Комбинаторные алгоритмы вычислительной геометрии

Рандомизированный алгоритм построения выпуклой оболочки

Рандомизированный алгоритм построения выпуклой оболочки

Рандомизированные геометрические алгоритмы

Рандомизированная пошаговая конструкция:

n объектов, составляющих входные данные к задаче, рассматриваются *по одному* в каждый момент времени *в произвольном порядке*, и вычисляется результат воздействия каждого добавленного объекта на решение задачи.

Для многих геометрических задач этот подход похож на другие известные алгоритмы, за исключением того, что в них обычно объекты обрабатываются в порядке,

существующем на входе, а не в произвольном порядке.

РАНДОМИЗИРОВАННЫЕ АЛГОРИТМЫ В ЗАДАЧАХ ВЫЧИСЛИТЕЛЬНОЙ ГЕОМЕТРИИ

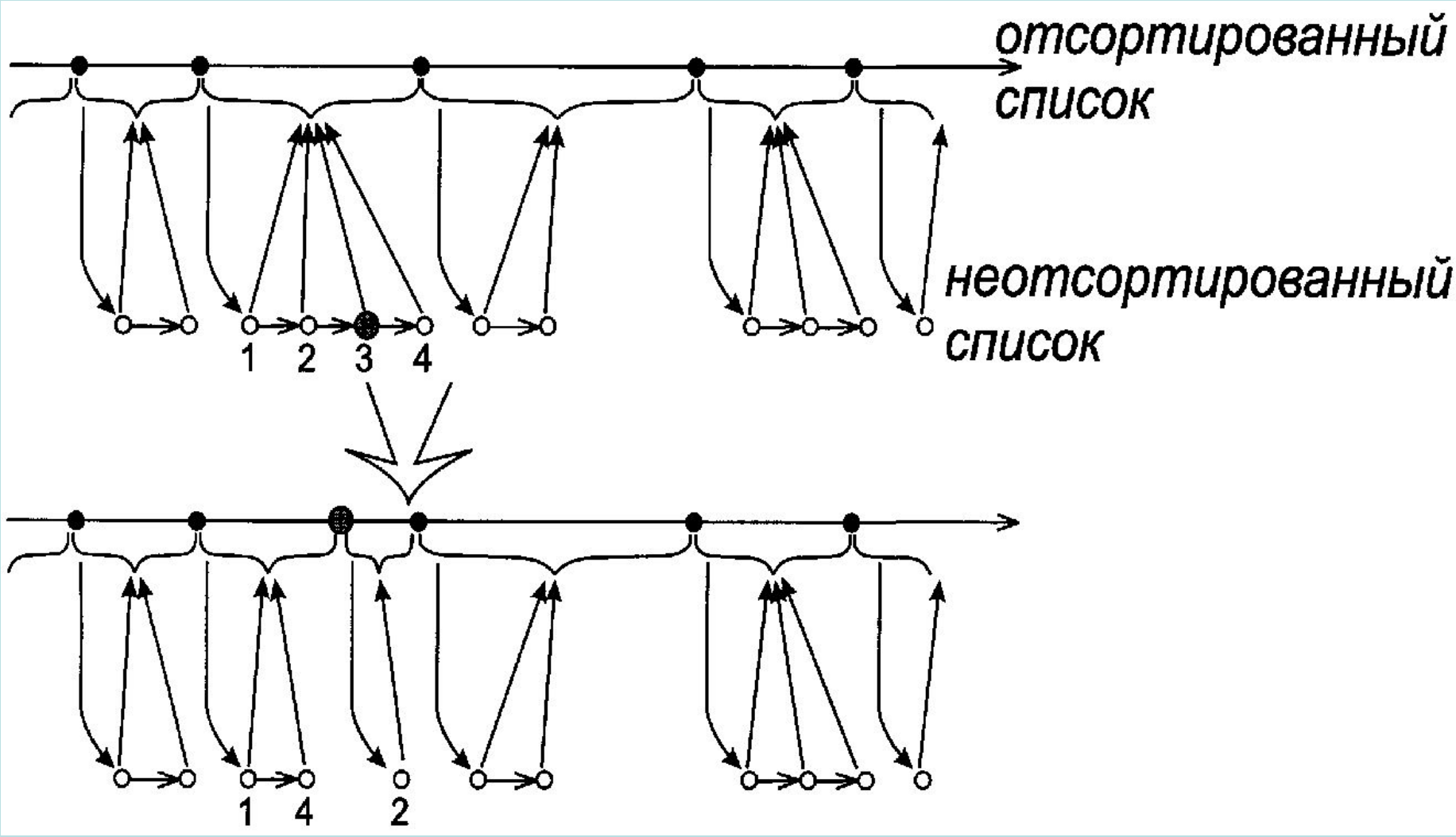
Рандомизированная пошаговая сортировка

Дано n чисел, которые нужно отсортировать.

