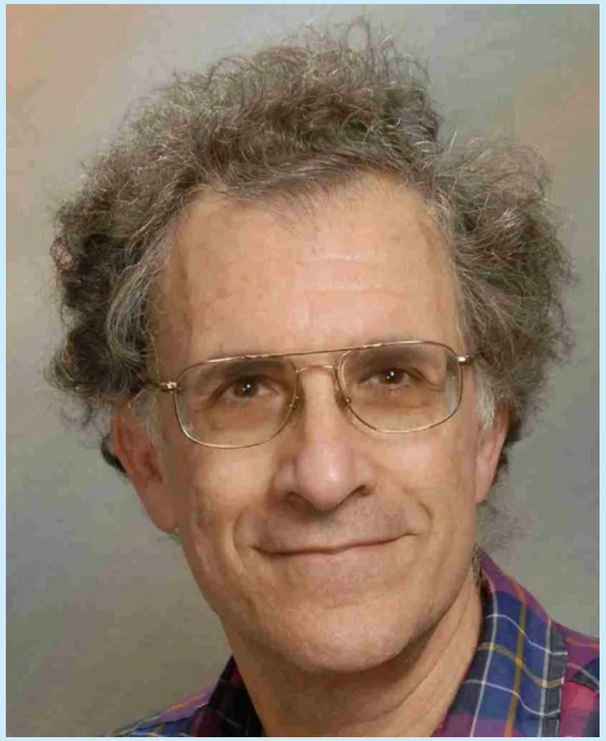
Дерево квадрантов (Quadtree)

Выполнил: Бекмансуров Илья, гр. 11-102

План

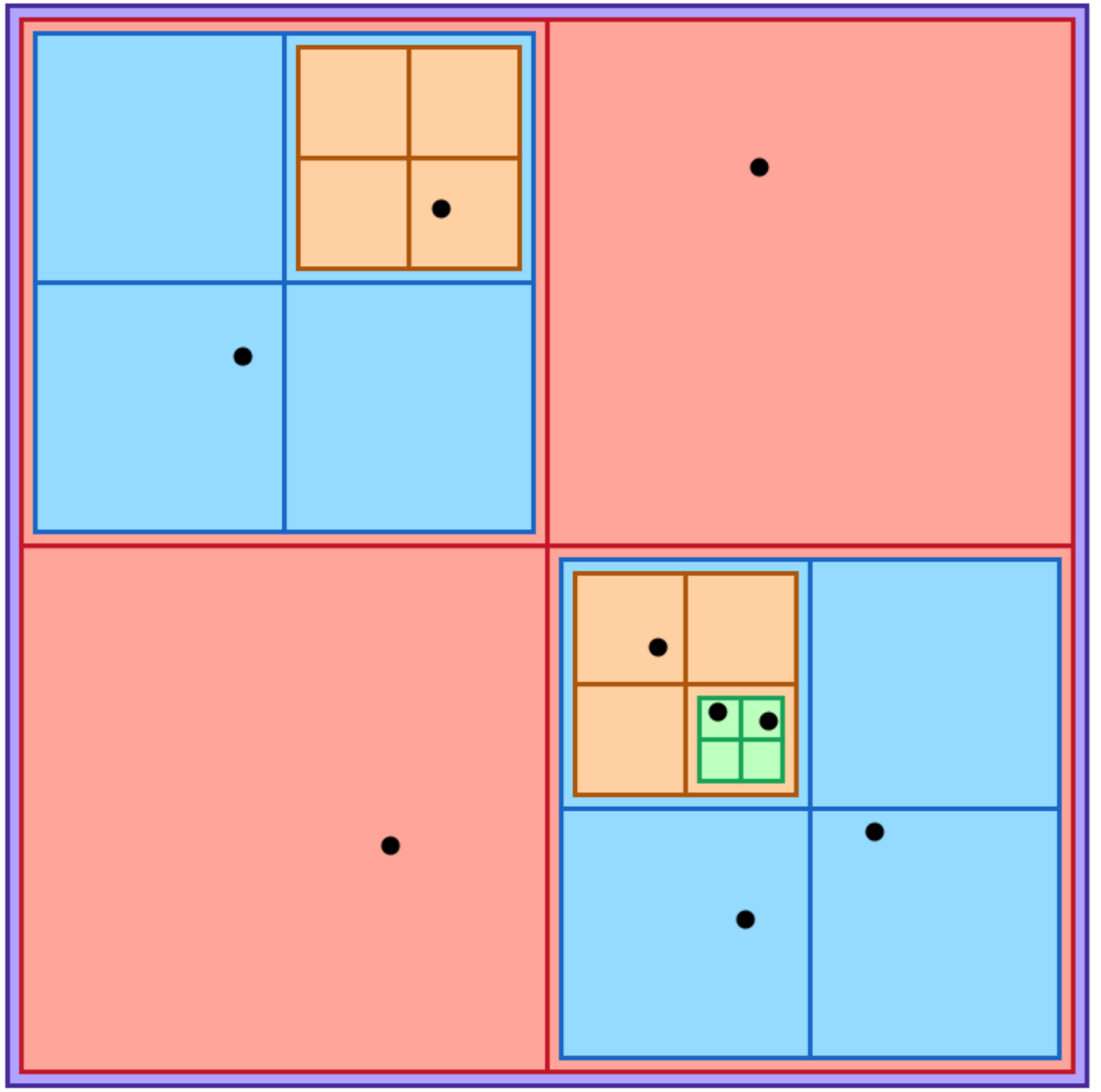
Что мы обсудим сегодня

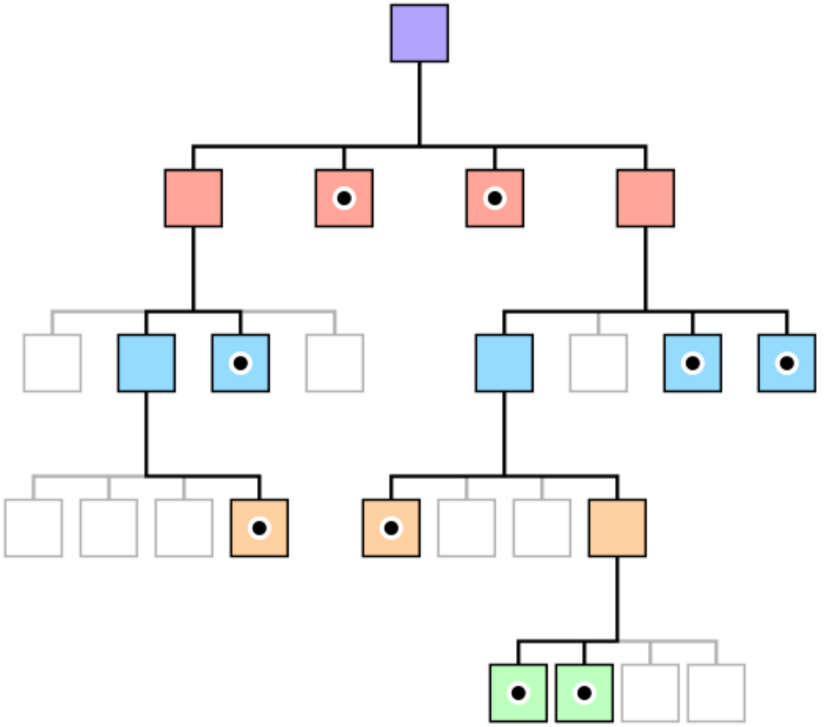
- Историческая справка
- Основной принцип устройства
- Оценка временной сложности
- Плюсы и минусы
- Применимость

