**Лабораторная работа №1.**

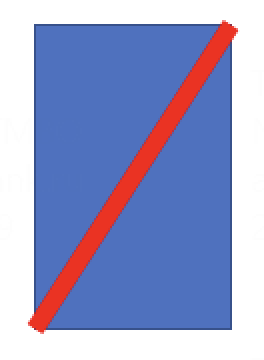
**Описание:**

Задание данной лабораторной работы заключалось в проверке наличия и нахождения элемента в двумерном массиве, отсортированном и по столбцам, и по строкам.

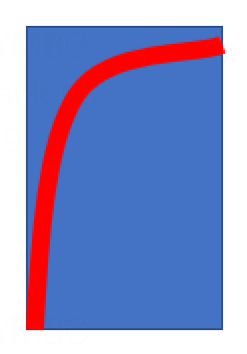
**Задача:**

Была поставлена задача проверить и сравнить время работы трех различных алгоритмов: линейного поиска (Linear search), бинарного поиска (Binary search) и экспоненциального поиска (Exponential search) на двух наборах входных данных.

* Первый набор (Ordinary table generation) – генерация данных, направленная на то, чтобы элементы, близкие по значению к тому, который нужно найти (target), находились на побочной диагонали матрицы.



* Второй набор (Hyperbolic table generation) – генерация данных, в которой элементы, близкие к target, находятся по индексам, примерно задающим гиперболу см. рисунок).



**Ход работы:**

Весь код был написан на языке C++, в среде разработки CLion. Тесты проводились на платформе Apple MacBook на процессоре M1. Все файлы с кодом находятся в [репозитории](https://github.com/Kaparya/Algorithms-and-Data-structures.git) на GitHub.

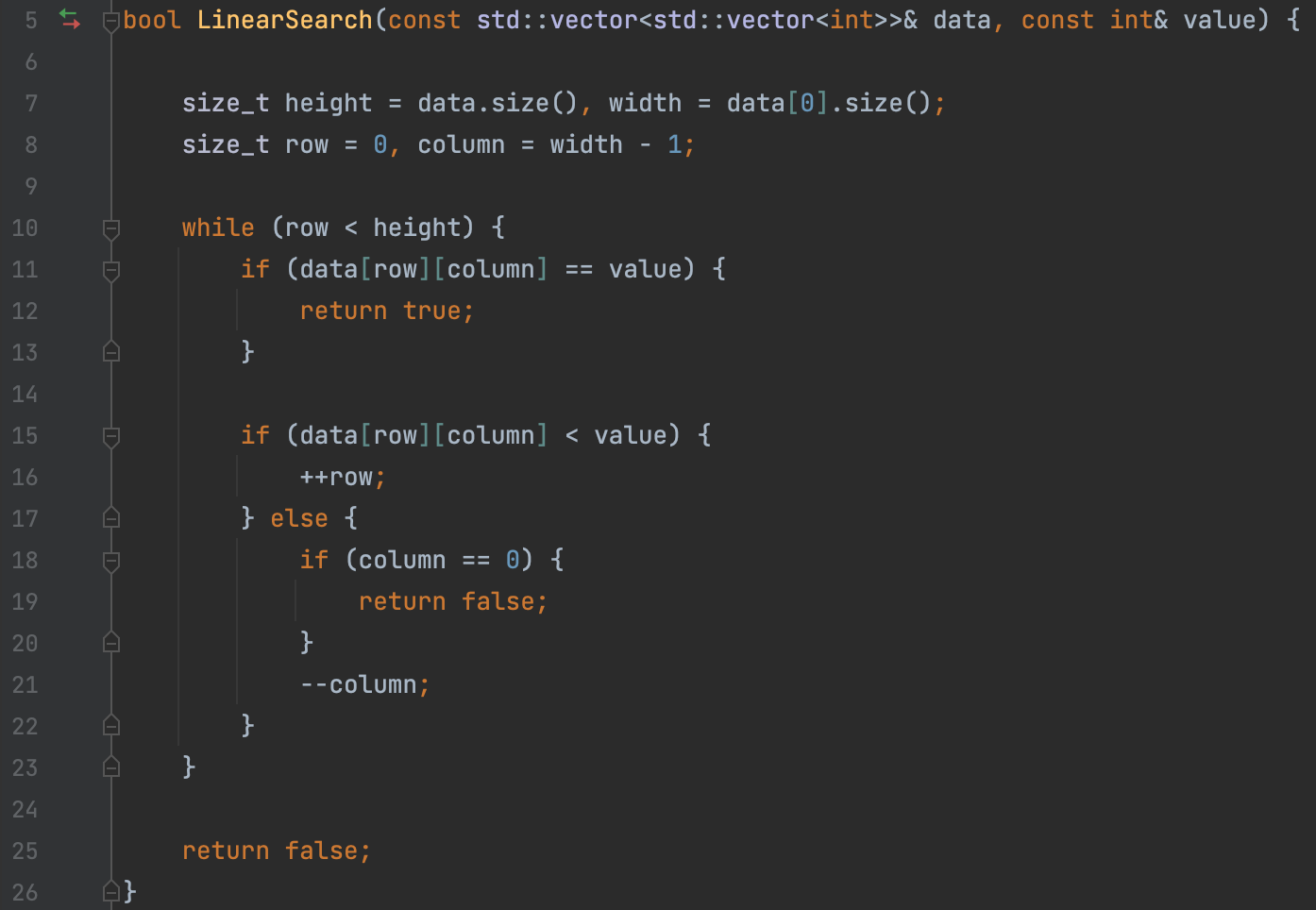
1. Были реализованы все функции поиска:

На вход функции принимали текущую таблицу значений (двумерный вектор) и элемент, который требуется найти (value). Результатом работы является значение bool – найден элемент или не найден.

* Linear search – поиск элемента, путем перехода к соседнему элементу. Начало в правом верхнем углу. Существует три случая:
  + текущий элемент равен target => мы нашли нужный элемент, заканчиваем алгоритм;
  + текущий элемент больше target: так как строки отсортированы по неубыванию, требуется проверить элемент, находящийся левее текущего => переходим на элемент, находящийся слева от текущего;
  + текущий элемент меньше target: заметим, что на прошлом шаге мы сдвигались влево до момента, когда текущий элемент не окажется меньше либо равен текущему => если target не найден, значит в этой строке он отсутствует. Столбцы отсортированы по возрастанию, а значит элементы больше текущего находятся ниже него => переходим к следующей строке.