Схема сортировки:

- *После i-ого из n шагов $(1 < i < n)$ имеем i вставленных чисел в отсортированном списке.*
- *Эти i отсортированных чисел разобьют ранги $(n-i)$ еще не отсортированных чисел на $(i+1)$ интервалов.*
- *$(i+1)$-ый шаг состоит в выборе одного из $(n-i)$ еще не отсортированных чисел случайным образом и вставки его в отсортированный список.*

Рандомизированная пошаговая сортировка



Рандомизированная пошаговая сортировка

Какая работа требуется, чтобы сохранить указатели на каждое число?

Мы вставляем число x, указатель которого указывает на интервал I.

При вставке x, мы имеем три задачи:

(1) найти все числа, указатели которых указывают на I;

(2) обновить указатели всех чисел, чьи указатели указывают на I;

(3) удалить указатель от x к I.

Рандомизированная пошаговая сортировка

Рассмотрим работу, проделанную на *i*-ом шаге

Обратный анализ:

представим, что алгоритм выполняется *назад*, начиная с того, что уже имеется отсортированный список.

При анализе *i*-ого шага прямого алгоритма мы представляем, что удаляется одно из *i* чисел в *отсортированном списке* и обновляются указатели.

Работа по обновлению указателей в этом случае такая же, как если бы алгоритм выполнялся вперед как обычно.

Ключевой момент в обратном анализе:

так как в первоначальном алгоритме числа были добавлены в *произвольном* порядке,

то в обратном анализе мы можем принять, что каждое число в отсортированном списке *равновероятно* может быть удалено на этом шаге.

Рандомизированная пошаговая сортировка

Работана *i*-ом шаге

Каково ожидаемое число указателей, которые должны быть обновлены на этом шаге?

Так как у нас *i* интервалов и (*n*-*i*+1) указателей, оставшихся после удаления, то ожидаемое число указателей, которые были изменены на *i*-ом шаге - *O*((*n*-*i*)/*i*), которое есть *O*(*n*/*i*).

Просуммируем проделанную работу по всем шагам и получим верхнюю границу математического ожидания полной работы. Так как

$$\lim_{n\to\infty}\left(\sum_{i=1}^{n}\frac{1}{i}-\ln n\right)=\gamma$$

где γ - константа Эйлера, то получаем $O(\sum_i n/i)$ =

= *O*(*n log n*).

Рандомизированный алгоритм построения выпуклой оболочки

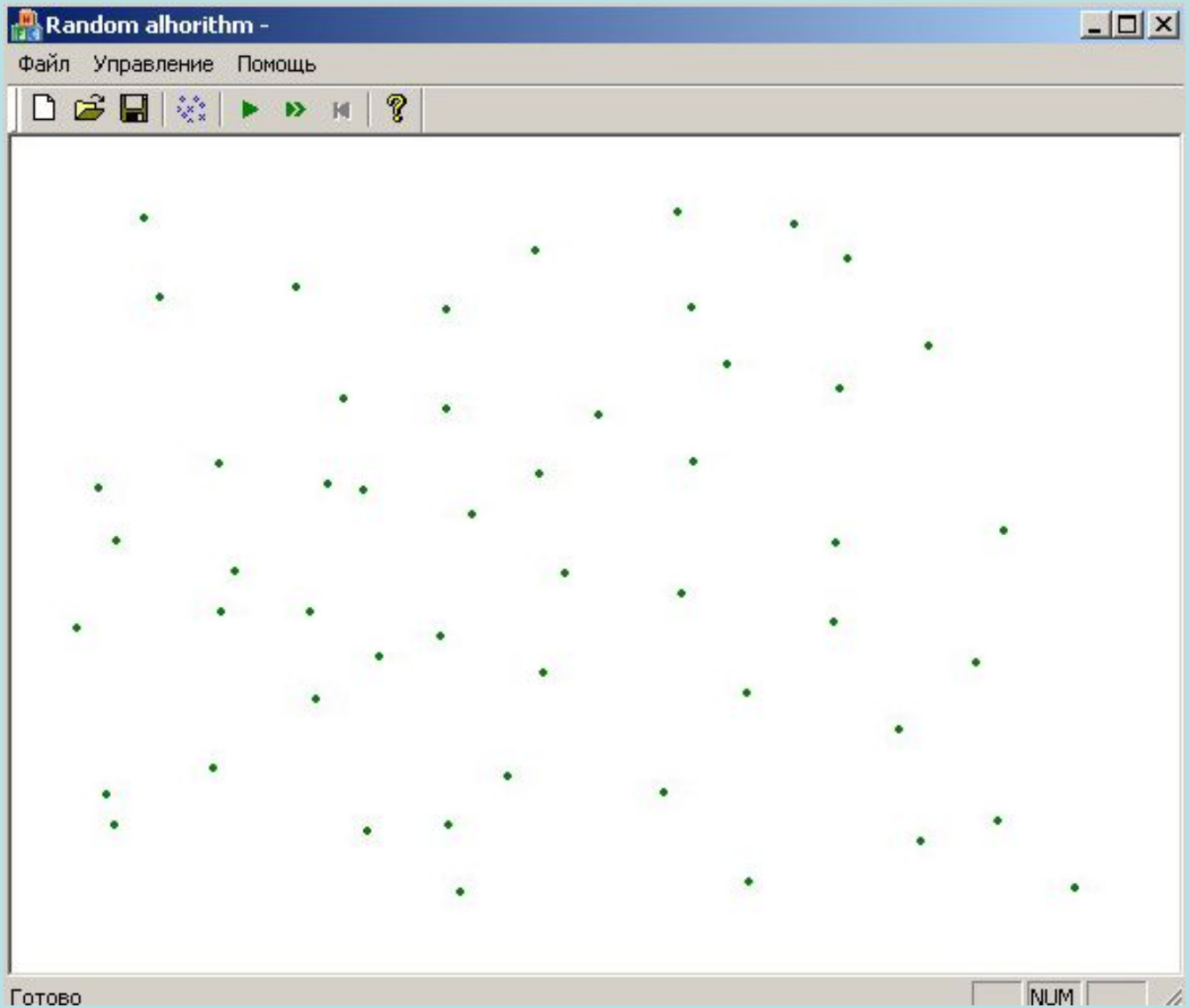
S – множество всех точек на плоскости;