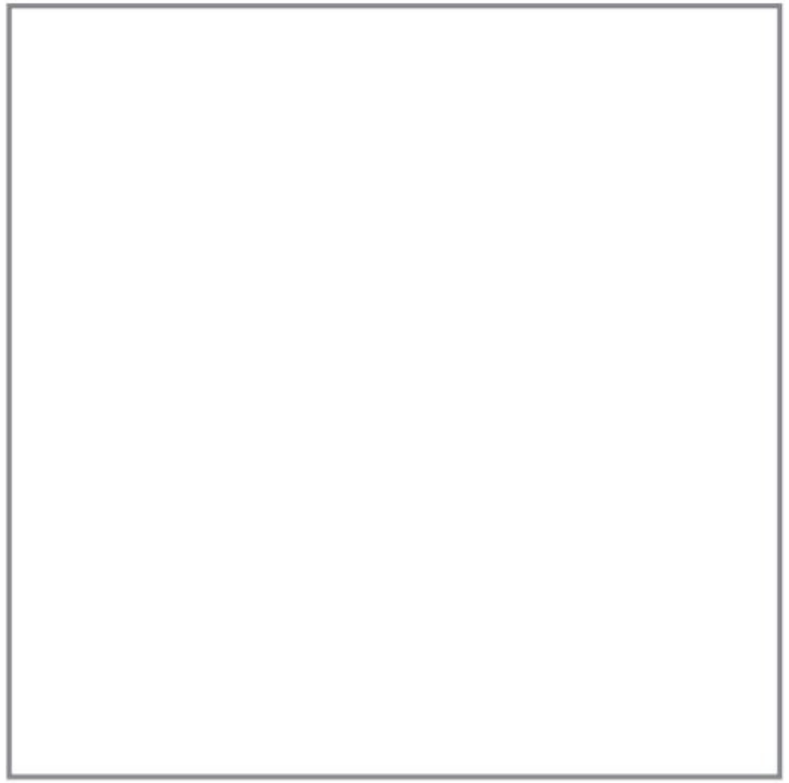


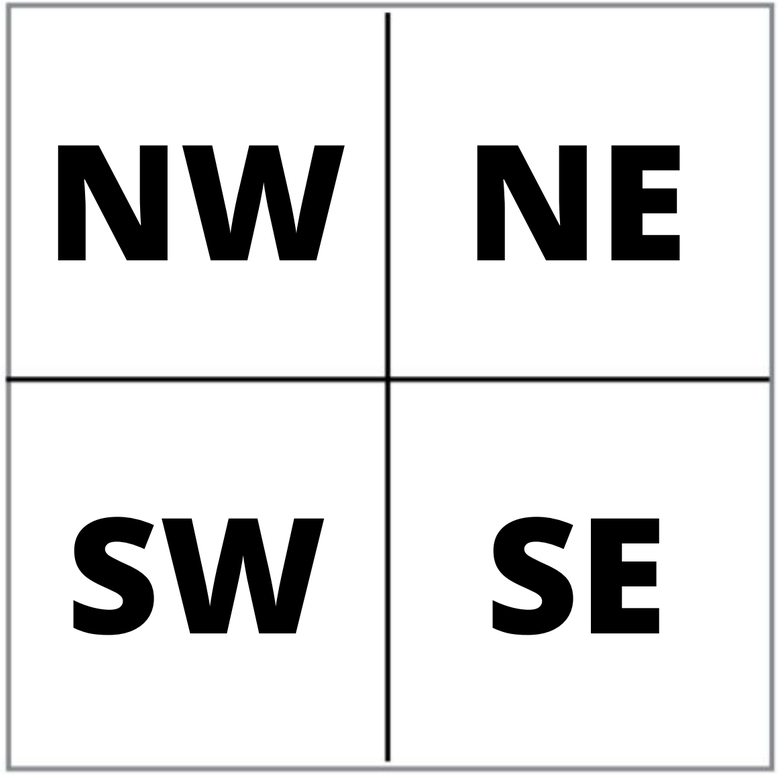
Р.Финкель

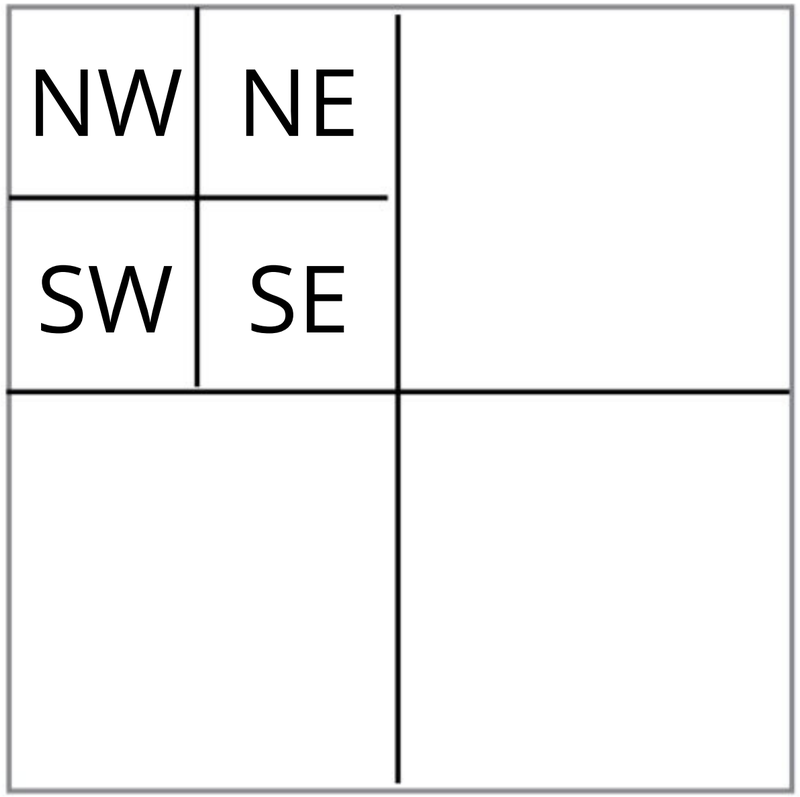
Дерево квадрантов считается одной из первых структур данных для данных высокой размерности (многомерных данных). Авторами дерева квадрантов являются американские учёные Рафаэль Финкель и Джон Бентли. Впервые эта структура данных была представлена и рассмотрена ими в работе «Quad Trees: A Data Structure for Retrieval on Composite Keys» в апреле 1974 года.

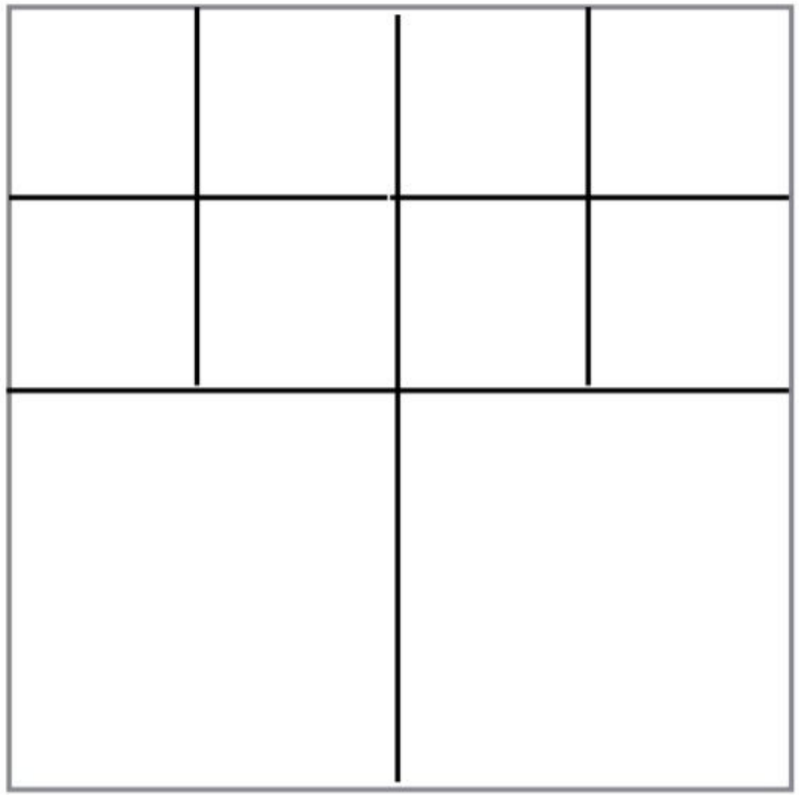


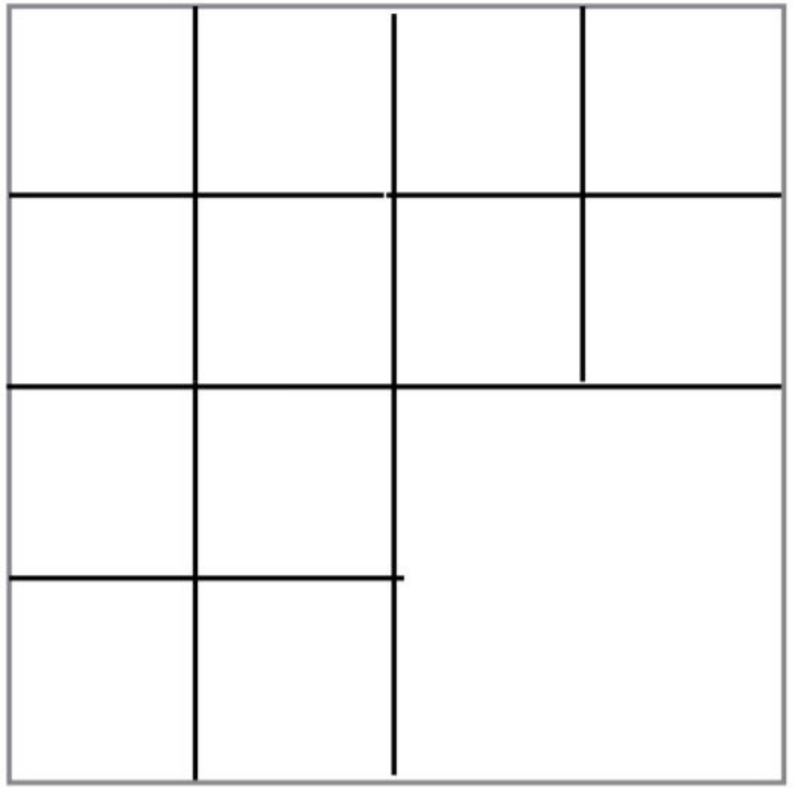





NW
NE
SW
SE


NW
NE
SW
SE





Основные операции

Вставка

Метод осуществляет рекурсивную вставку точки в соответствующий узел дерева, осуществляя разбиение, если это необходимо. Сначала происходит проверка, принадлежит ли объект дереву. Если нет, то он игнорируется. Затем проверяется, может ли объект поместиться в текущий узел. Если это так, то осуществляется вставка объекта в данный узел, иначе – узел разбивается на 4 квадранта, и лишь потом объект вставляется в тот квадрант, которому он принадлежит

Разбиение

Метод создает 4-х потомков, которые делят исходный узел на 4 равных квадранта. Затем уже вставленные ранее в разбитый узел объекты перемещаются в соответствующие дочерние узлы

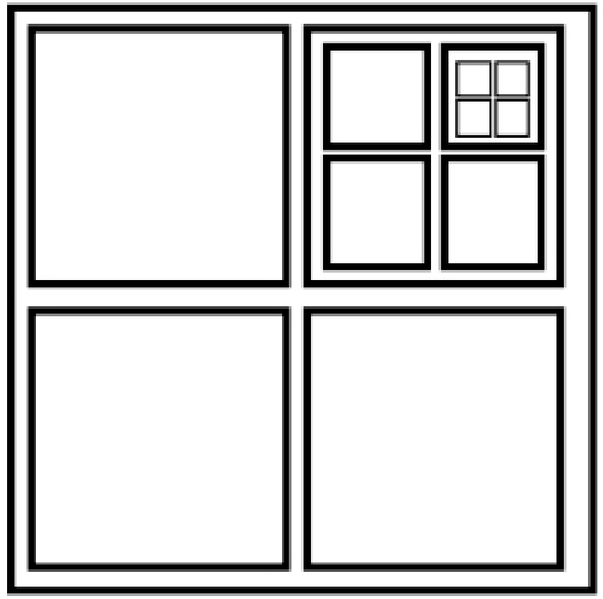
Поиск

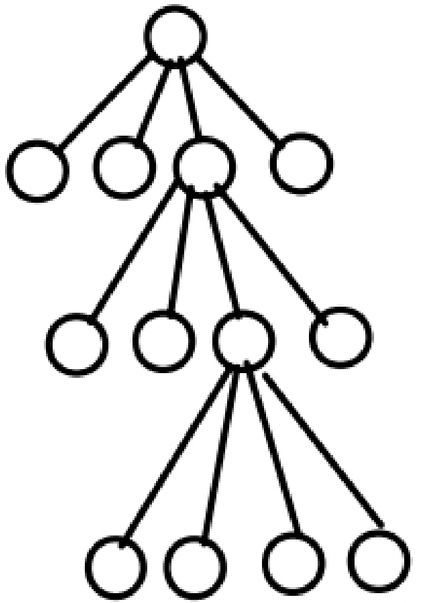
Метод ищет все точки, расположенные внутри области поиска. Вначале проверяется, пересекаются ли границы узла с границами области поиска. Если это не так, то можно пропустить рассматриваемый узел со всеми его дочерними узлами, что значительно сокращает время поиска, т.к. не приходится проходится по узлам и объектам, которые заведомо не попадают в область поиска

Пусть N - количество объектов в дереве квадрантов; P - максимальное количество объектов в одном узле.

