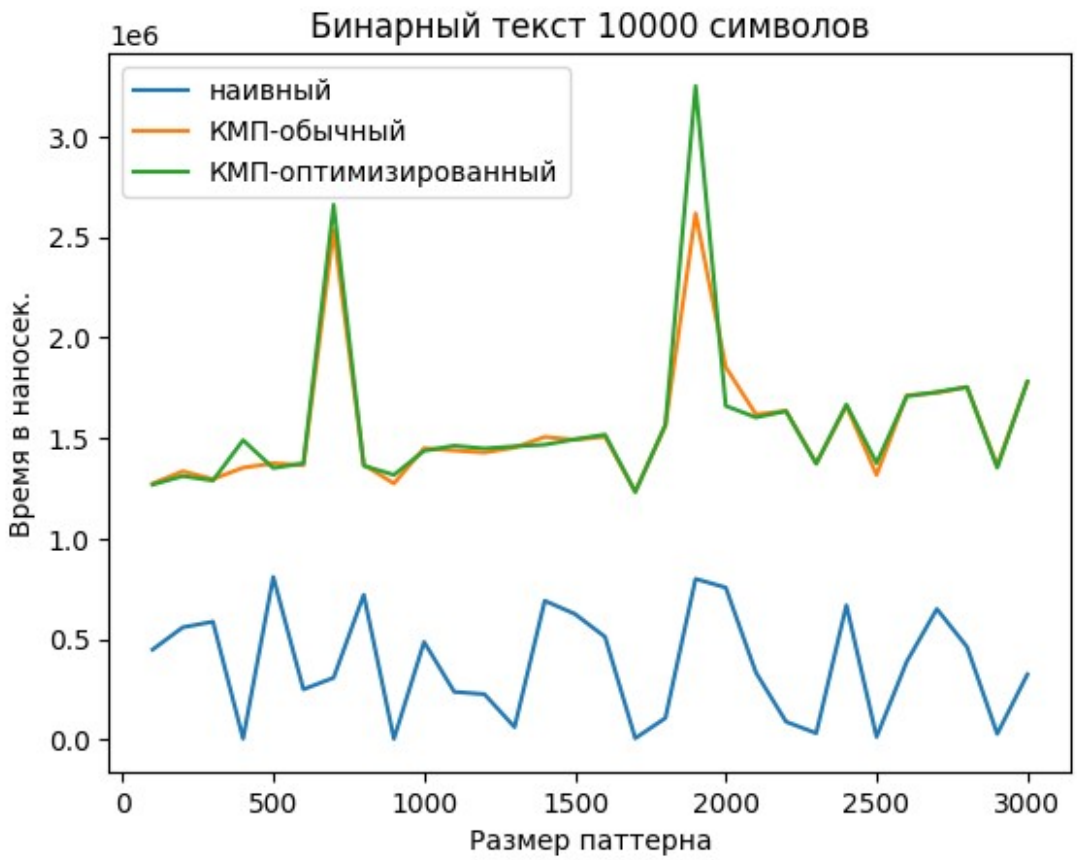
АИСД-2, 2023. Демков Михаил Кириллович БПИ212. Среда разработки - CLion

Описание программы можно посмотреть в README.md (в корне проекта)