$conv(S)$ – выпуклая оболочка множества S;

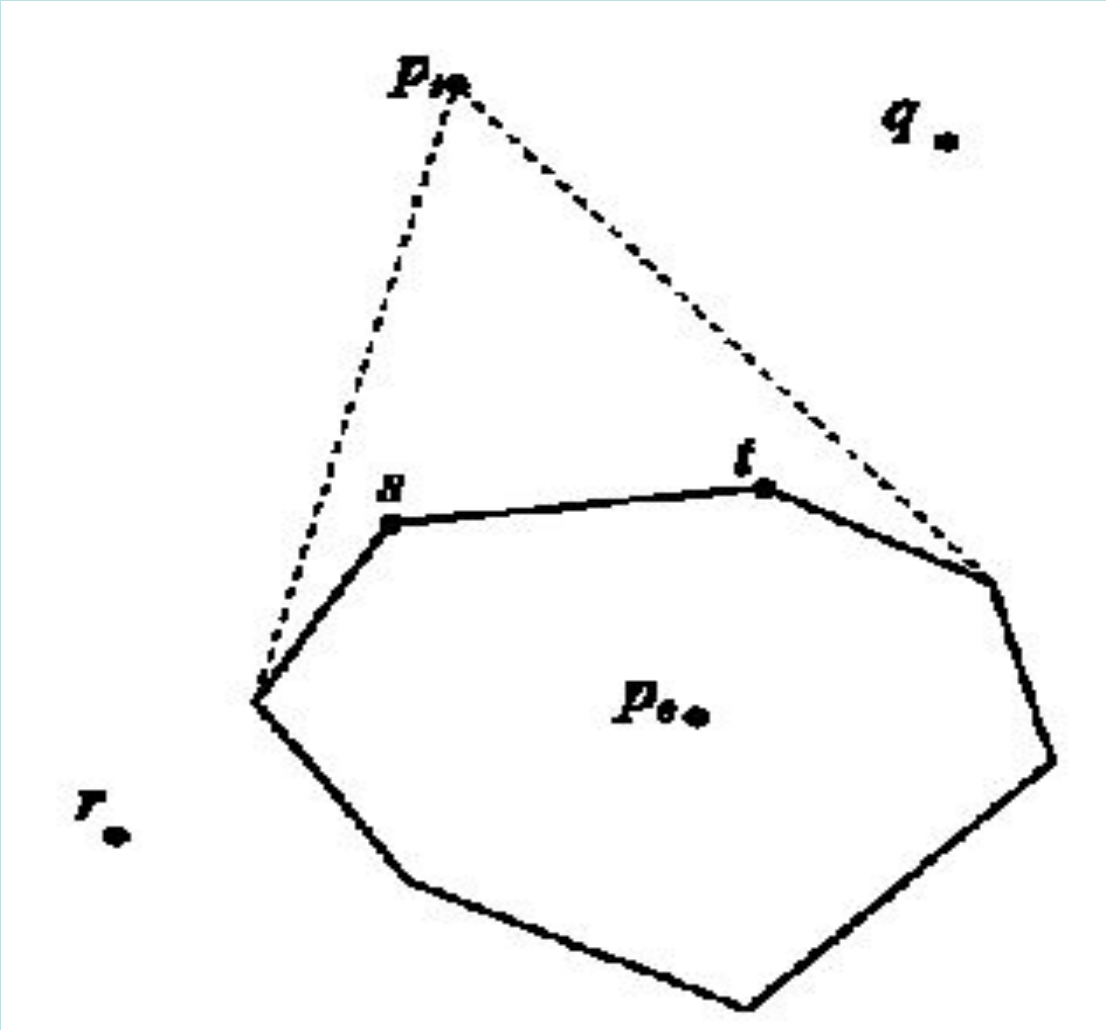
Для упрощения предполагаем, что в S нет трех точек лежащих на одной прямой

Пример работы алгоритма

1.