Поиск:

асимптотическая оценка – ***O(N/P)***, так как дерево может выродиться в список, и тогда до самой удаленной от корня позиции придется спускаться N/P раз. Например, пусть имеется набор расположенных близко друг к другу точек. Каждый раз при вставке очередной точки, она добавляется в один и тот же узел предыдущего родительского узла. Таким образом, каждый узел имеет ровно одного потомка. Следовательно, дерево вырождается в связный список длины N.





Однако ***амортизационно*** сложность поиска – ***O(logN)***, так как предполагается, что дерево более-менее сбалансировано и, следовательно, в список не вырождается.

Вставка:

Для ***амортизационного анализа*** рассмотрим начальное и конечное положение каждого из *N* вставляемых объектов.

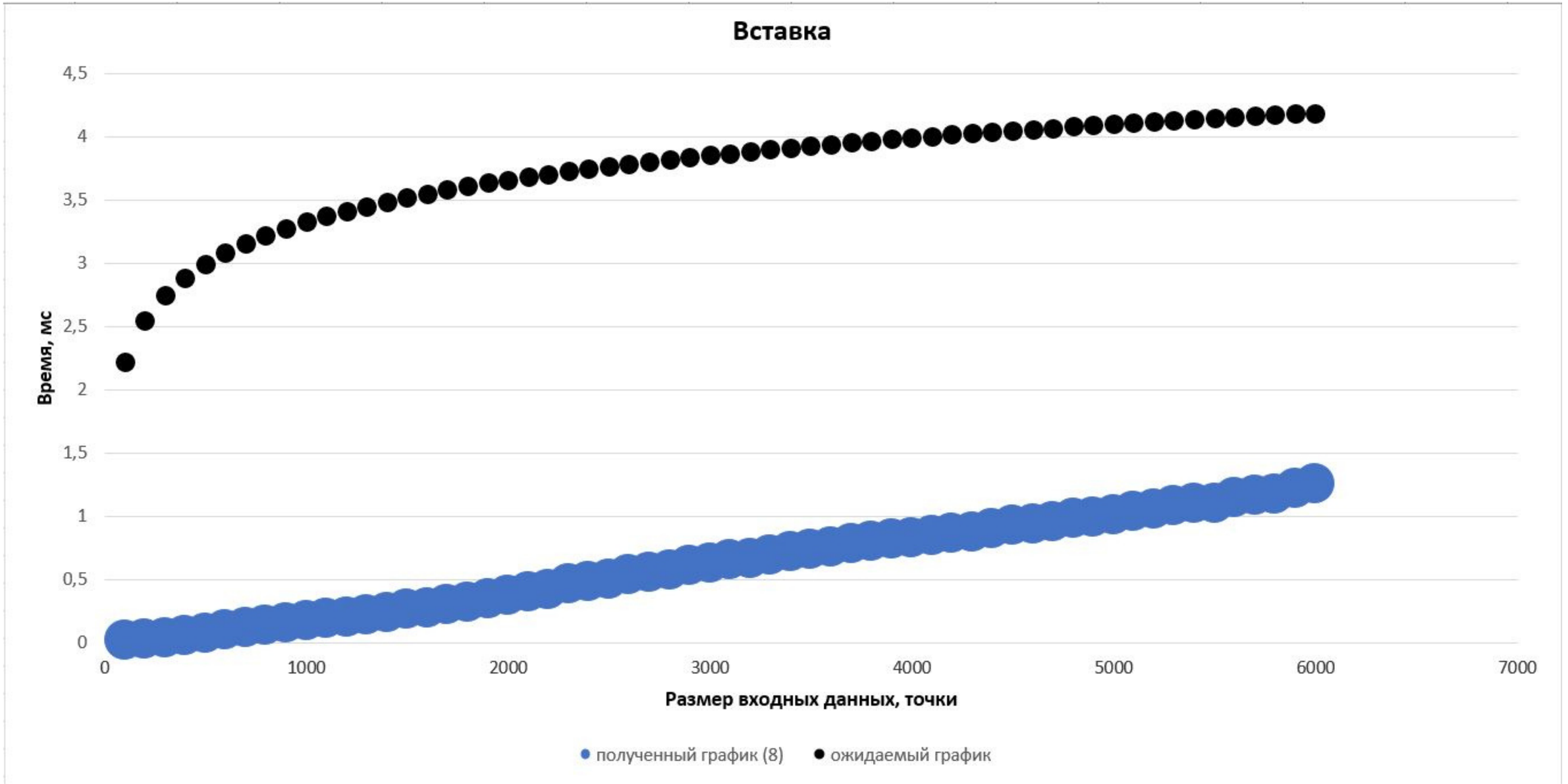
Пусть *h(T)* – конечная высота дерева квадрантов *T*, *h(i)* – конечная позиция (глубина) объекта *i*, *t(i)* – позиция объекта *i* в момент, когда он был вставлен.

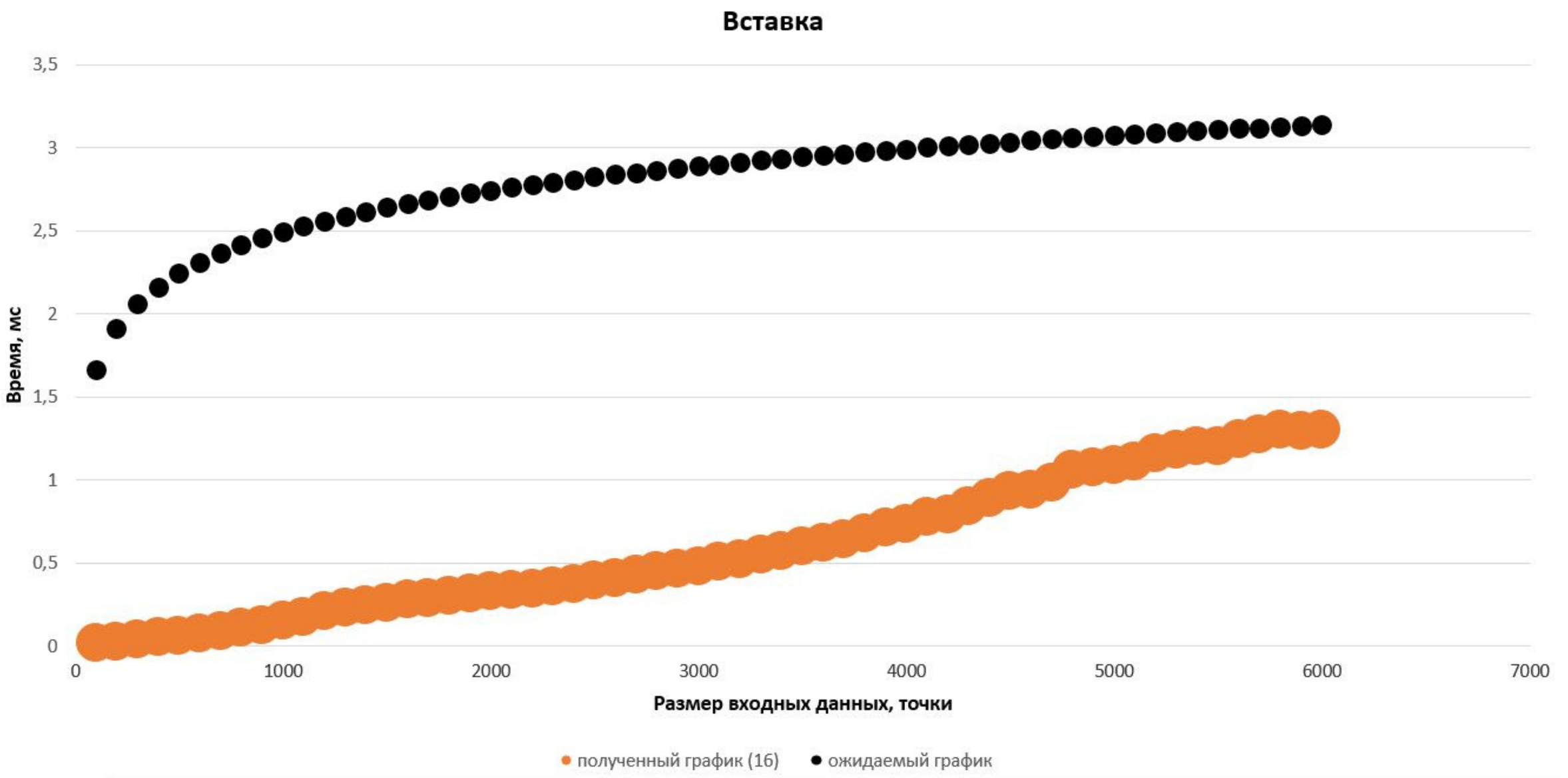
При вставке *i*-ый объект посетит *t(i)* узлов дерева, пока не найдет корректный для вставки. Стоимость этой операции – *t(i)*.