Сдвигаем текущий элемент до тех пор, пока его индексы не выйдут за границы (станут меньше нуля, либо станут больше или равны размера столбца / строки).

**Реализация:**

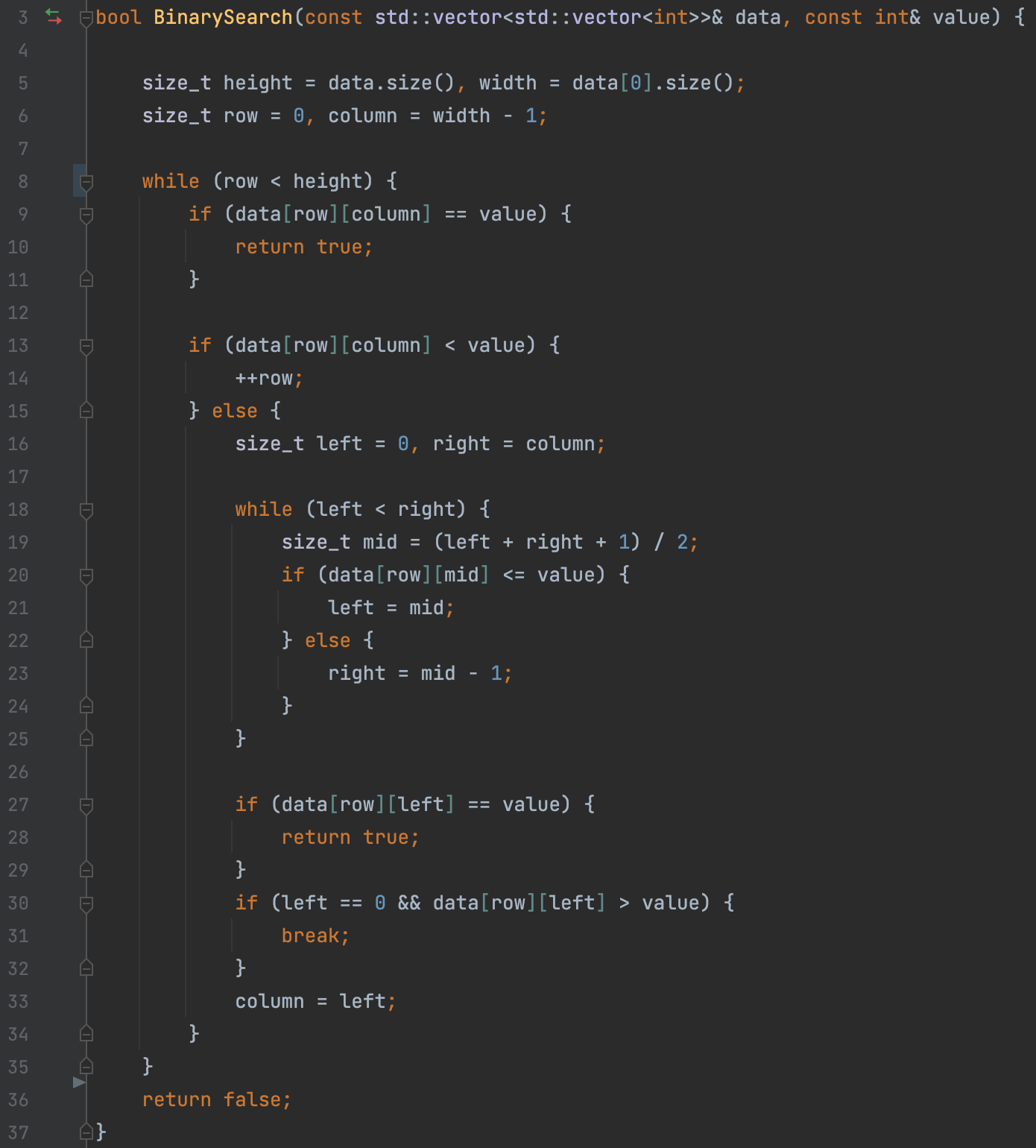
****

-----------------------------------------------------------------------------------------------------------------------

* Binary search – поиск элемента похож на Linear search, однако сдвиг по строке влево теперь происходит путем перехода на ближайший элемент, найденный **бинарным** поиском.Существуют те же три случая, первый и третий аналогичны Linear search, отличается второй:
  + текущий элемент больше target: так как строки отсортированы по неубыванию, требуется проверить элементы, находящиеся левее текущего => запускаем бинарный поиск, где левая граница – нуль (минимальный возможны индекс), а правая – текущий столбец. Ищем элемент ближайший по значению к target и меньший чем он.

Сдвигаем текущий элемент до тех пор, пока его индексы не выйдут за границы (станут меньше нуля, либо станут больше или равны размера столбца / строки) или бинарный поиск не остановится на нулевом элементе, значение которого больше значения target (случай, когда по строке сдвигались влево до тех пор, пока не вышли за границу).