Рандомизированный алгоритм построения выпуклой оболочки



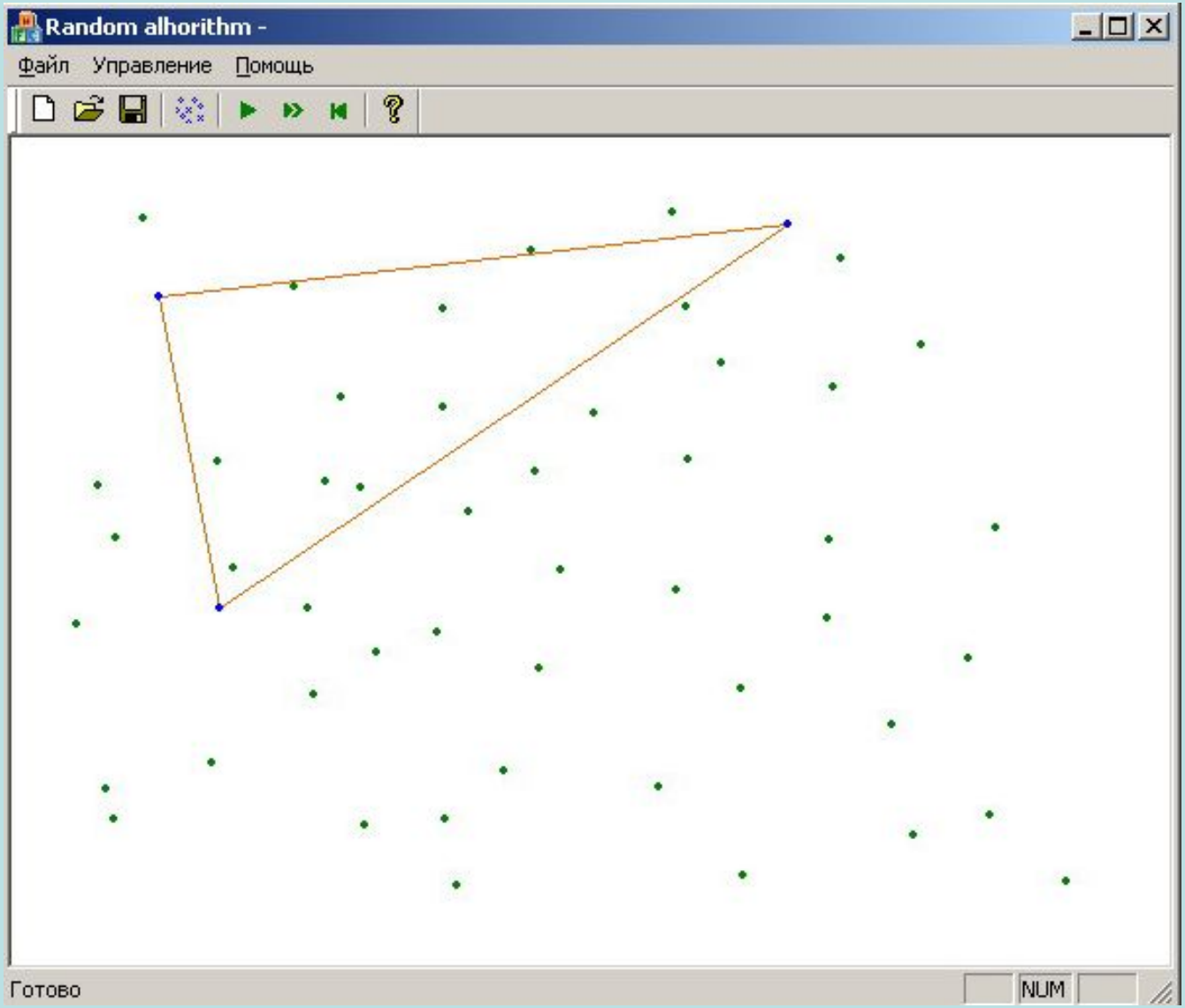
Рандомизированный алгоритм построения выпуклой оболочки

Пошаговое описание

1. *Построить выпуклую оболочку из трех точек $conv(S_3)$.*
2. *Найти точку p_0 во внутренней области $conv(S)$ - это центр масс треугольника $conv(S_3)$.*
3. *Для каждой точки p определить ребро $conv(S_3)$, пересекаемое отрезком pp_0, и сформировать двунаправленный указатель от p на ребро и обратно. При это исключить из дальнейшего рассмотрения точки, находящиеся внутри $conv(S_3)$.*