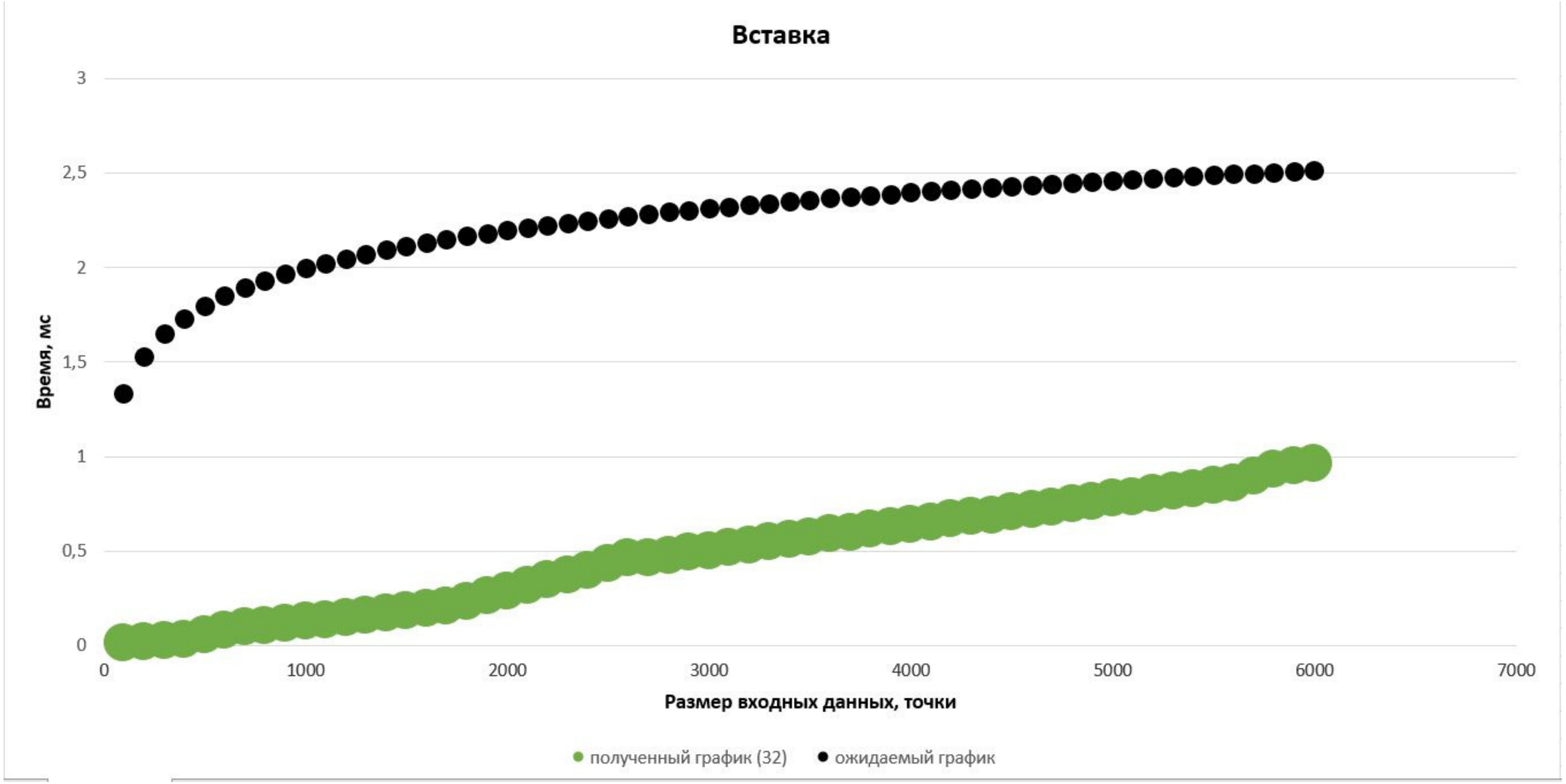
Как мы уже знаем, когда число объектов в узле превышает его вместимость, он разбивается на 4 подквадранта, и, следовательно, глубина каждого объекта увеличивается на 1. Таким образом, в процессе вставки всех *N* объектов из-за того, что узлы дерева разбиваются, объект *i* переместится *h(i) – t(i)* раз.

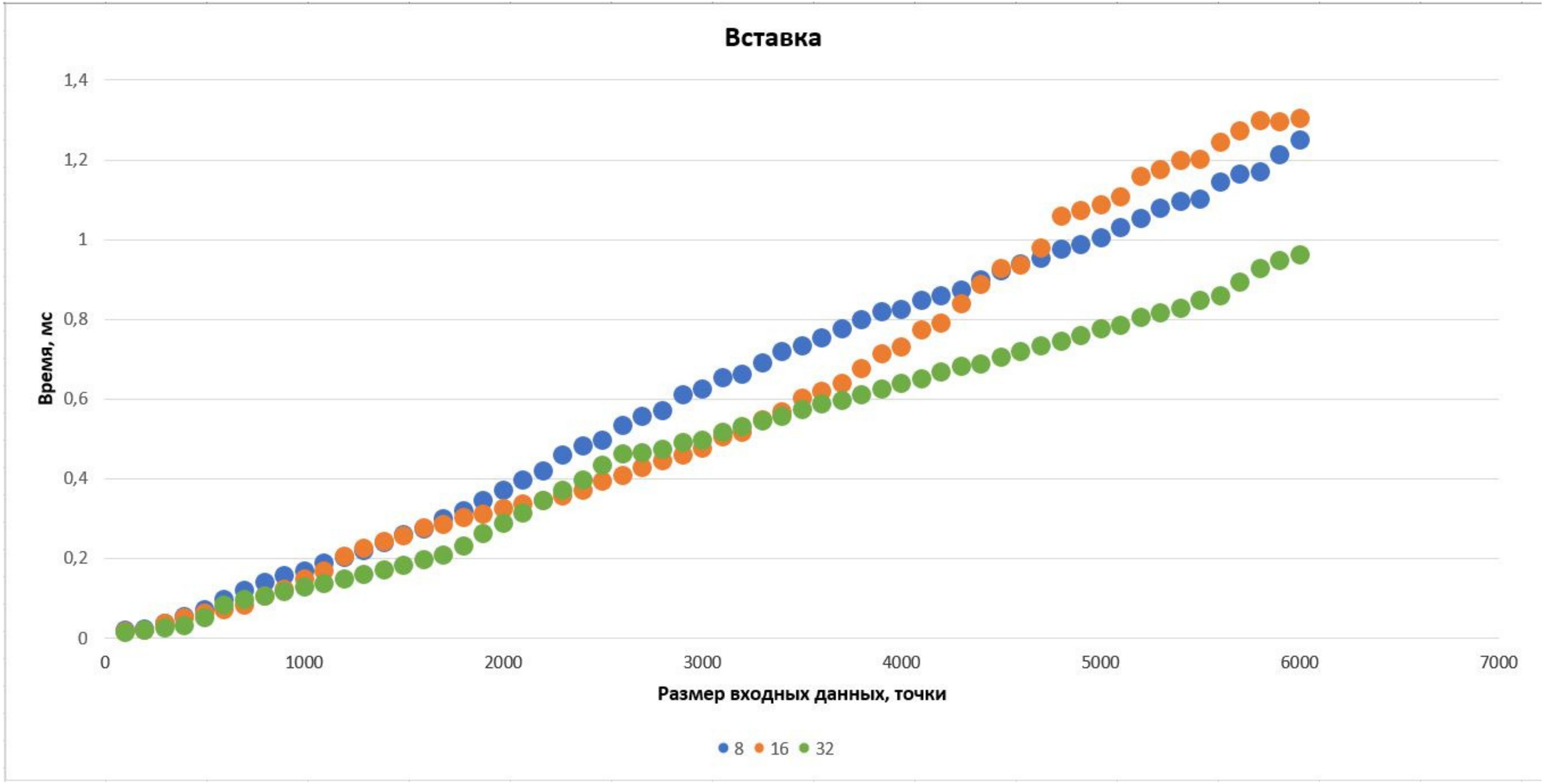
Получается, чтобы достичь своей конечной позиции *h(i)* объект *i* накапливает сложность $O(t(i) + h(i) - t(i)) = O(h(i)) \sim$ ***O(logN)*** (так как высота дерева = $O(logN)$).

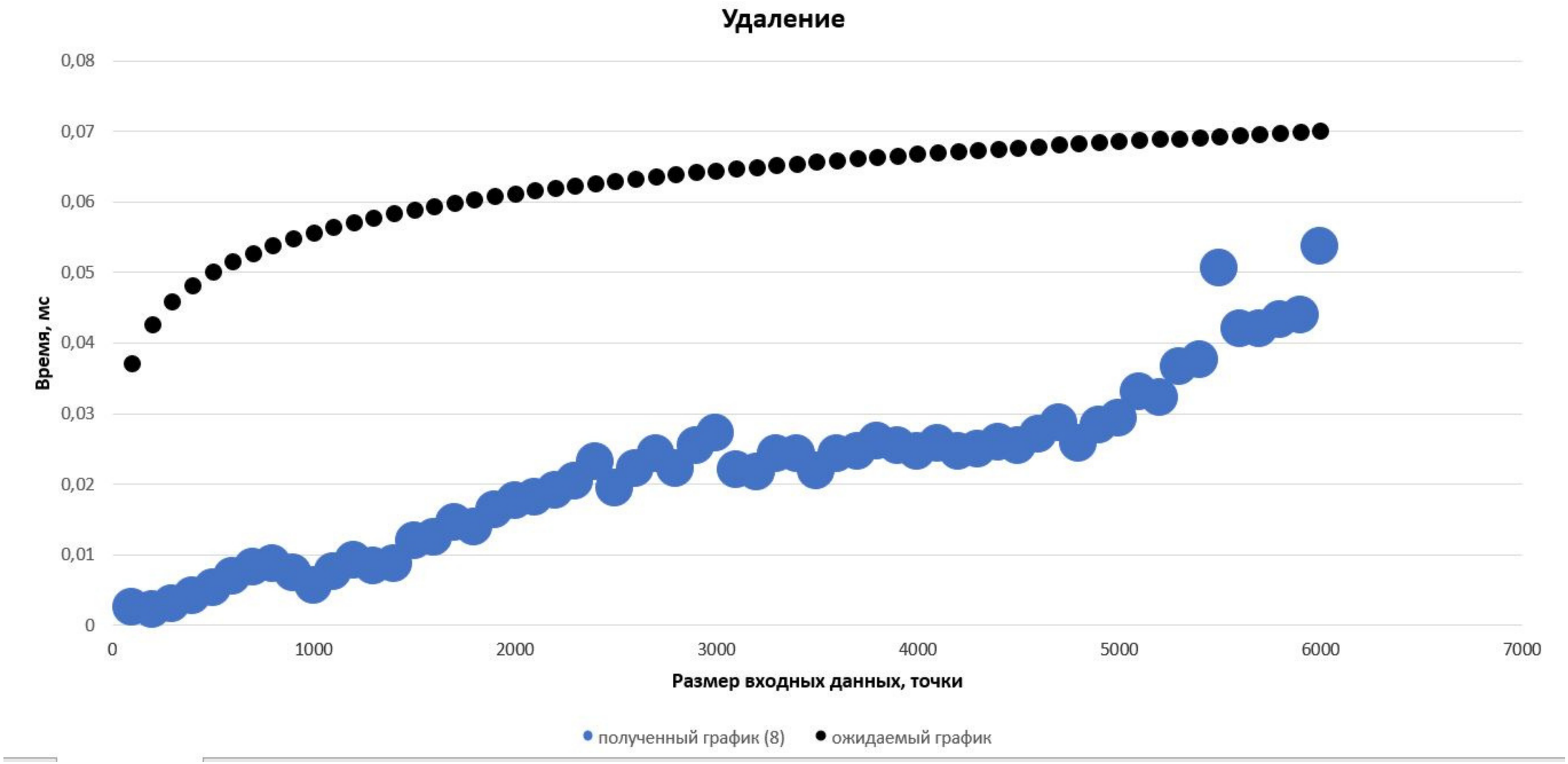
Временная сложность операции ***удаления*** также ***O(logN)***

