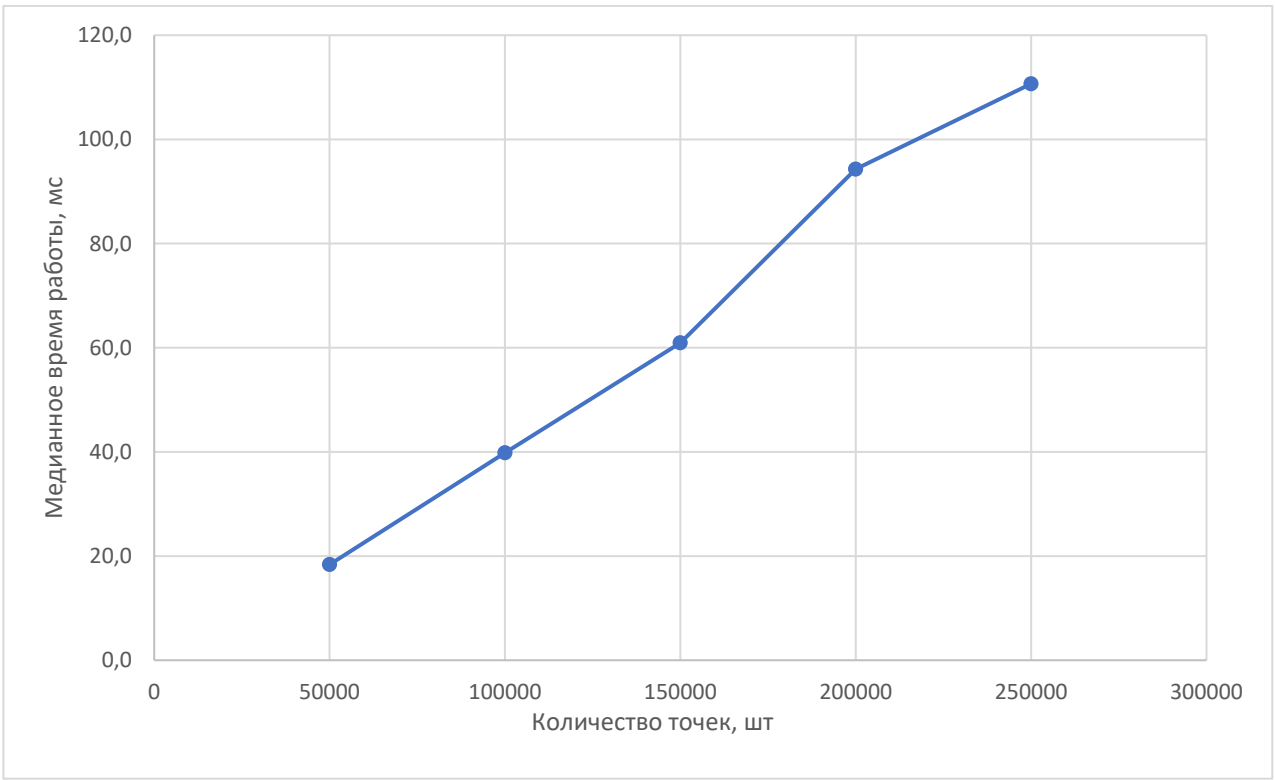

Рисунок 3.2 – График медианного времени

3.2.2. Нормальное распределение

На рисунке 3.3 изображено нормальное распределение точек на единичной плоскости.

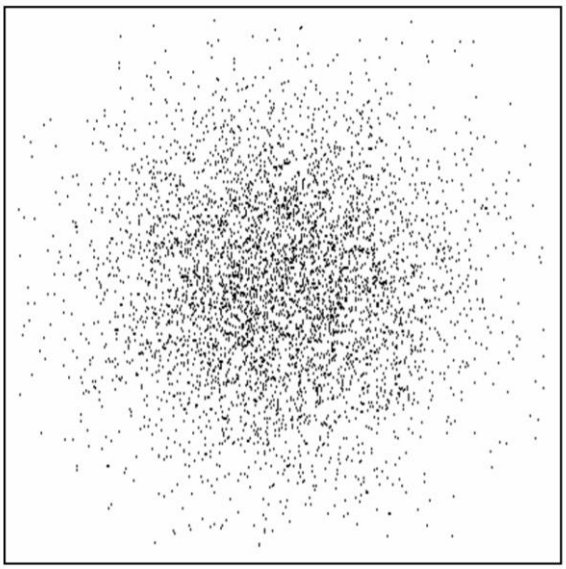

Рисунок 3.3 – Нормальное распределение

На рисунке 3.4 изображен график медианного времени выполнения алгоритма от количества точек в случае их нормального распределения.