**Релизация:**

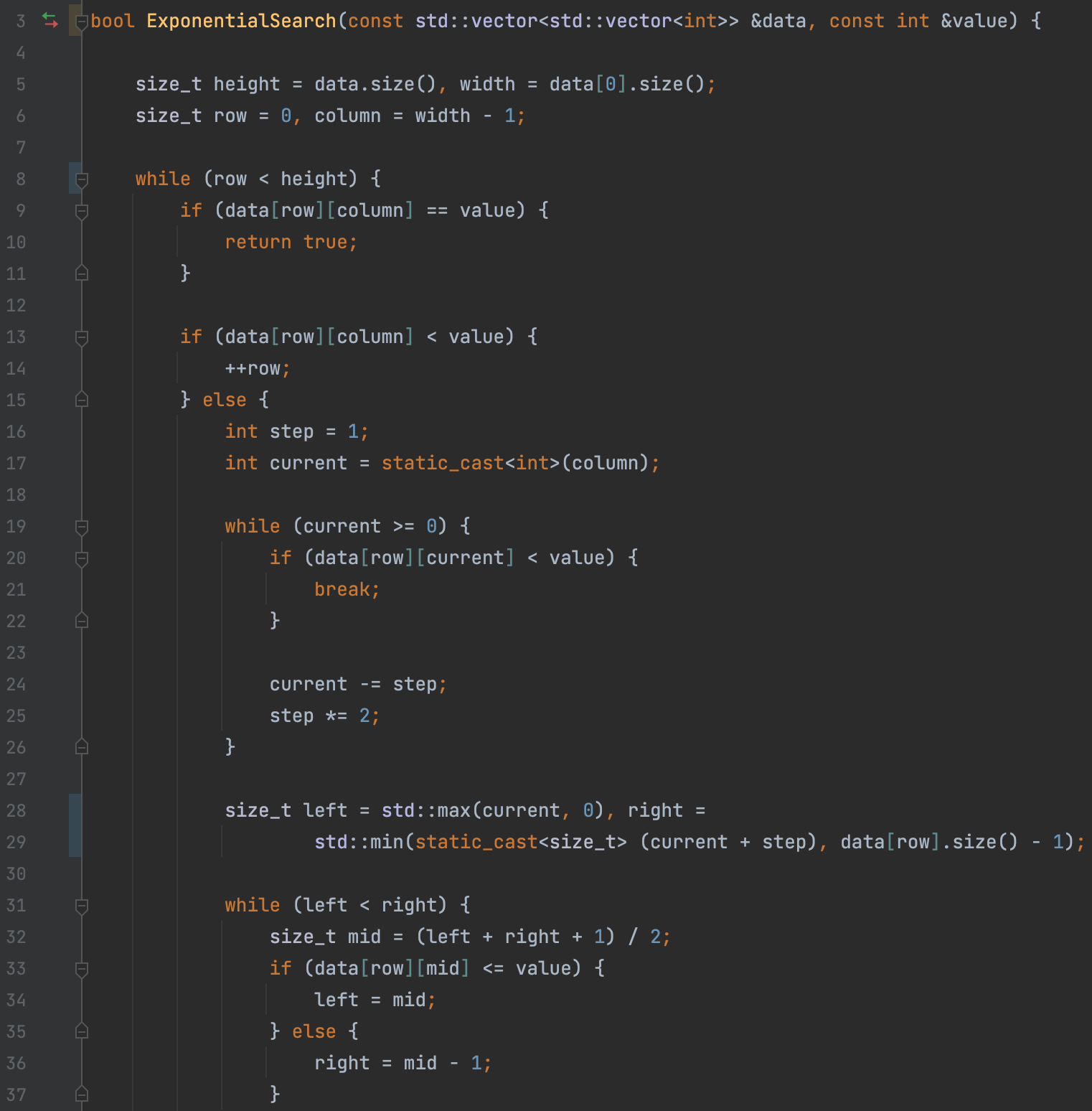


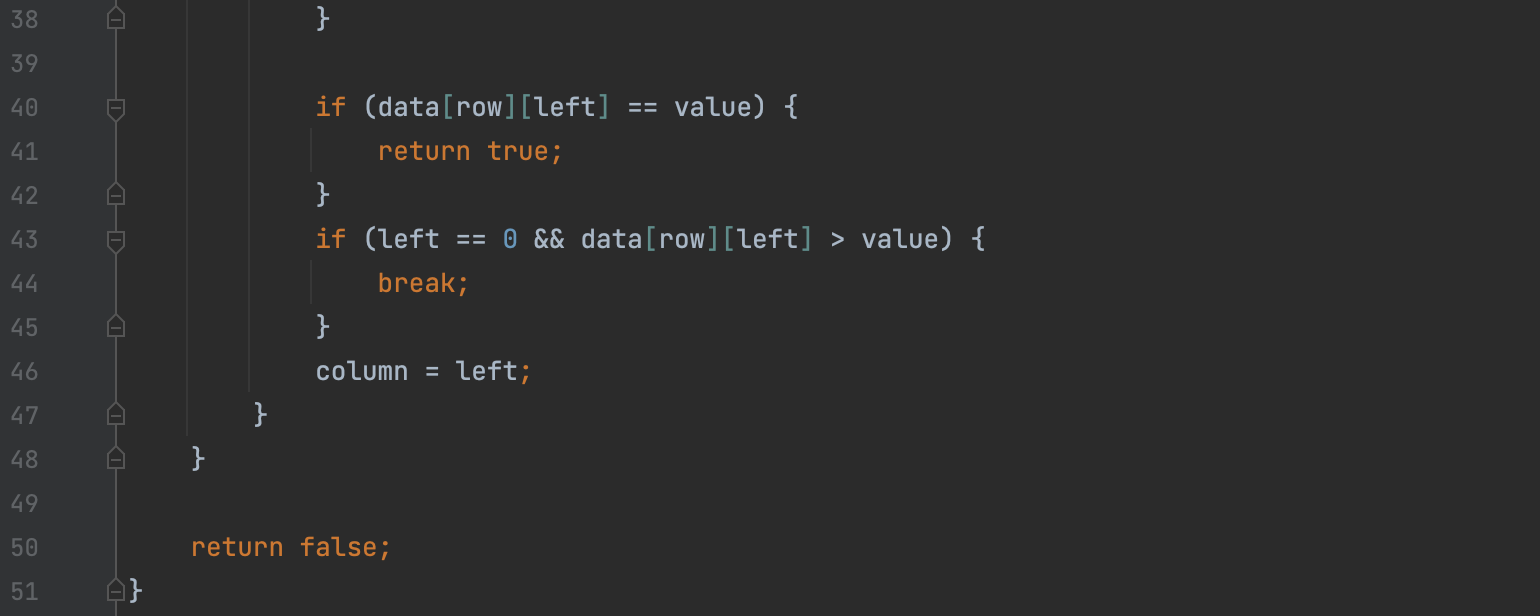
-----------------------------------------------------------------------------------------------------------------------

* Exponential search – поиск элемента похож на Linear search, однако сдвиг по строке влево теперь происходит путем перехода на ближайший элемент, найденный **экспоненциальным** поиском.Существуют те же три случая, первый и третий аналогичны Linear search, отличается второй:
  + текущий элемент больше target: так как строки отсортированы по неубыванию, требуется проверить элементы, находящиеся левее текущего => запускаем **экспоненциальный** поиск с текущего столбца и сдвигаемся на текущую степень двойки до тех пор, пока индекс не окажется меньше нуля или элемент по этому индексу не станет меньше target. Затем запускаем бинарный поиск, где найденный индекс – левая граница, а правая – найденный индекс + текущая степень двойки (самая обычная реализация экспоненциального поиска).

Сдвигаем текущий элемент до тех пор, пока его индексы не выйдут за границы (станут меньше нуля, либо станут больше или равны размера столбца / строки) или экспоненциальный поиск не остановится на нулевом элементе, значение которого больше значения target (случай, когда по строке сдвигались влево до тех пор, пока не вышли за границу).