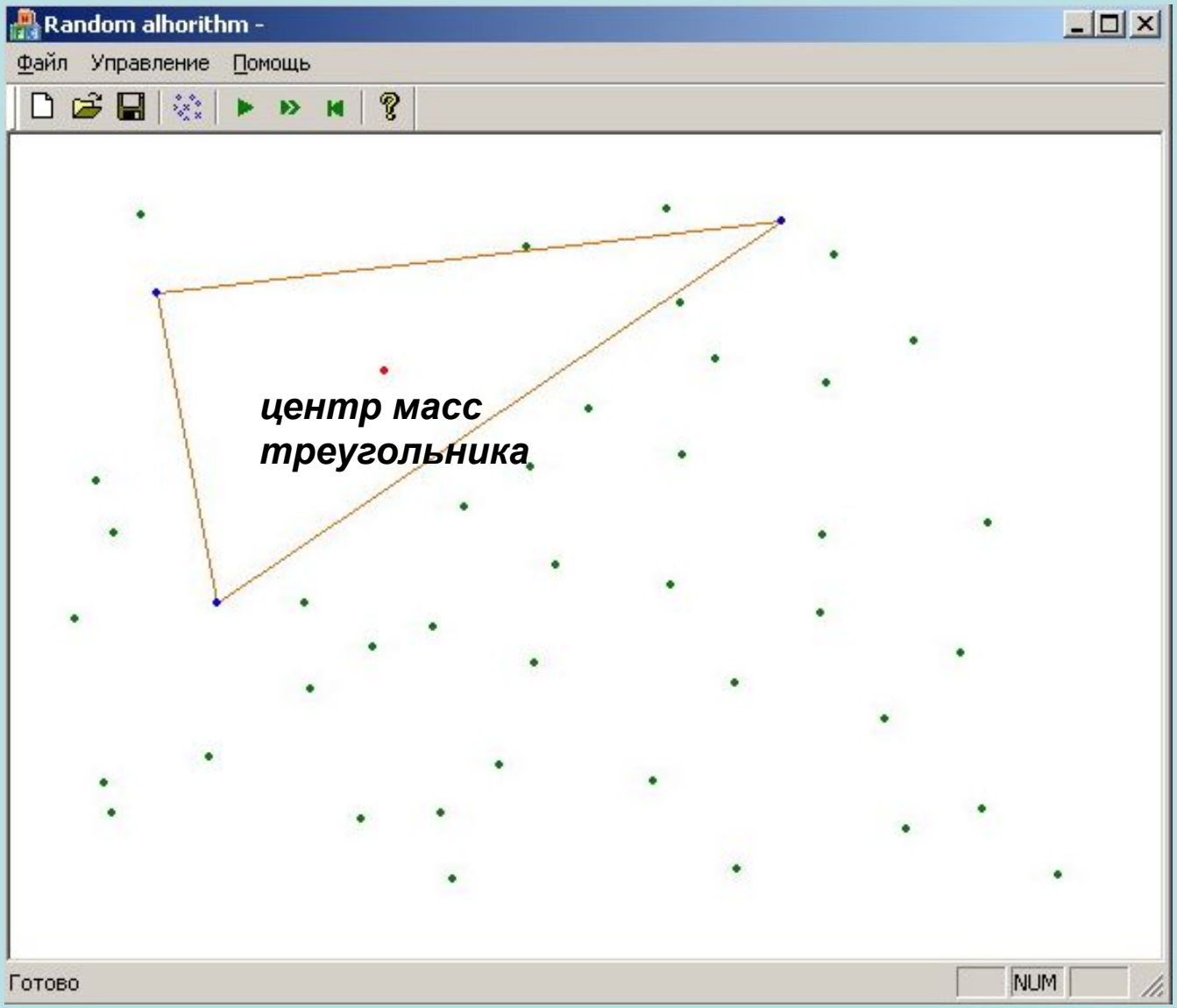
Построение произвольного треугольника

2.