Графики зависимости времени работы от размера шаблона

График для бинарного текста длиной 10000

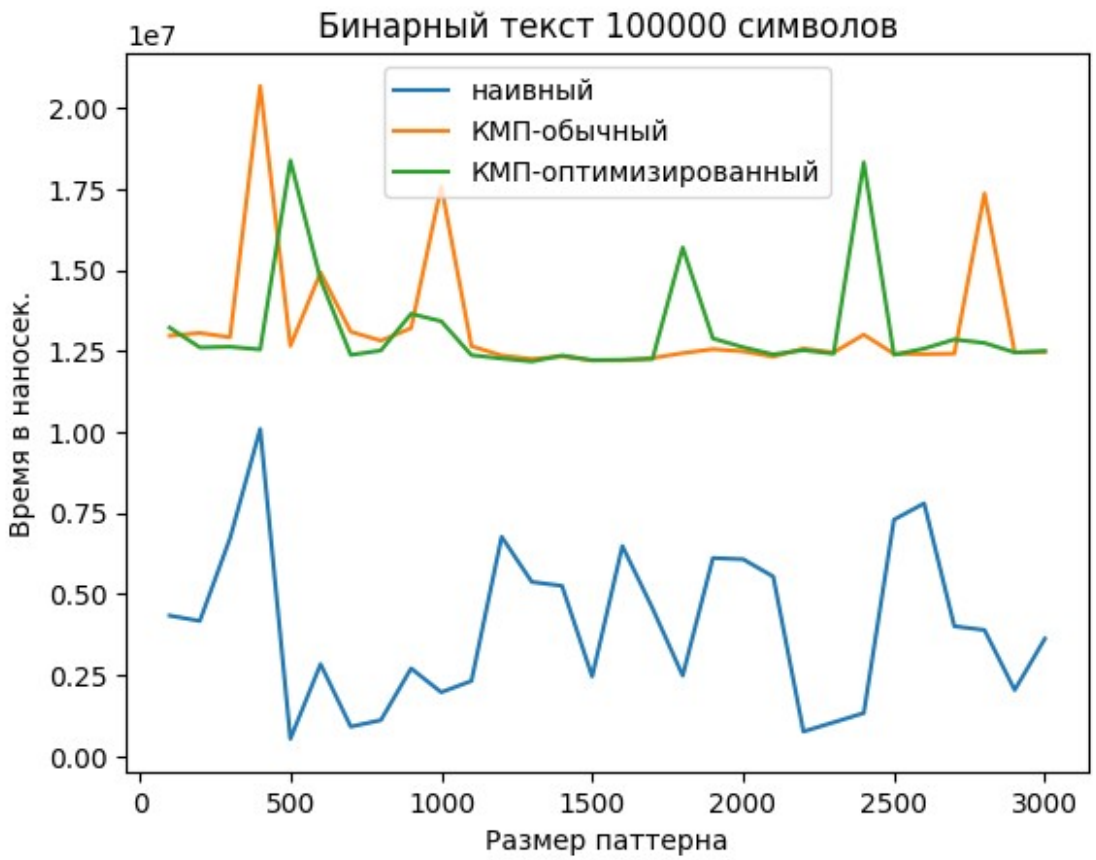
```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv("./tables/table.csv", sep=';')


binary_10000 = data[(data['Type of text'] == 'бинарный') & (data['Text
size'] == 10000)]

different_algo = binary_10000['Algorithm'].unique()

for algo in different_algo:
    cur_algo = binary_10000[binary_10000['Algorithm'] == algo]

    plt.plot(cur_algo['Pattern size'], cur_algo['Time(ns)'],
label=str(algo))

plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.title('Бинарный текст 10000 символов')

plt.legend()
plt.show()
```



Как можно заметить, наивный алгоритм работает быстрее других. Связано это с его реализацией - т.к мы ищем последнее (крайнее) вхождение подстроки в строку, наивный алгоритм реализован так, что начинает проверять с конца

График для бинарного текста длиной 100000

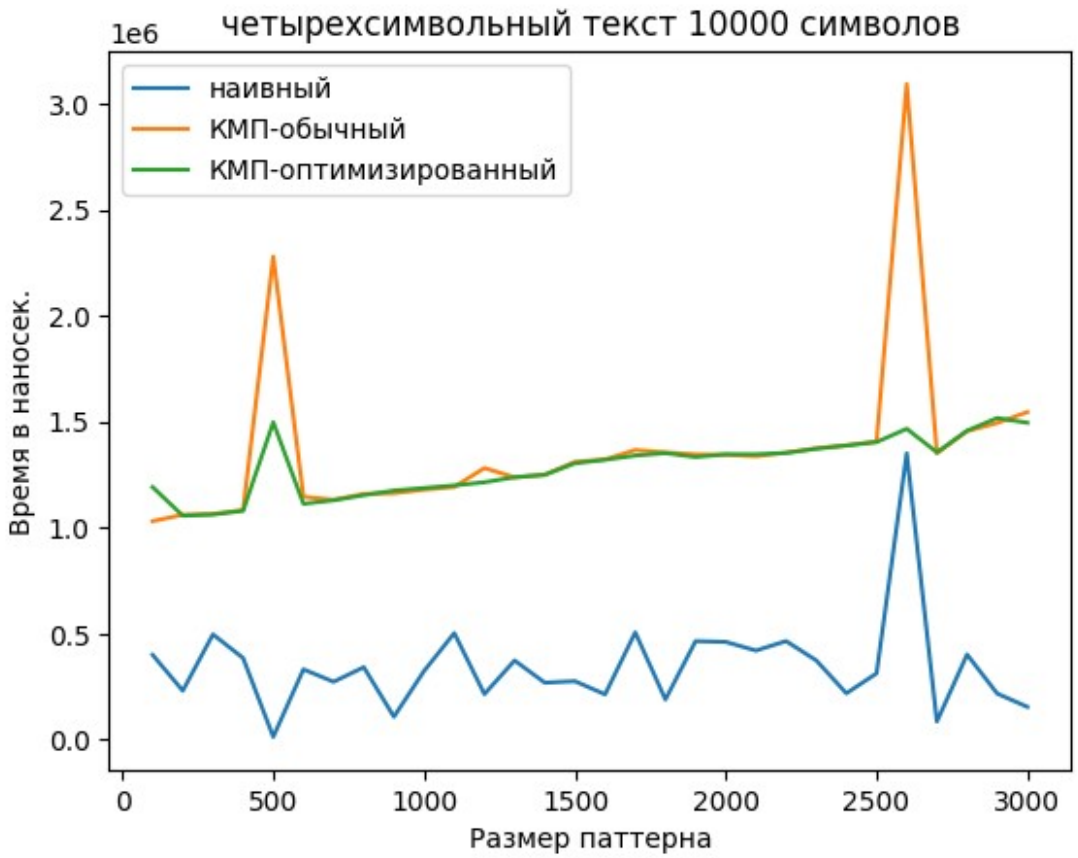
```
binary_100000 = data[(data['Type of text'] == 'бинарный') &
(data['Text size'] == 100000)]

different_algo = binary_100000['Algorithm'].unique()

for algo in different_algo:
    cur_algo = binary_100000[binary_100000['Algorithm'] == algo]

    plt.plot(cur_algo['Pattern size'], cur_algo['Time(ns)'],
label=str(algo))

plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.title('Бинарный текст 100000 символов')

plt.legend()
plt.show()
```