**Релизация:**

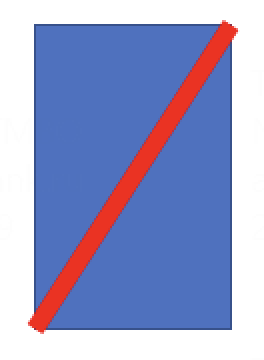




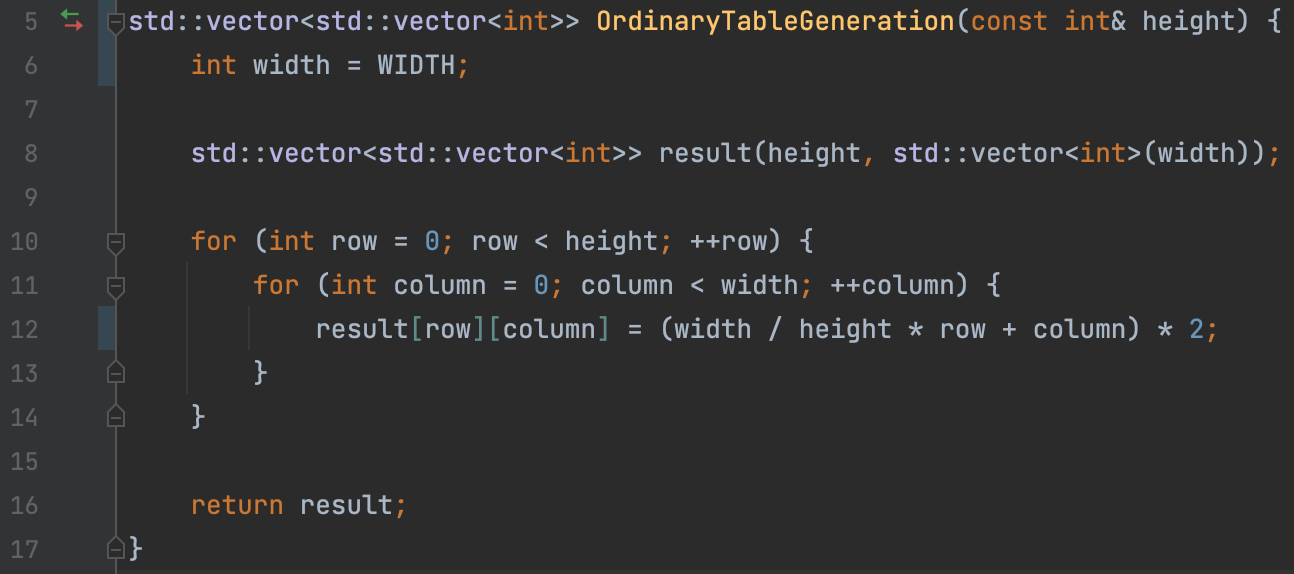
1. Были реализованы функции генерации данных.

На вход функция принимает значение текущей ширины таблицы, а высота задана нам по условию (она вынесена в константу WIDTH = 2^13). Результатом работы является двумерный вектор, в котором лежит сформированная таблица.

* Ordinary table generation – генерация, в которой элементы близкие к target находятся на побочной диагонали.

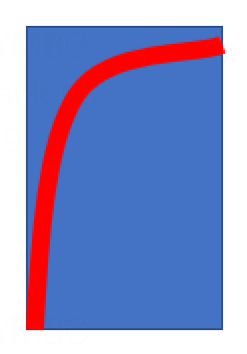


**Реализация:**

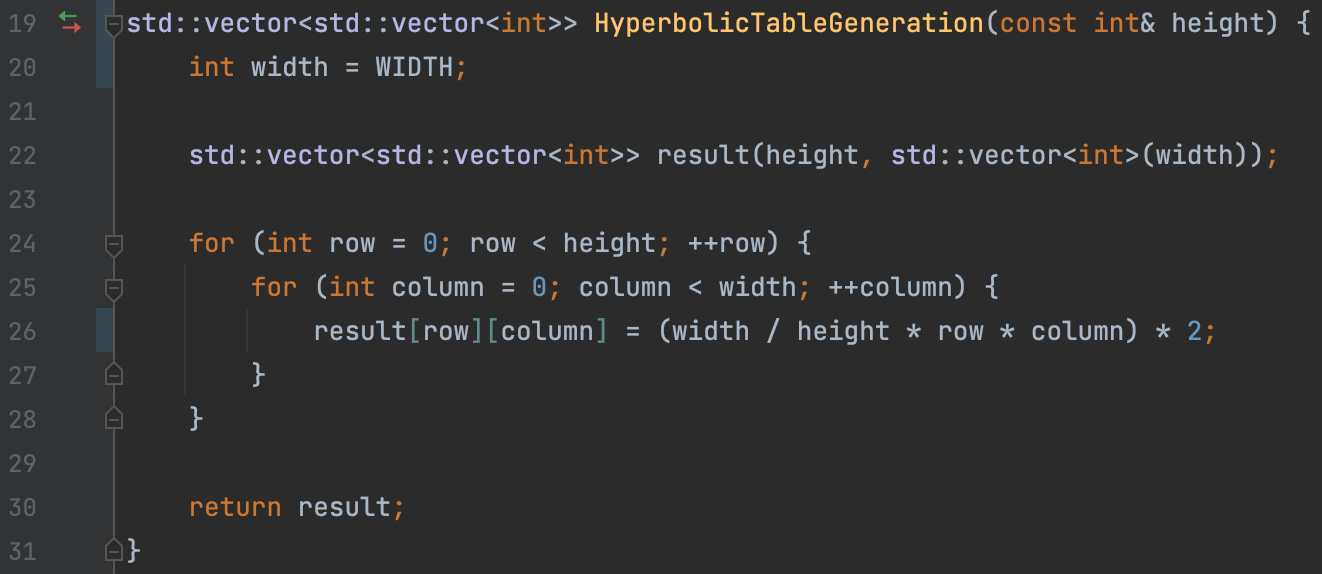
****

**-----------------------------------------------------------------------------------------------------------------------**

* Hyperbolic table generation – генерация данных, в которой элементы, близкие к target, находятся по индексам, примерно задающим гиперболу (см. рисунок).