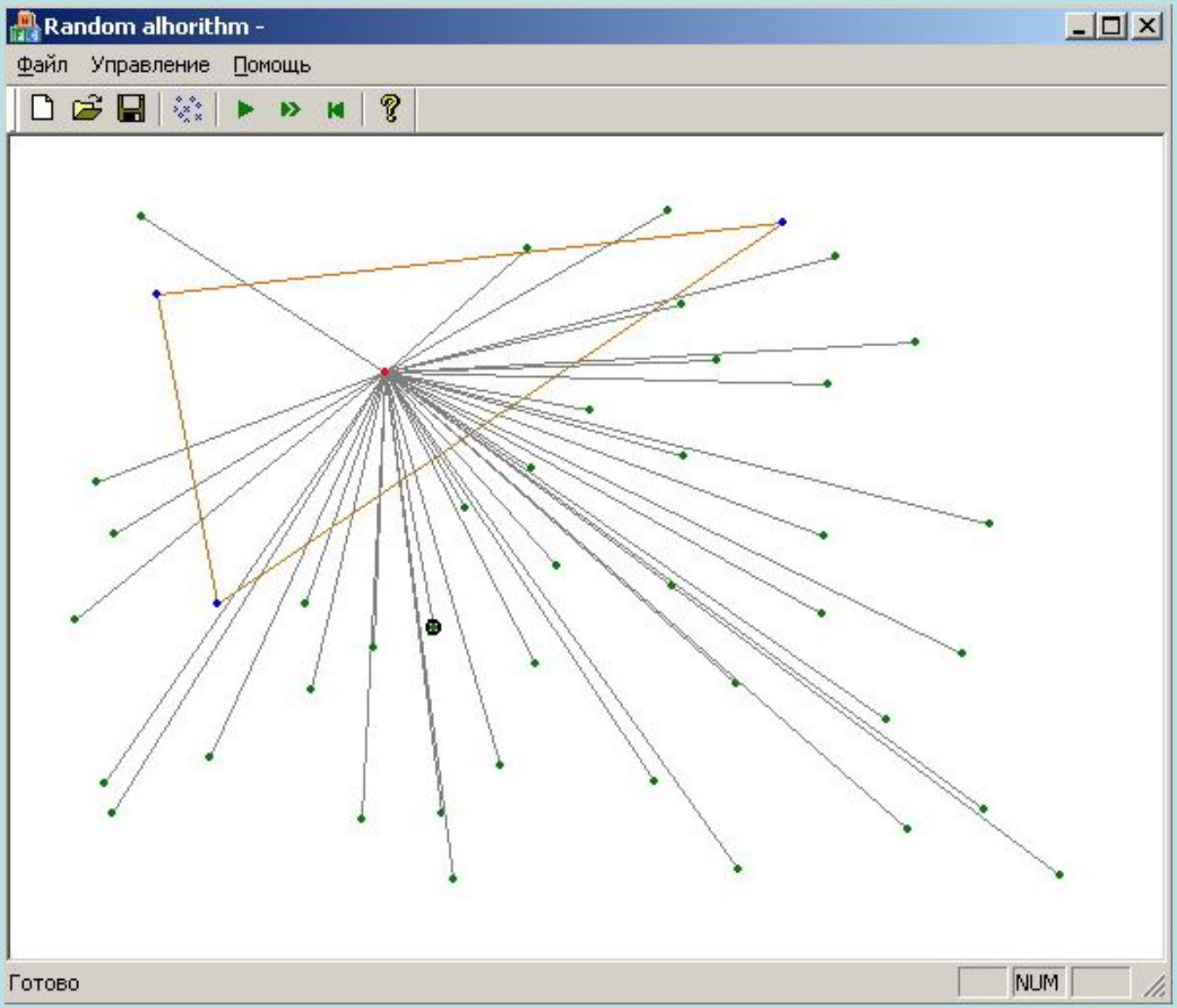


Нахождение центра масс треугольника

3.



4.



По шаговое описание (продолжение)

4. ПОКА ($S \backslash S_{i-1}$ не пусто) {

Выбрать случайным образом точку p_i из $S \backslash S_{i-1}$;

Найти в $conv(S_{i-1})$ вершину v_1, которая является левой соседней (опорной) *для p_i в $conv(S_i)$. В процессе поиска запомнить в списке recheck указатели от удаляемых ребер;*

Найти в $conv(S_{i-1})$ вершину v_2, которая является правой соседней (опорной) *для p_i в $conv(S_i)$, дополнив при этом список recheck указателями от удаляемых ребер;*

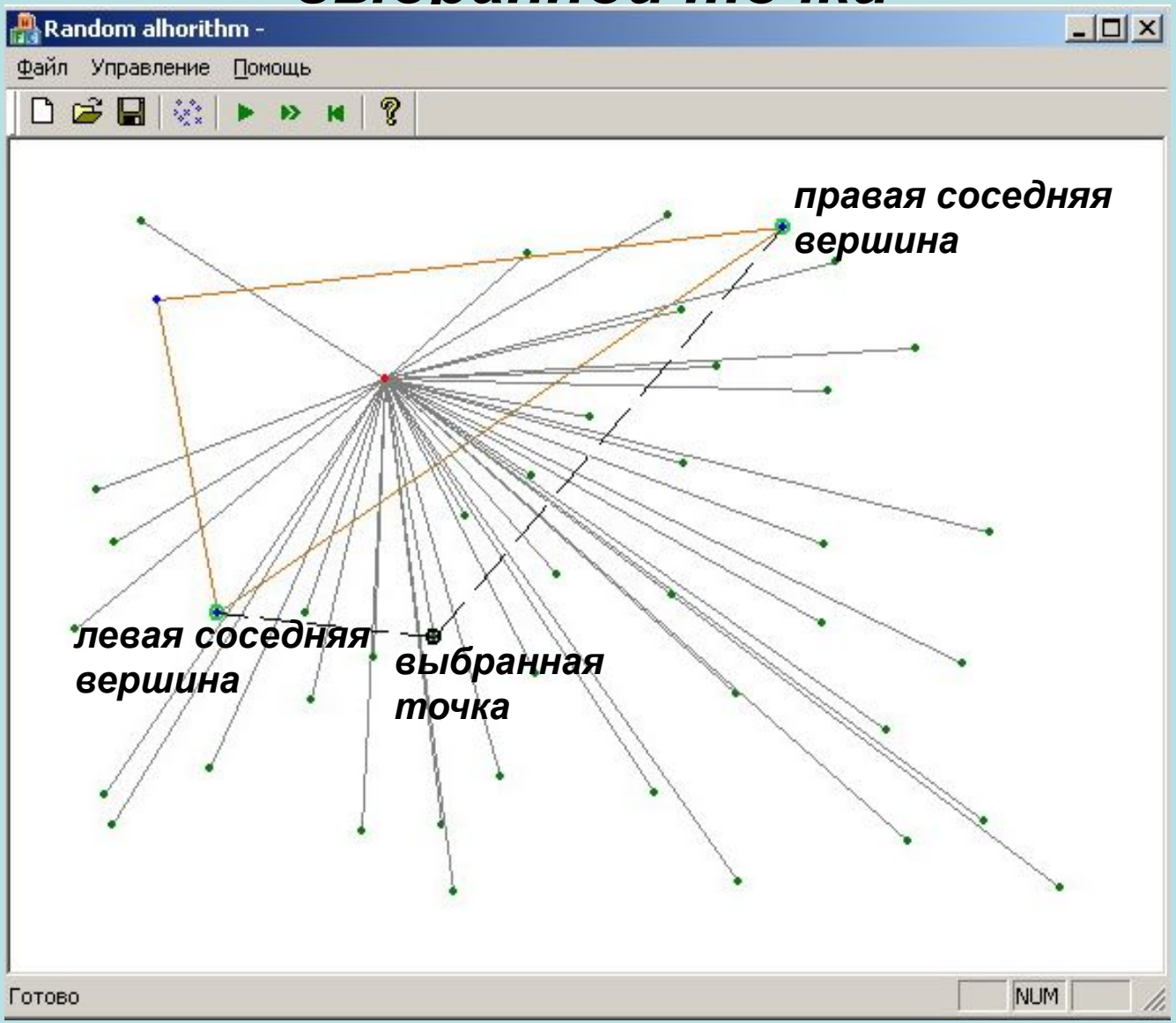
Вставить p_i в выпуклую оболочку, сформировав $conv(S_i)$;