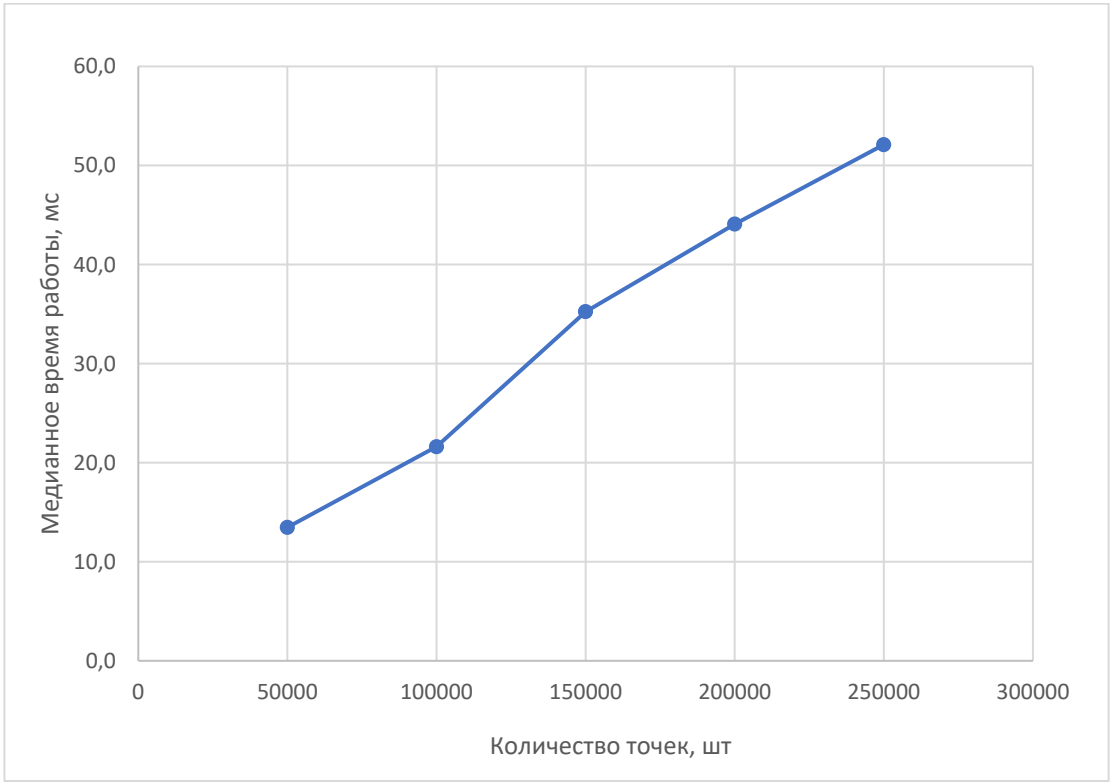

Рисунок 3.4 – График медианного времени

3.3. Анализ результатов

Совмещенный график для двух наборов данных представлен на рисунке 3.5.

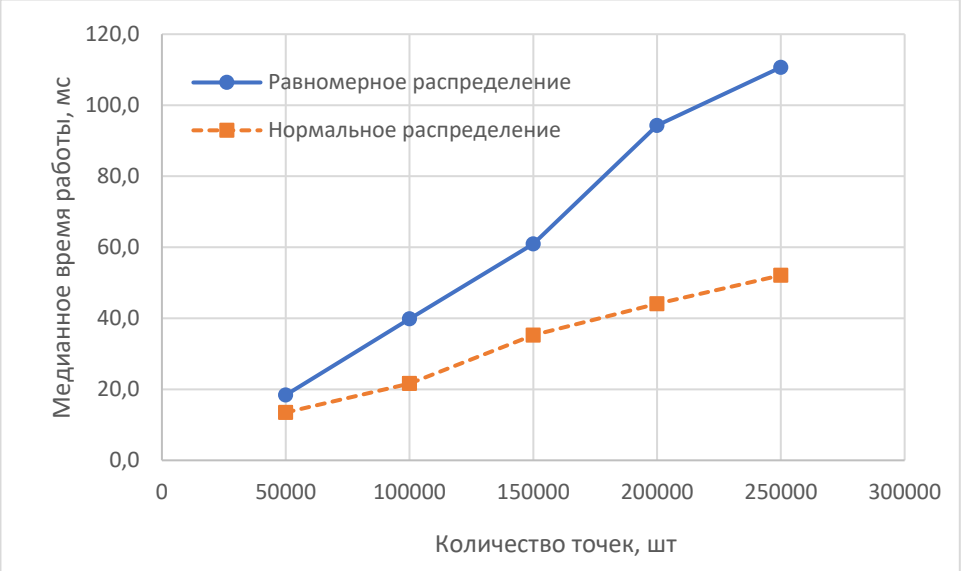

Рисунок 3.5 – Совмещенный график

Как можно видеть из рисунка 3.5, алгоритм Джарвиса быстрее работает с набором точек, расположенных в соответствии с нормальным распределением, что объясняется меньшим количеством точек, образующих выпуклую оболочку. Это наблюдение подтверждает результаты анализа метода в п. 2.3.

Так же стоит отметить отсутствие квадратичного роста времени при увеличении числа точек в приведенных экспериментах. Для наблюдения этого эффекта требуется создание специального массива данных, в котором точки, например, будут расположены по окружности.

4. ЗАКЛЮЧЕНИЕ

В работе был рассмотрен алгоритм Джарвиса, позволяющий находить выпуклую оболочку множества.