**Реализация:**

****

1. **Результаты запусков (все значения представлены в наносекундах).**

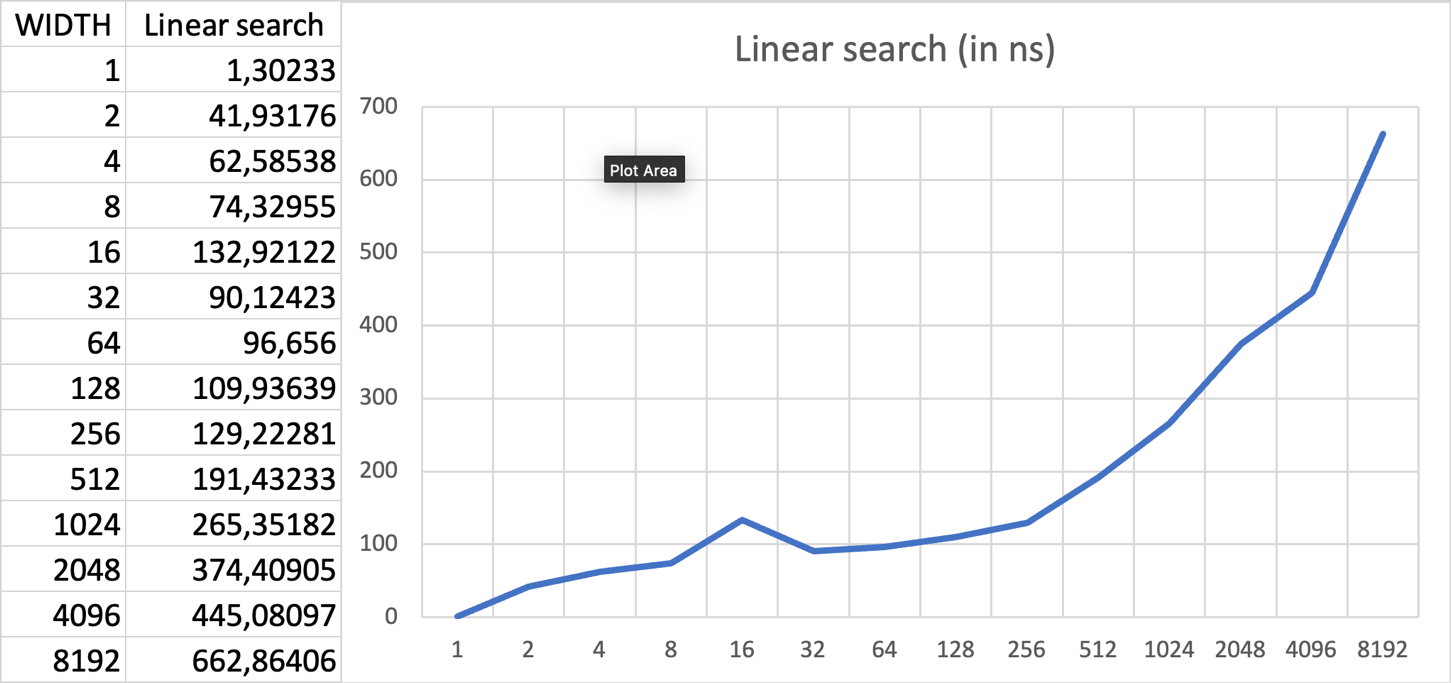
Для каждой генерации данных были запущены каждый из трех алгоритмов и было замерено время.

**Ordinary table generation.**

* Linear search for Ordinary table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

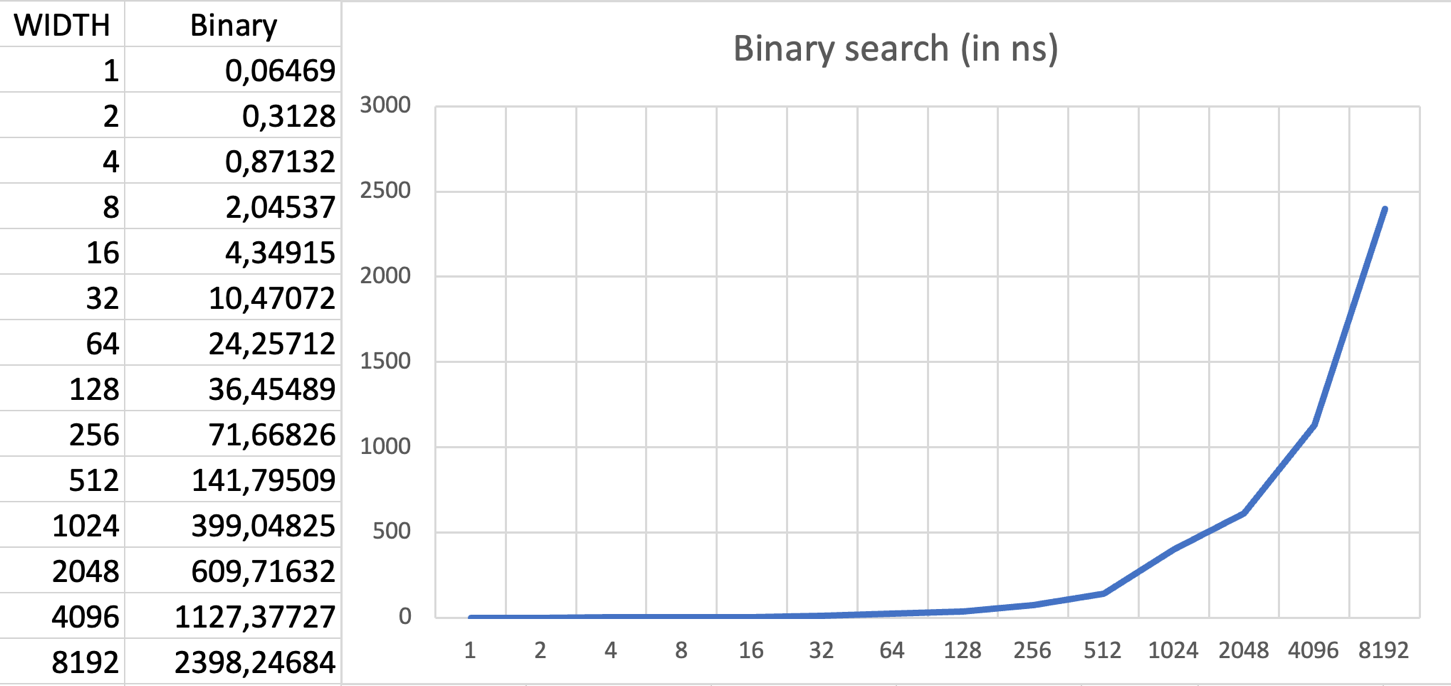
Можно заметить, что график практически стабильно неубывает (существуют неточности, вызванные погрешностями в измерениях). Однако при больших значениях время начинает резко возрастать в геометрической прогрессии.