Для каждой точки , указатель на которую содержится в списке recheck, обновить её указатель на ребро $conv(S_i)$, пересекаемое отрезком pp_0, указав его на $[p_i, v_1]$ или $[p_i, v_2]$. При этом исключить из дальнейшего рассмотрения точки, которые попадают внутрь $conv(S_i)$;

}

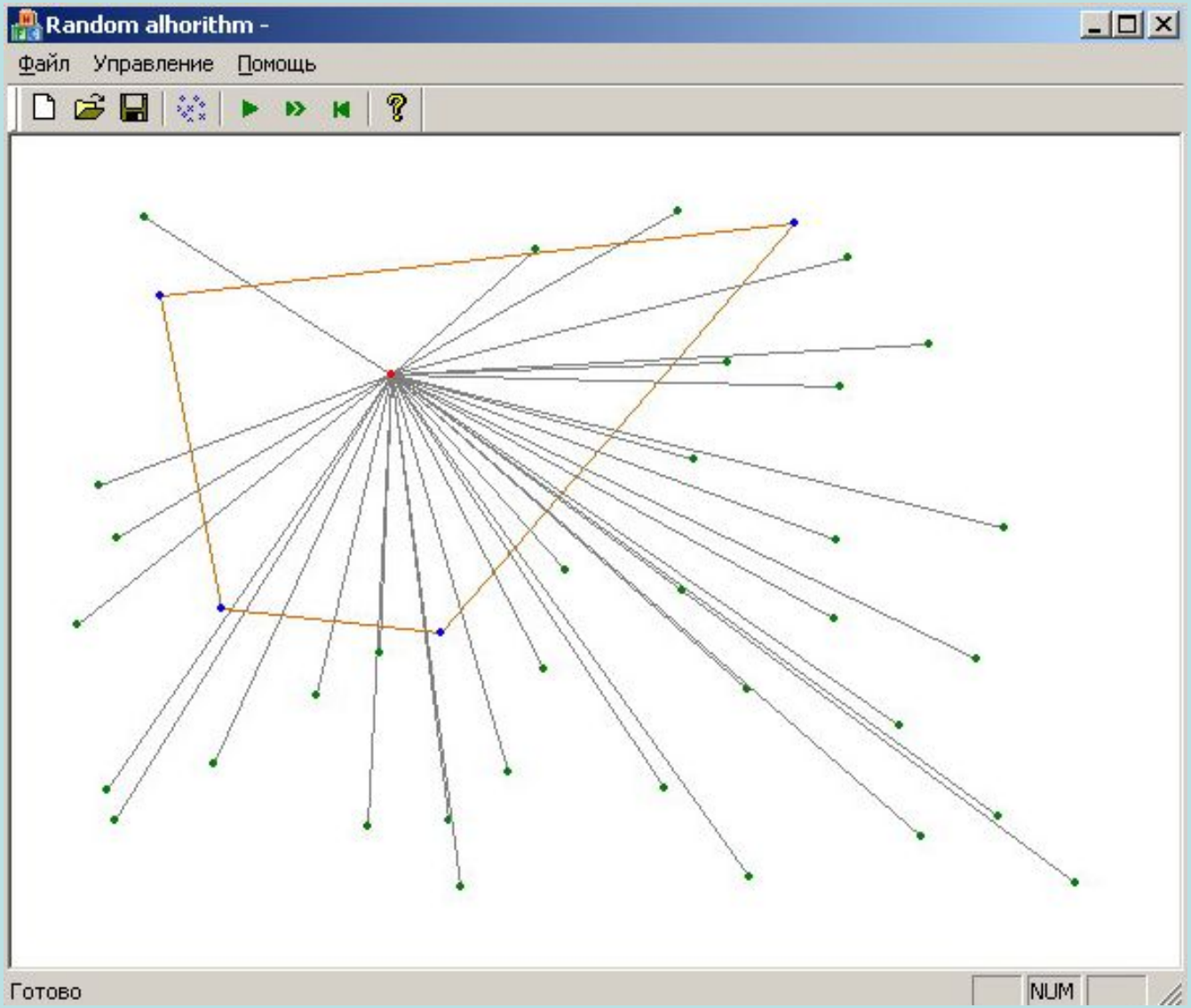
Нахождение соседних вершин для выбранной точки