Здесь вывод такой же, как и на графике выше

График для четырехсимвольного текста длиной 10000

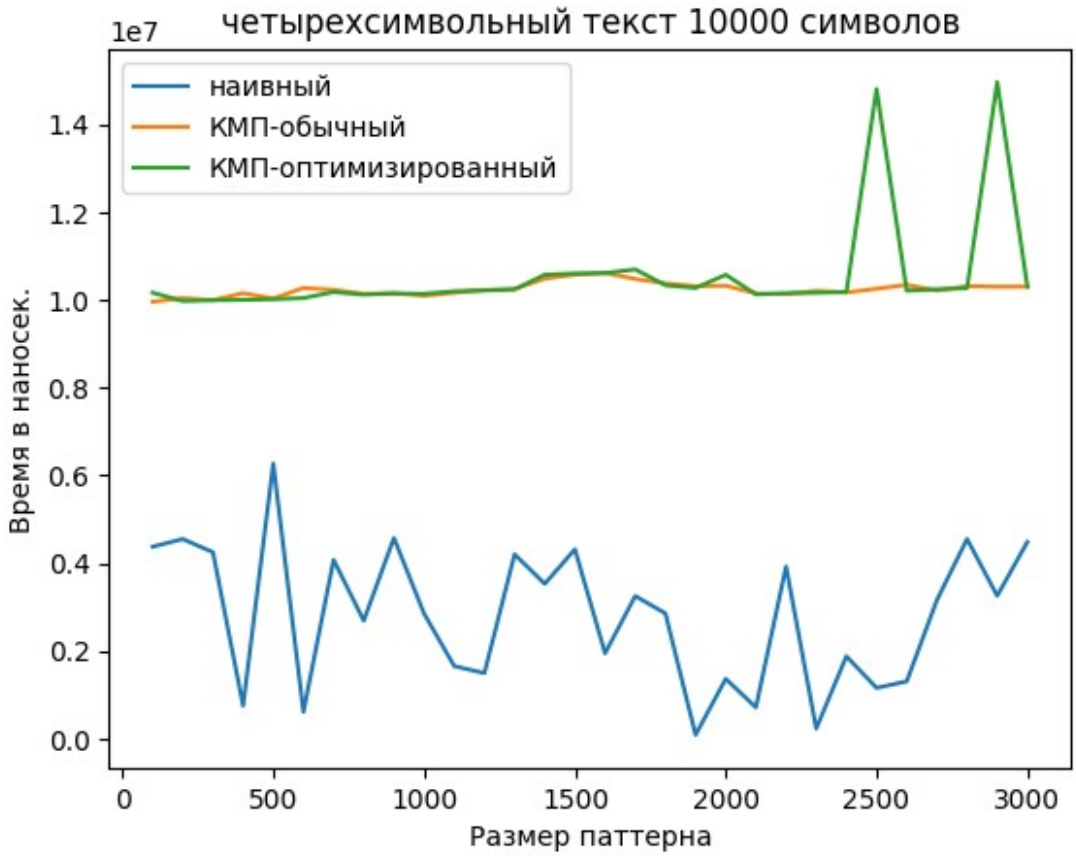
```
dnk_10000 = data[(data['Type of text'] == 'четырехсимвольный') &
(data['Text size'] == 10000)]

different_algo = dnk_10000['Algorithm'].unique()

for algo in different_algo:
    cur_algo = dnk_10000[dnk_10000['Algorithm'] == algo]

    plt.plot(cur_algo['Pattern size'], cur_algo['Time(ns)'],
label=str(algo))

plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.title('четырехсимвольный текст 10000 символов')

plt.legend()
plt.show()
```