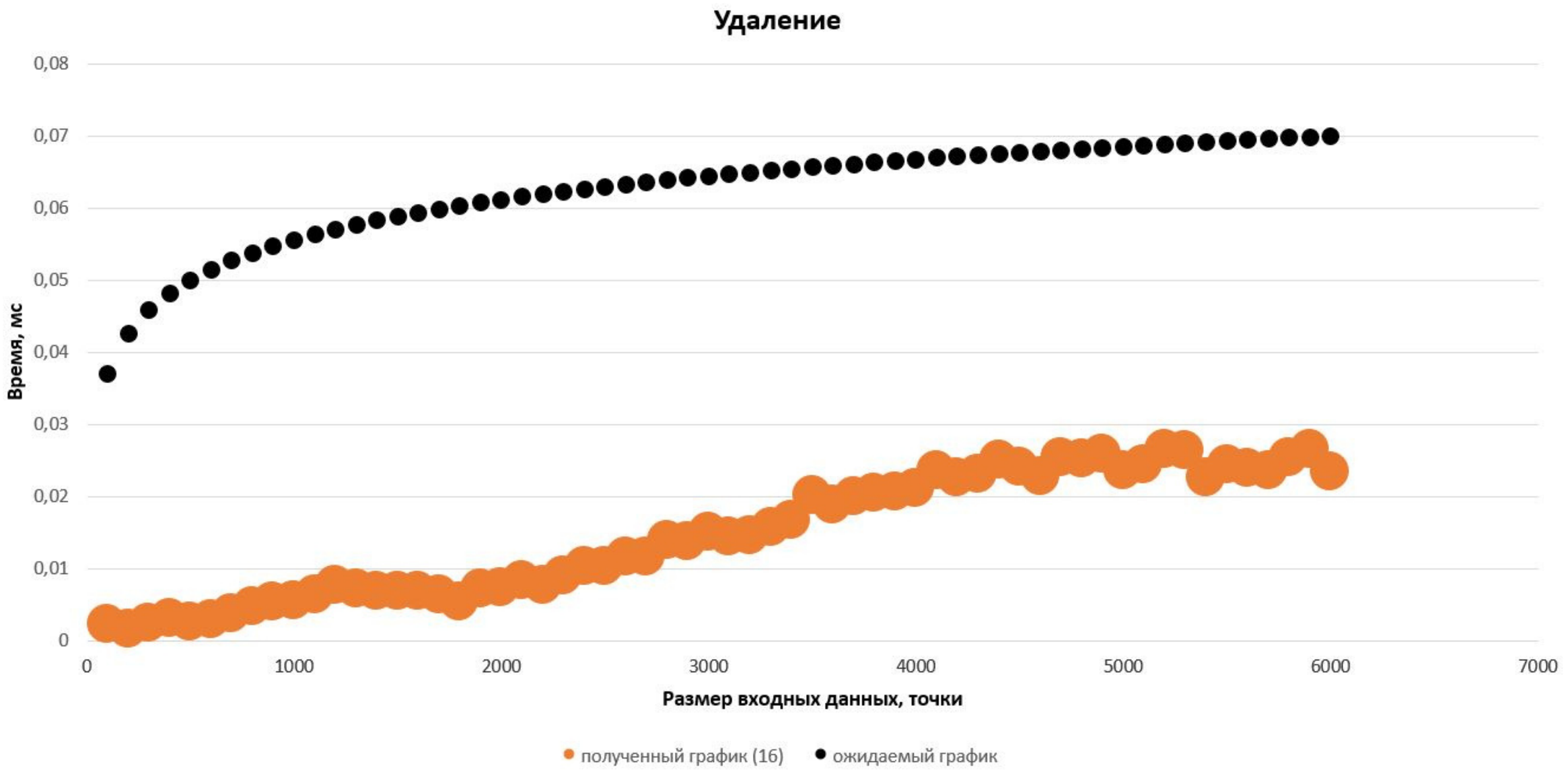

Вставка
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (8)
ожидаемый график

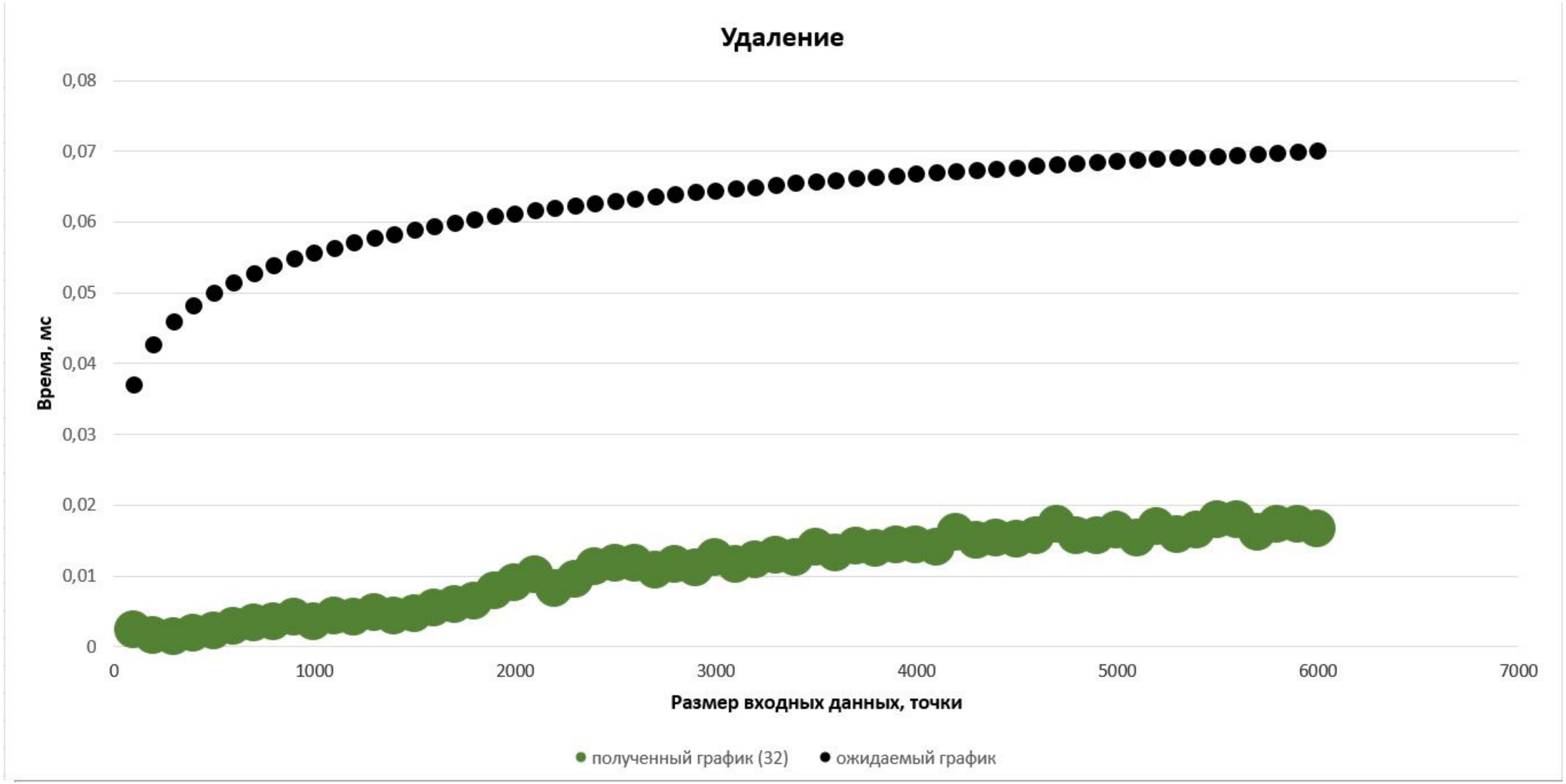

Вставка
3,5
3
2,5
2
1,5
1
0,5
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (16)
ожидаемый график