* Binary search for Ordinary table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

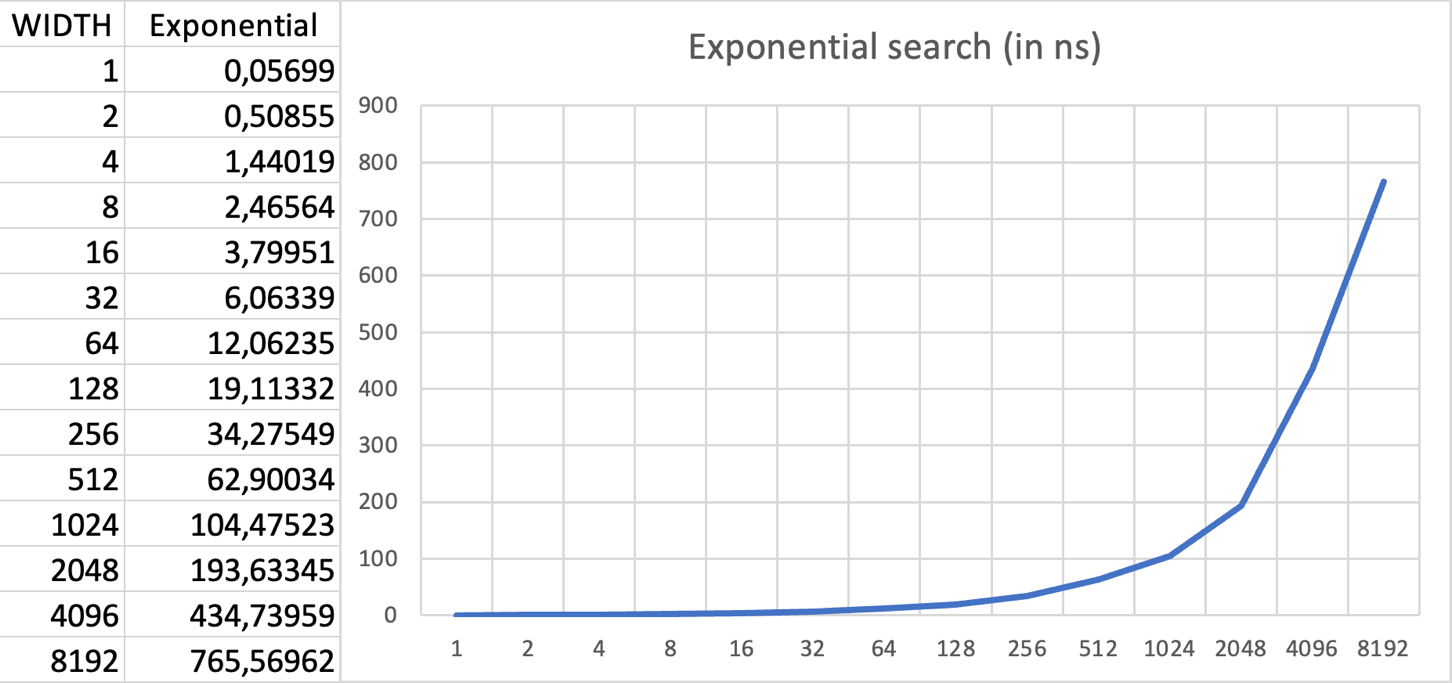
График лежит ниже, чем график Linear search, для маленьких значений, однако заметен резкий рост для больших значений.



* Exponential search for Ordinary table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

Поведение графика похоже на Binary search. Однако для больших значений время растет не так быстро.



* Linear vs Binary vs Exponential search for Ordinary table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

Можно заметить, что для маленьких данных оптимальным вариантом является Exponential search, однако на больших данных он уступает Linear search.

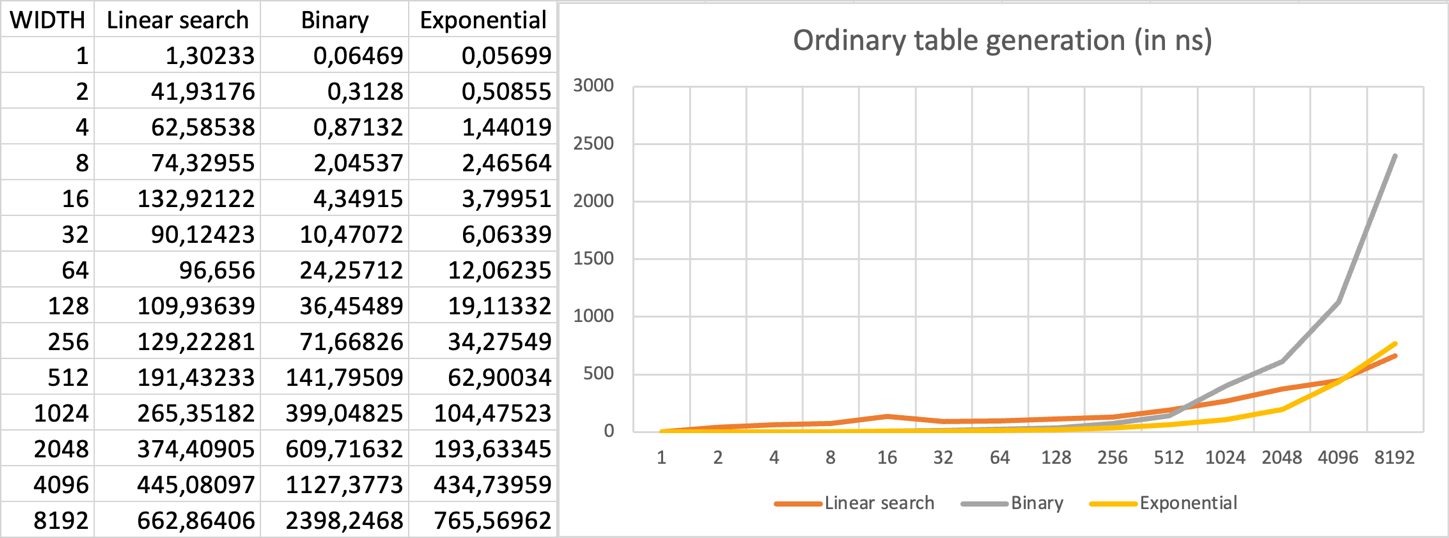
Binary search – самый медленный.

Причина таких результатов заключается в том, что при такой генерации (элементы близкие по значению target лежат на побочной диагонали) мы чередуем шаги влево и вниз. Соответственно, вниз каждый алгоритм переходит за O(1), а влево – все по разному.

Linear search сдвигается на 1 элемент влево за O(1).