Анализ алгоритма и экспериментальные исследования показали неэффективность данного метода в случае, когда элементы расположены на границе области. Однако же, в случае, когда априорно известно небольшое число элементов, составляющих выпуклую оболочку, данный метод способен обеспечить линейную сложность по времени.

Стоит отметить, что этот метод прост в реализации и не требует дополнительной подготовки входных данных.

СПИСОК ЛИТЕРАТУРЫ

1. Jarvis A. On the identification of the convex hull of a finite set of points in the plane. // Information Processing Letters. – 1973. – Vol. 2. – pp.18–21.
2. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение / Пер. с англ. – М.: Мир, 1989. – 478 с.

ПРИЛОЖЕНИЕ А

Полный код, используемый для исследования алгоритма

```
#include <iostream>
#include <vector>
#include <chrono>
#include <random>
struct point { //Структура для хранения координат
    double x,y;
    point() {
        x = 0;
        y = 0;
    }
    point(double X, double Y) {
        x = X;
        y = Y;
    }
};

bool operator != (const point &a, const point &b) {
    return !(a.x == b.x && a.y == b.y);
}

int numb;
std::vector<point> mas;
std::vector<int> convex_hull;

std::default_random_engine generator;
std::normal_distribution<double> distribution(0.5,0.1);
//std::uniform_real_distribution<double> distribution(0,1);


void input(int count) {
    numb = count;
    if (numb > 0) {
        mas.resize(numb);
    }
    for (int i=0;i<numb;i++) {
        mas[i].x = distribution(generator);
        mas[i].y = distribution(generator);
    }
}
double OrientTriangl2(const point &p1,const point &p2, const point &p3) {
    return p1.x * (p2.y - p3.y) + p2.x * (p3.y - p1.y) + p3.x * (p1.y -
p2.y);
}

bool isInside(const point &p1, const point &p, const point &p2) {
    return ( p1.x <= p.x && p.x <= p2.x &&
             p1.y <= p.y && p.y <= p2.y);
}
void ConvexHullJarvis(const std::vector<point> &mass, std::vector<int>
&convex_hul, int count) {
    auto start_time = std::chrono::steady_clock::now();
    // находим самую левую из самых нижних
    if (count < 1) {
```

```
        return;
    }
    int base = 0;
    for (int i=1;i<count;i++) {
        if ((mass[i].y < mass[base].y) || (mass[i].y == mass[base].y &&
mass[i].x < mass[base].x)) {
            base = i;
        }
    }
    convex_hul.push_back(base);

    int first = base;
    int cur = base;
    do {
        int next = (cur + 1) % count;
        for (int i=0;i<count;i++) {
            double sign = OrientTriangl2(mass[cur], mass[next], mass[i]);
            if (sign < 0)
                next = i;
            else if (sign == 0) {
                       if (isInside(mass[cur],mass[next],mass[i]))
                    next = i;
            }
        }
        cur = next;
        convex_hul.push_back(next);
    }
    while (cur != first);
    auto end_time = std::chrono::steady_clock::now();
    auto elapsed_ns =
std::chrono::duration_cast<std::chrono::nanoseconds>(end_time -
start_time);
    std::cout << elapsed_ns.count() << "\n";
}

int main() {

    for (int k = 0; k < 10; ++k) {
        input(250000);
        ConvexHullJarvis(mas,convex_hull, numb);
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

Таблица Б.1 –Равномерное распределение

Равномерное распред.		Наборы точек, шт				
	№, п.п.	50000	100000	150000	200000	250000
Время выполнения, нс	1	18037300	38577800	88570500	145780300	112893200
	2	21695400	41175500	70689500	120443900	112756200
	3	26289900	37954300	69731500	101174600	135745800
	4	88271000	48792000	60602200	103896500	99442100
	5	19933500	33496800	57253500	86511300	117738500
	6	16362000	41386600	58278100	86847000	114352500
	7	15851300	34279600	50331900	93598200	105515100
	8	15111000	33134300	60976900	94988400	94286600
	9	18717900	41061800	60901100	81674600	87729100
	10	13225000	44417400	64745200	80487900	108581300
Медианное время, мс		18,4	39,8	60,9	94,3	110,7

Таблица Б.2 – Нормальное распределение

Нормальное распред.		Наборы точек, шт				
	№, п.п.	50000	100000	150000	200000	250000
Время выполнения, нс	1	22315400	42879800	66185800	59444200	121873300
	2	14598900	31489000	35044500	38798500	46625000
	3	19055200	24226900	35439300	42877100	44655200
	4	12344200	21424700	28627000	35838500	39933600
	5	15678100	18559800	25529900	31335200	52393600
	6	15882500	21804900	33037800	57725900	53555000
	7	8019200	27604200	30062500	46984600	51813800
	8	9061700	20574500	36967800	45283100	65198600
	9	10564400	20512500	45742100	40572300	49250300
	10	8935500	16843400	40808900	48607900	69444800
Медианное время, мс		13,5	21,6	35,2	44,1	52,1