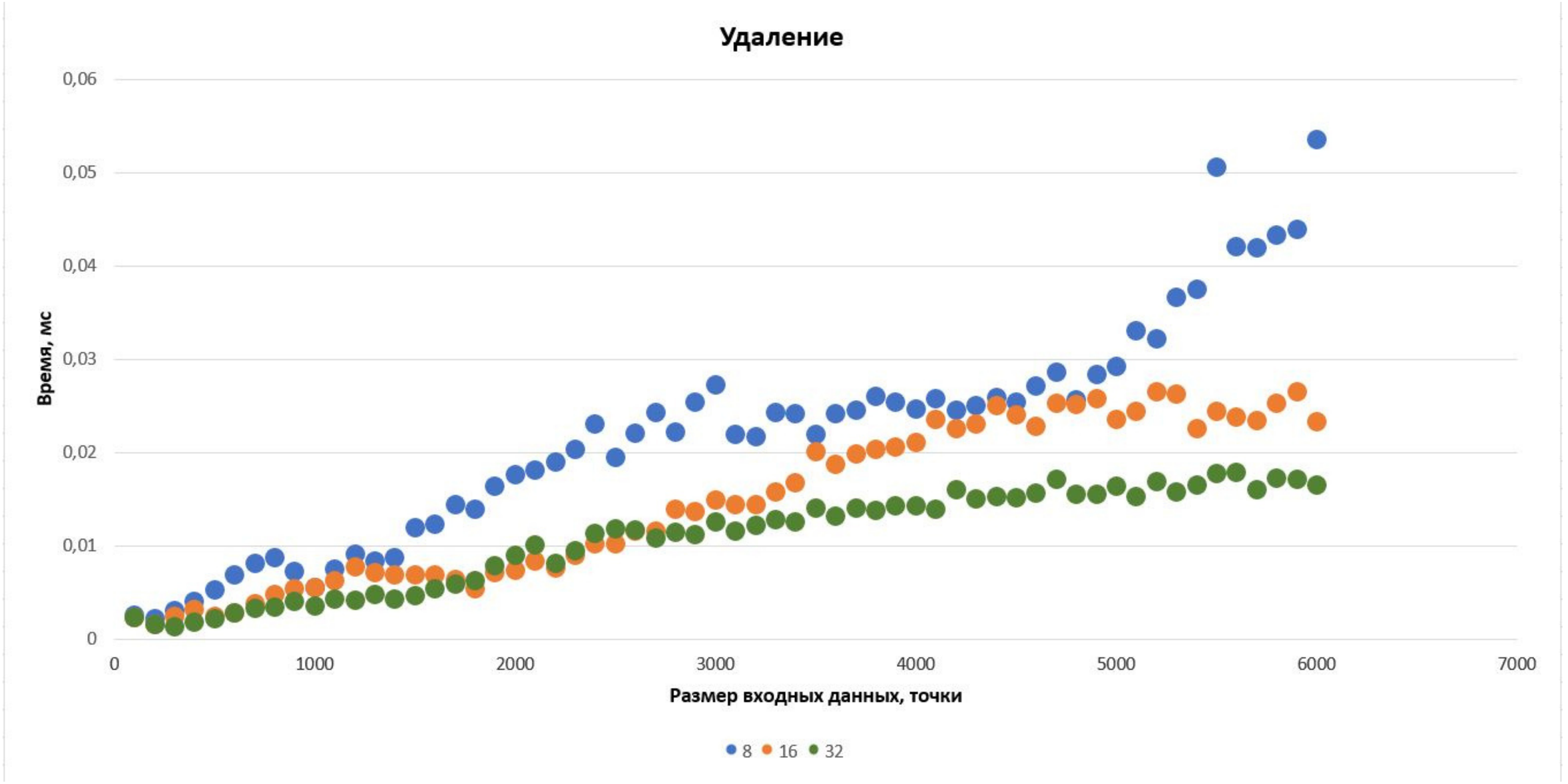

Вставка
3
2,5
2
1,5
1
0,5
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (32)
ожидаемый график

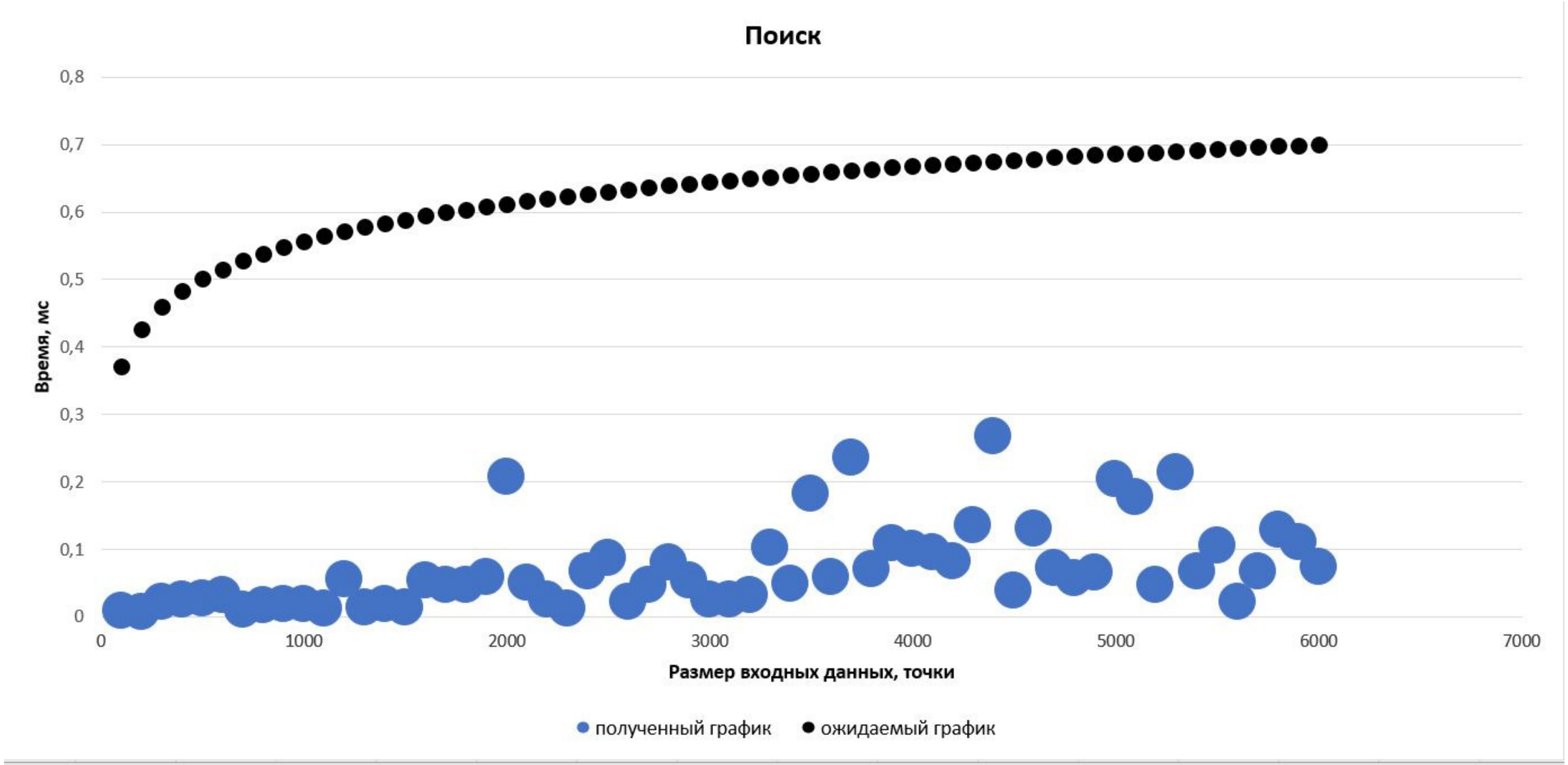

Вставка
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
8
16
32

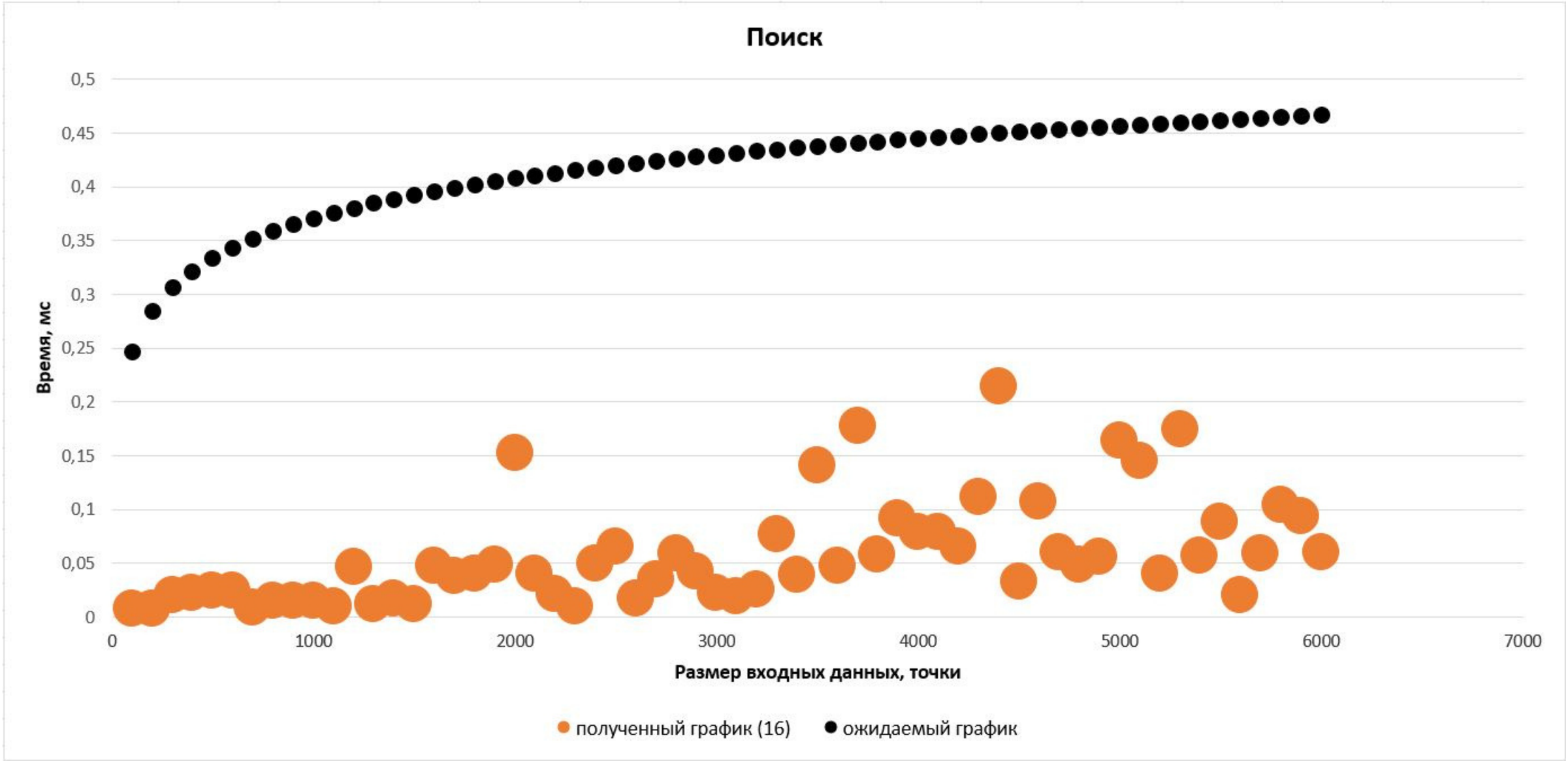

Удаление
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (8)
ожидаемый график