Binary search за O(log2(current\_width)), где current\_width – это количество элементов левее текущего.

Exponential search тоже за О(1), однако засчет большого количества проверок и операций константа тут больше, что становится заметно на больших тестах.



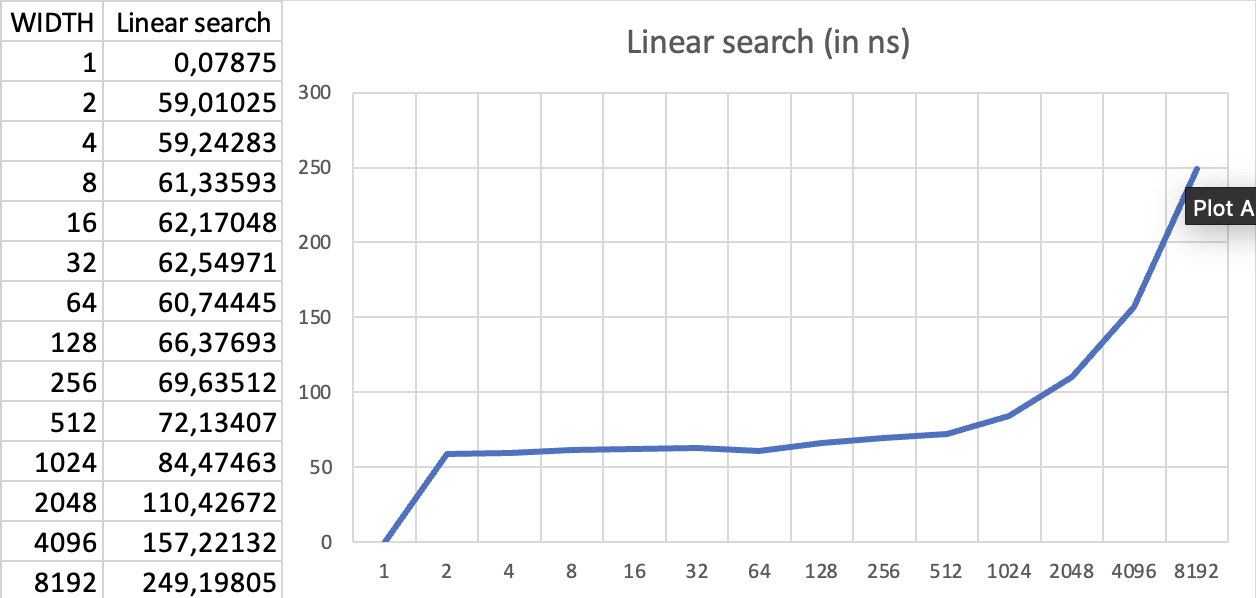
**Hyperbolic table generation.**

* Linear search for Hyperbolic table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

Заметен резкий скачок во времени в самом начале, происходящий из-за количества операций сдвига влево (необходимо пройти практически всю таблицу). Заметим, что и сложность этого алгоритма составляет O(height + width).

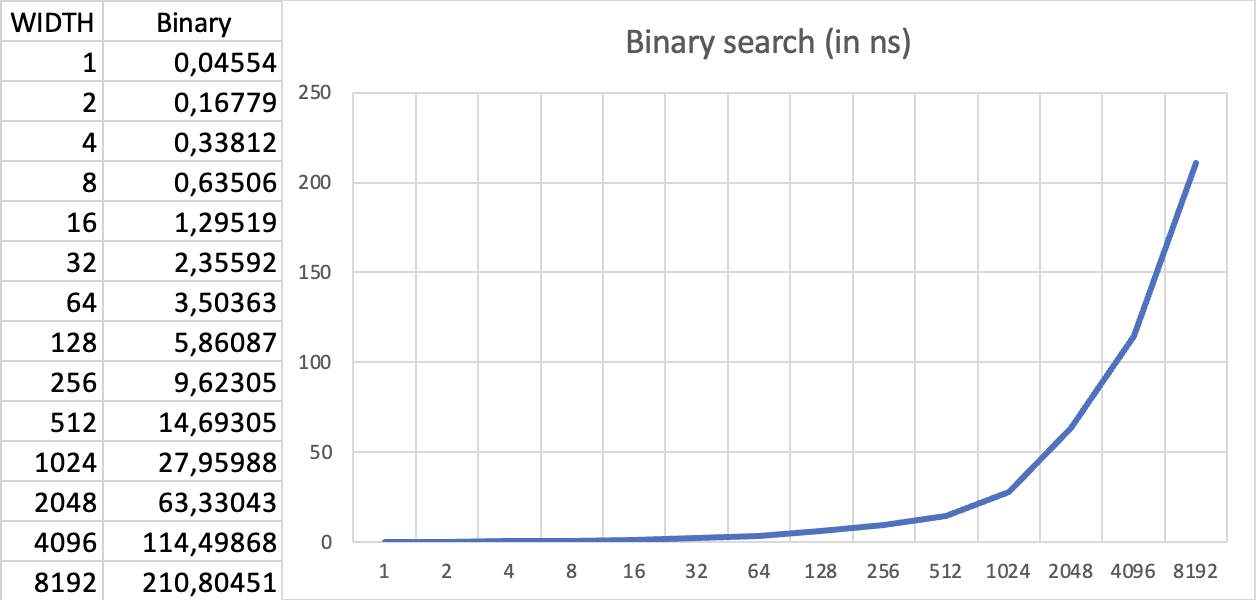
Затем график возрастает.



* Binary search for Hyperbolic table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