5.



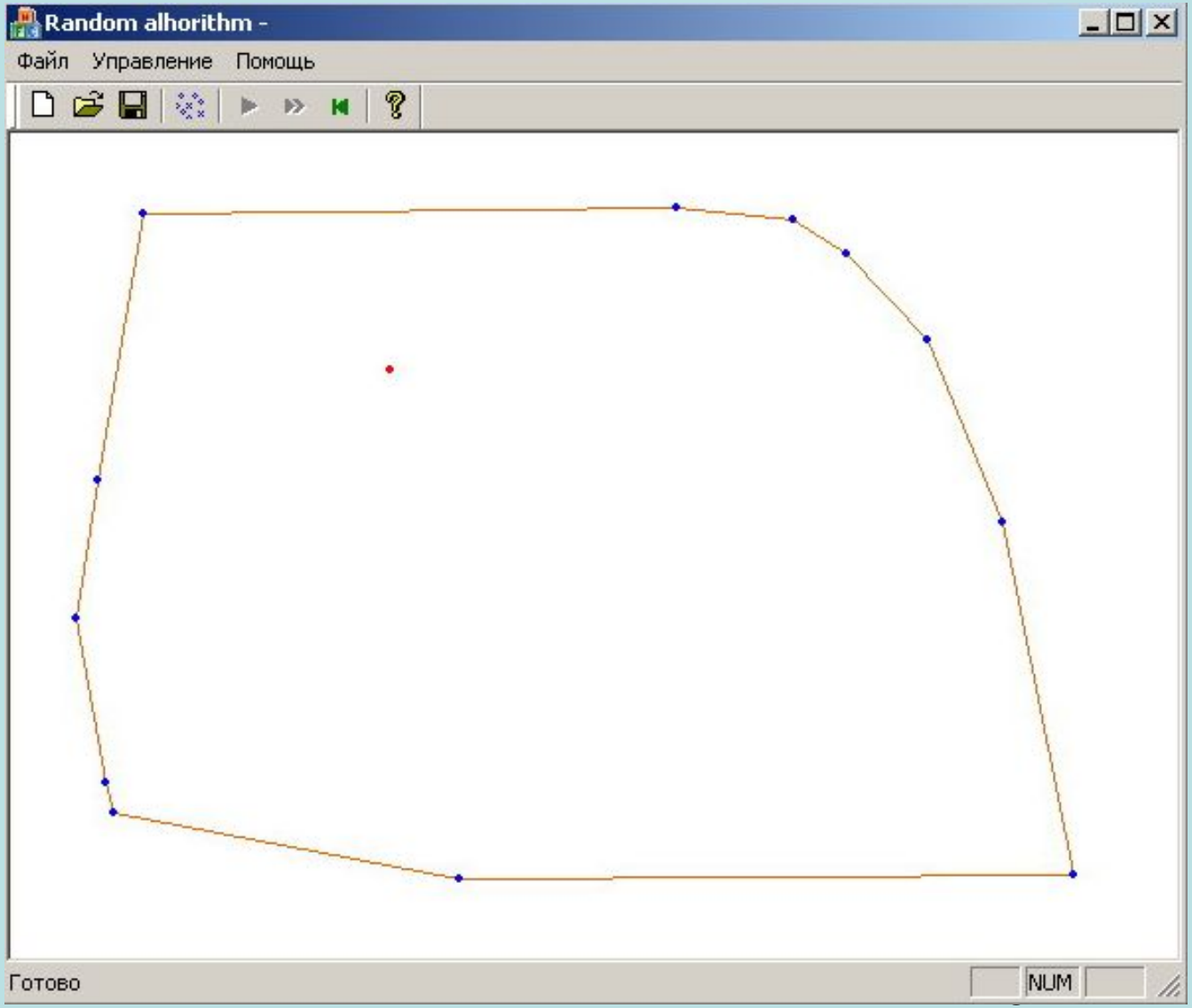
Добавление новой вершины

6.



Построенная выпуклая оболочка

7.



Рандомизированный алгоритм построения выпуклой оболочки

Теорема: *Среднее время работы описанного выше рандомизированного алгоритма для вычисления выпуклой оболочки n точек на плоскости -* $O(n \log_2 n)$.