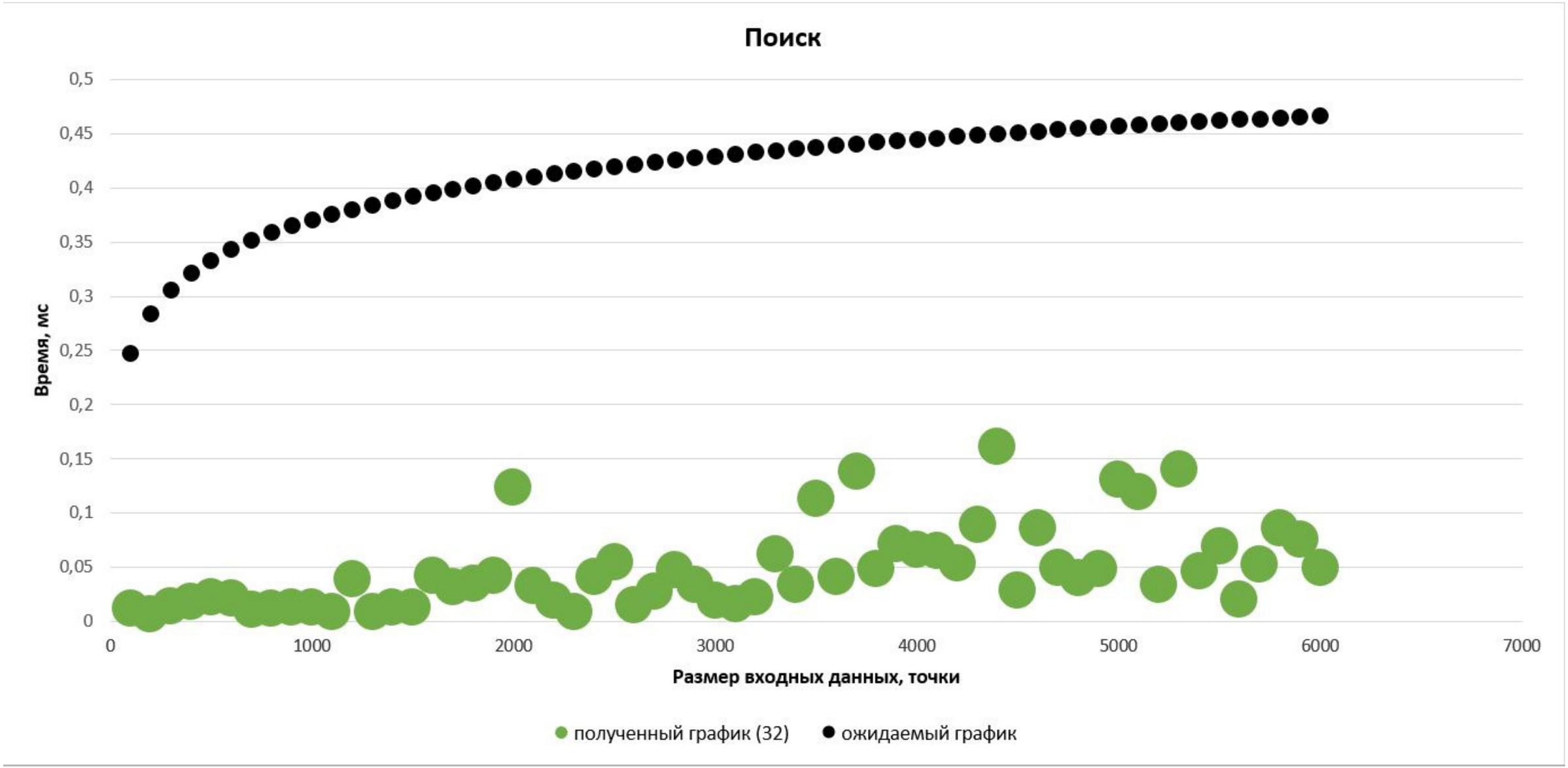

Удаление
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (16)
ожидаемый график

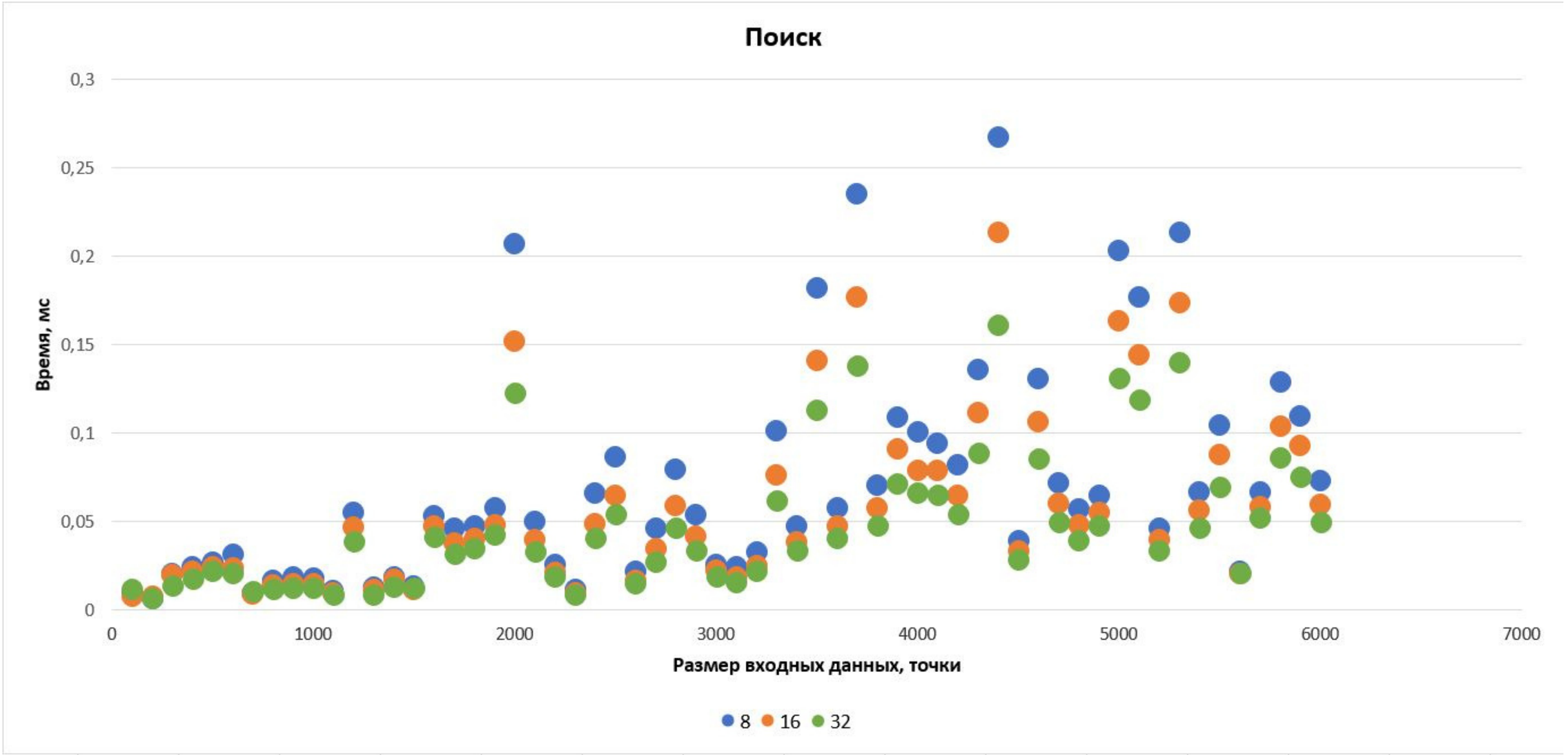

Удаление
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (32)
ожидаемый график


Удаление
0,06
0,05
0,04
0,03
0,02
0,01
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
8
16
32


Поиск
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график
ожидаемый график

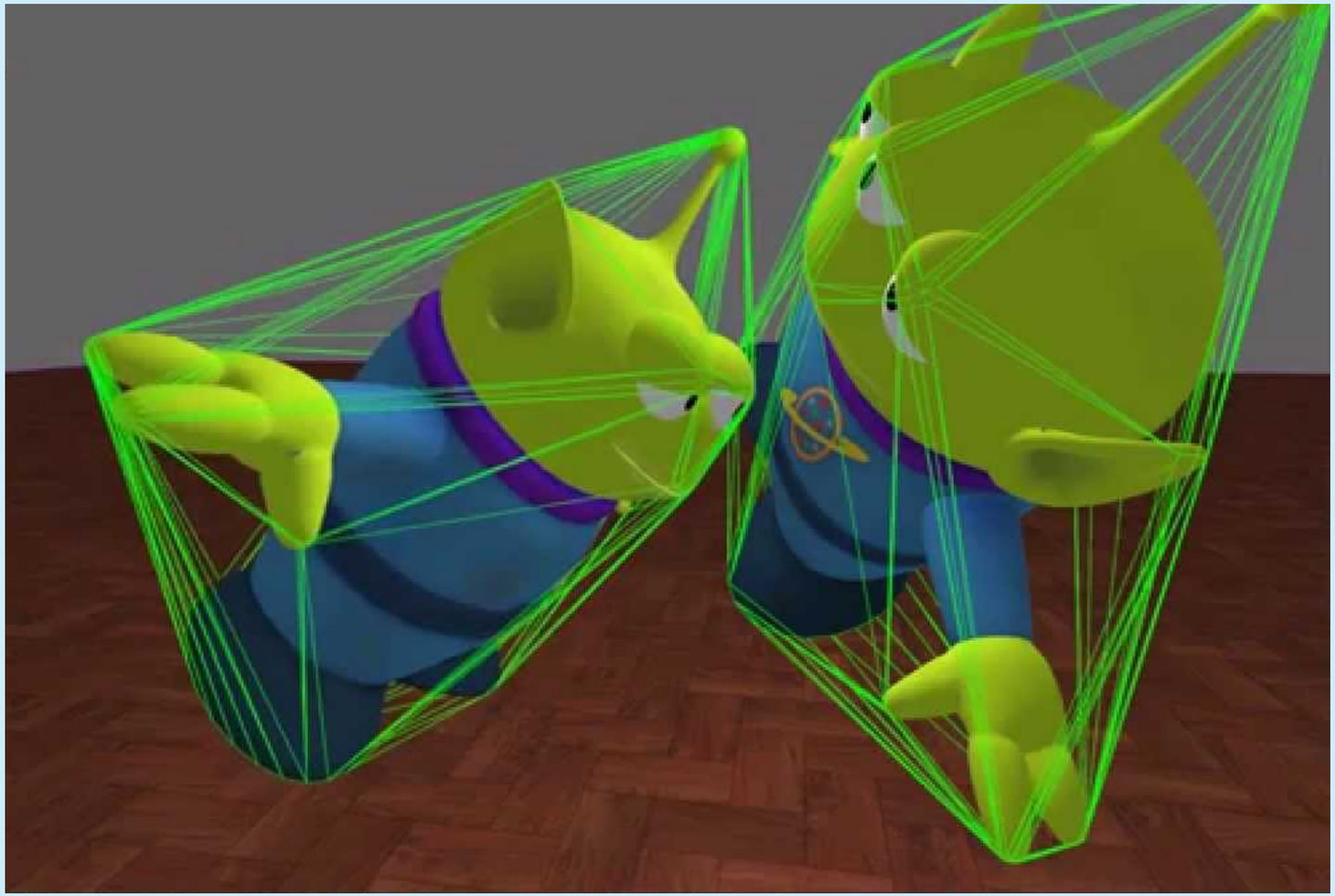

Поиск
0,5
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (16)
ожидаемый график