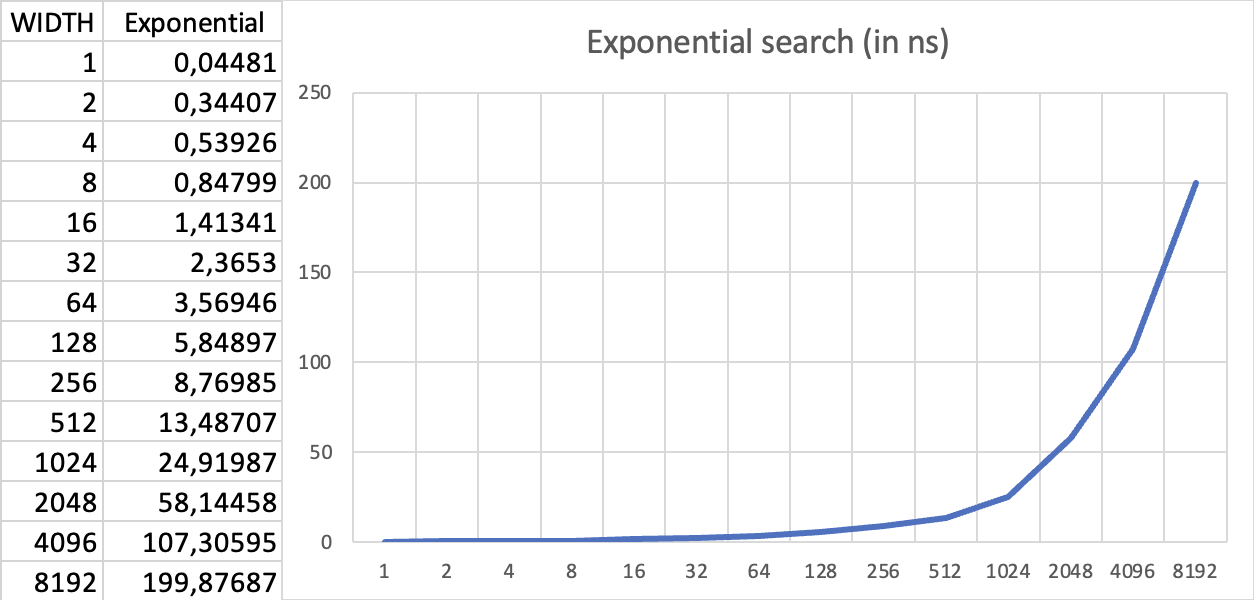
В случае Binary search, все идет стабильно – для маленьких значений алгоритм работает быстро, для больших – дольше.



* Exponential search for Hyperbolic table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

График похож на график Binary search, однако он возрастает чуть медленнее.



* Linear vs Binary vs Exponential search for Hyperbolic table generation

В таблице слева представлены значения ширины таблицы и время в наносекундах, соответствующее данной ширине. Справа – график, показывающий зависимость времени от ширины таблицы.

Заметим, что ситуация в случае Hyperbolic table generation показывает обычное соотношение времени для Linear, Binary and Exponential searches, основанное на вычислительной сложности.

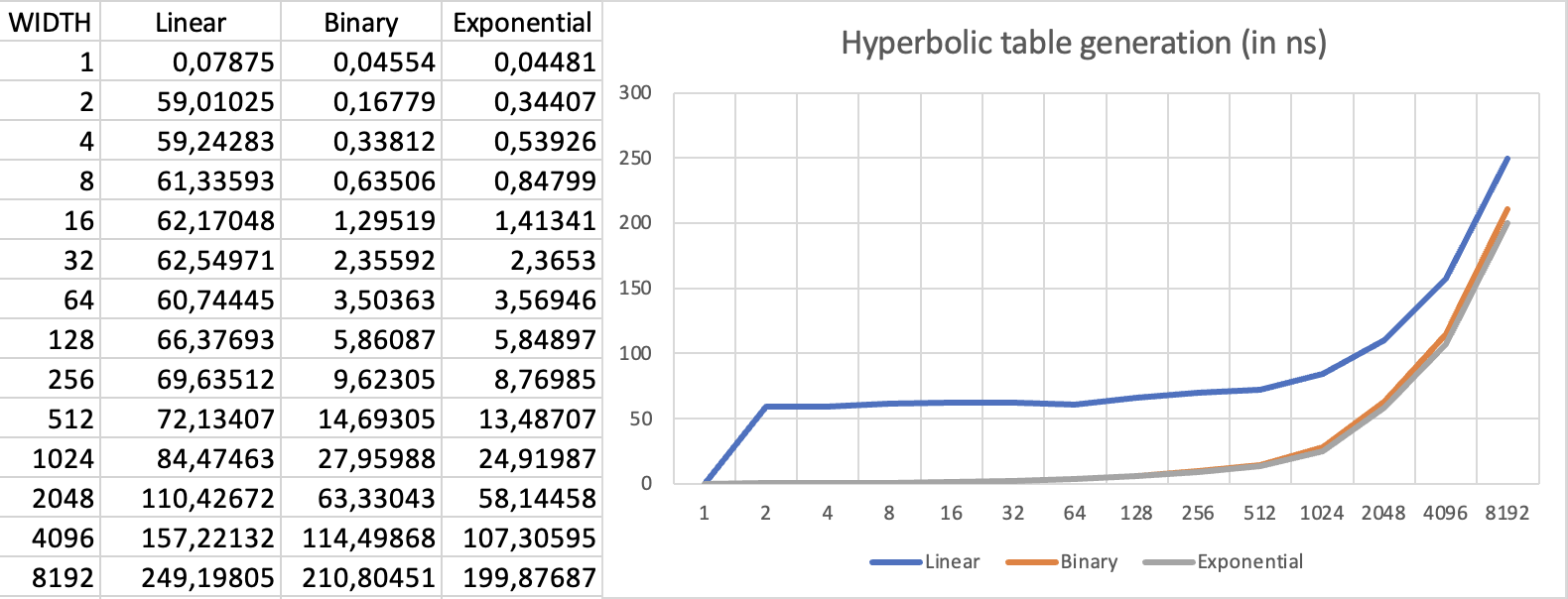
Linear search проходит всю таблицу слева направо по одному элементу => O(width).

Binary search проходит всю таблицу слева направо за O(log2(width)).

Exponential search проходит всю таблицу слева направо чуть меньше O(log2(width)) – примерно O(log2(width / height)).

O(width) > O(log2(width)) > O(log2(width / height))

График является подтверждением этому.



1. **Выводы.**

В данной лабораторной работе мы протестировали и сравнили по времени три алгоритма поиска элемента в таблице (Linear, Binary and Exponential search), отсортированной и по строкам, и по столбцам. Данные генерировались двумя способами: Ordinary and Hyperbolic generation.

Были получены следующие результаты:

* На маленьких тестах Linear search проигрывает Binary and Exponential search.
* Linear search, несмотря на большую complexity = O(n + m), показывает неплохие результаты и может обогнать и Binary, и Exponential searches, например, на тестах, когда элементы близкие к target находятся на побочной диагонали.
* Время работы Linear search может расти как линейно, так и квадратично.
* Время работы Binary search растет пропорционально увеличению входных данных (width увеличивается в 2 раза => time увеличивается в 2 раза).
* Время работы Exponential search при увеличении width в 2 раза растет в среднем в 1,75 раз.

В дополнение, мы попрактиковались в написании проекта на C++, а также пользовании Гитом. Ссылка еще раз: <https://github.com/Kaparya/Algorithms-and-Data-structures.git>