Поиск
0,5
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
полученный график (32)
ожидаемый график


Поиск
0,3
0,25
0,2
0,15
0,1
0,05
0
Время, мс
0
1000
2000
3000
4000
5000
6000
7000
Размер входных данных, точки
8
16
32

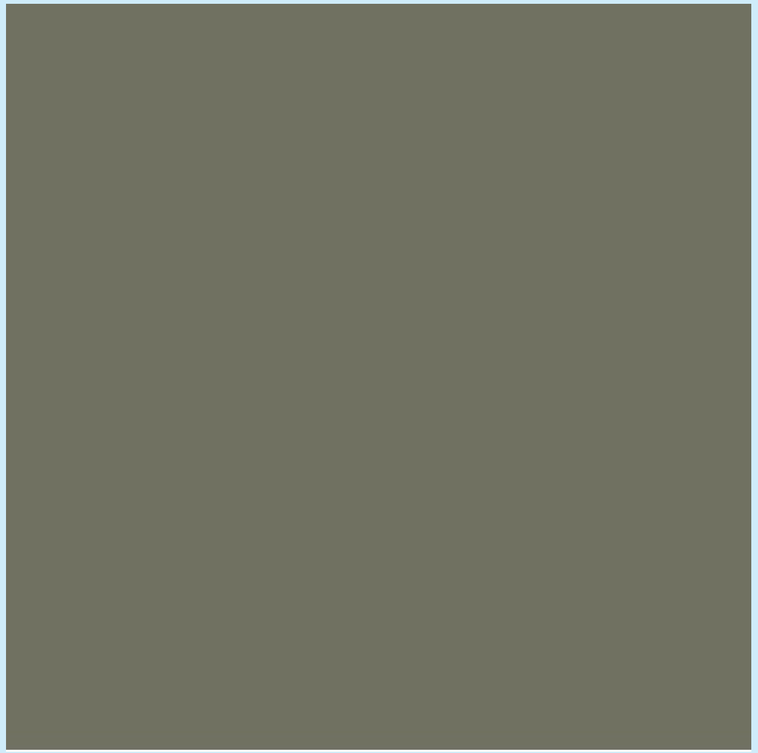
Применимость:

Collision detection algorithms (обнаружение пересечений между собой двух и более объектов) в GameDev, симуляции физических процессов



Применимость:

Сжатие изображений, где каждый узел содержит «средний» цвет каждого дочернего узла (чем глубже продвигаемся по дереву, тем более детальное изображение получаем)



Применимость:

Сжатие изображений, где каждый узел содержит «средний» цвет каждого дочернего узла (чем глубже продвигаемся по дереву, тем более детальное изображение получаем)

Применимость:

Сжатие изображений, где каждый узел содержит «средний» цвет каждого дочернего узла (чем глубже продвигаемся по дереву, тем более детальное изображение получаем)

Применимость:

Сжатие изображений, где каждый узел содержит «средний» цвет каждого дочернего узла (чем глубже продвигаемся по дереву, тем более детальное изображение получаем)

Применимость:

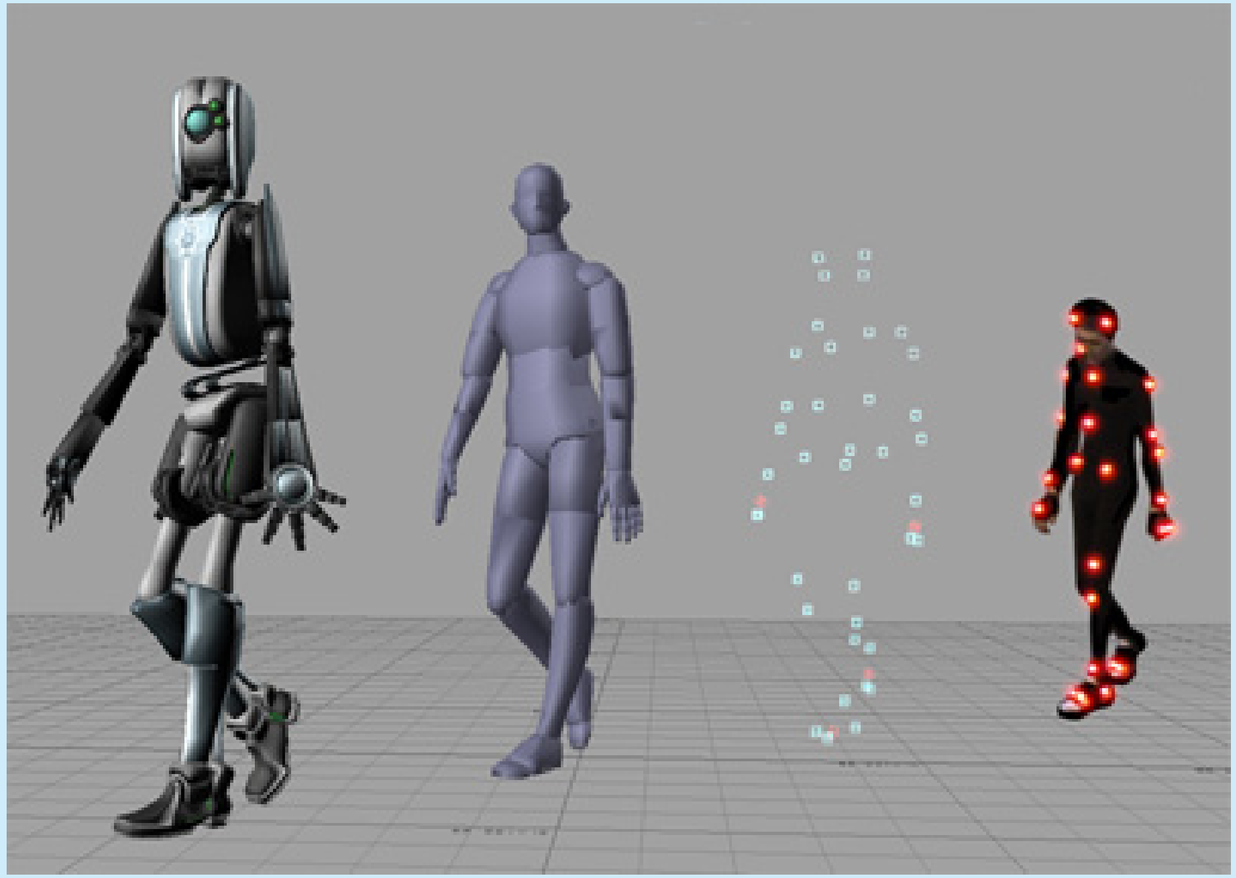
Сжатие изображений, где каждый узел содержит «средний» цвет каждого дочернего узла (чем глубже продвигаемся по дереву, тем более детальное изображение получаем)

Применимость:

Сжатие изображений, где каждый узел содержит «средний» цвет каждого дочернего узла (чем глубже продвигаемся по дереву, тем более детальное изображение получаем)

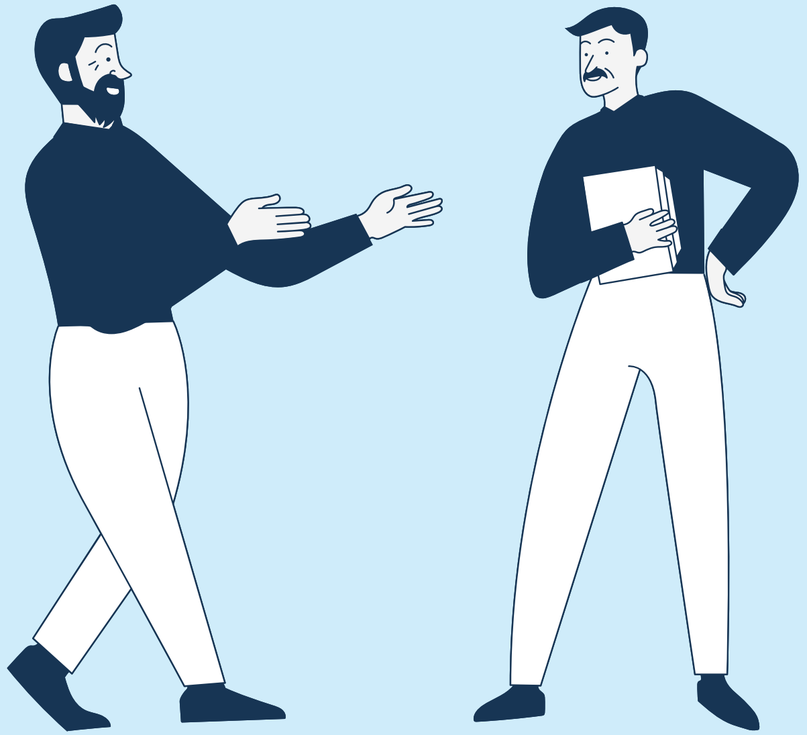
Применимость:

Определение скрытой поверхности (процесс определения того, какие поверхности и части поверхностей можно увидеть под определенным углом обзора) в 3D моделировании, VR и GameDev



Применимость:

- Вычисления, связанные с многомерными полями (в вычислительной гидродинамике, электромагнетизме)
- Симуляция игры «Жизнь»
- Пространственные базы данных – базы данных, оптимизированные для хранения и выполнения запросов к данным о пространственных объектах, представленных некоторыми абстракциями: точка, линия, многоугольник и им подобных
- Хранение данных для табличных или матричных вычислений



Плюсы:

- Простая в реализации и эффективная в использовании структура данных
- Быстро строится и значительно уменьшает количество перебираемых объектов, что увеличивает скорость и производительность алгоритмов, приложений, использующих рассматриваемую структуру данных

Минусы:

- Не константное время основных операций, что предоставляют другие структуры данных
- Деревья квадрантов не всегда является лучшей структурой данных для collision detection. Вместо них могут быть использованы, к примеру, разреженная сетка (sparse grid) и Zomorodian and Edelsbrunner’s algorithm, которые предоставляют лучшую производительность

Спасибо за внимание!!!