Мало что изменилось по сравнению с бинарными алафавитом и это неудивительно

График для четырехсимвольного текста длиной 100000

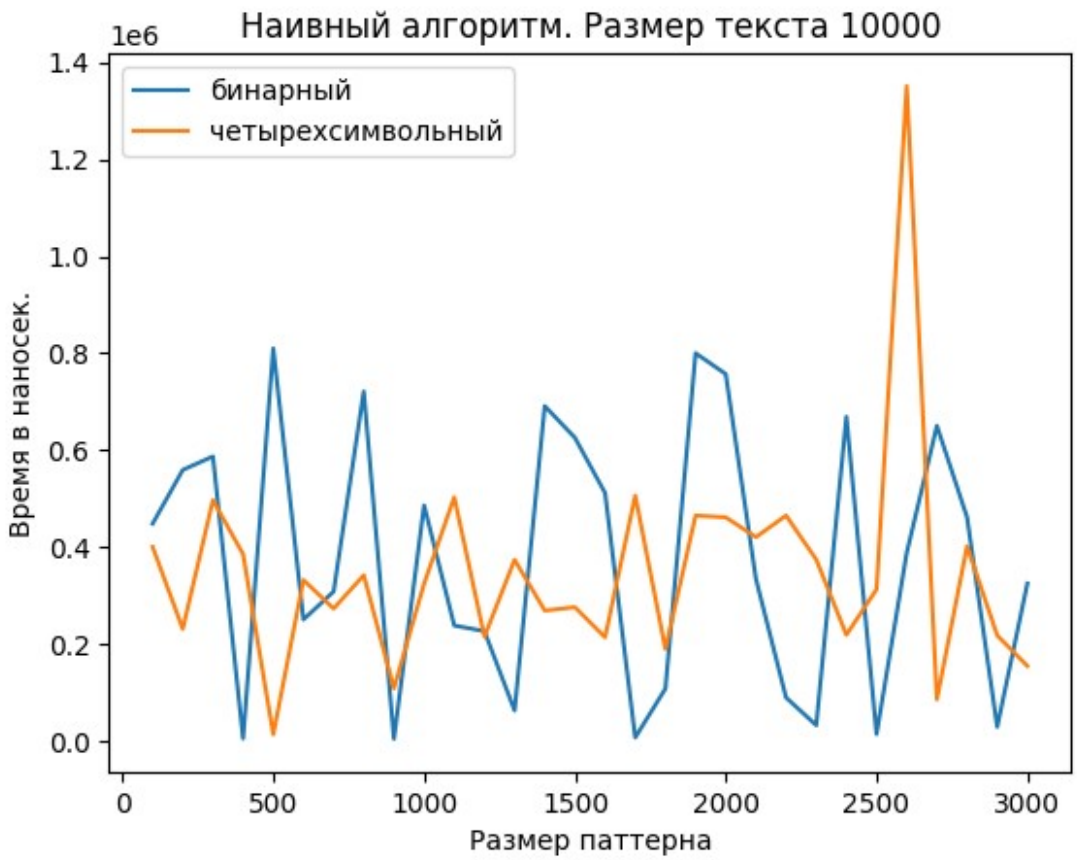
```
dnk_100000 = data[(data['Type of text'] == 'четырехсимвольный') &
(data['Text size'] == 100000)]

different_algo = dnk_100000['Algorithm'].unique()

for algo in different_algo:
    cur_algo = dnk_100000[dnk_100000['Algorithm'] == algo]

    plt.plot(cur_algo['Pattern size'], cur_algo['Time(ns)'],
label=str(algo))

plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.title('четырехсимвольный текст 10000 символов')

plt.legend()
plt.show()
```



Выбросы связаны с несовершенностью вычислительных машин. Если бы я считал количество обменов, то не было бы таких скачков

График для наивного алгоритма. Размер текста 10000

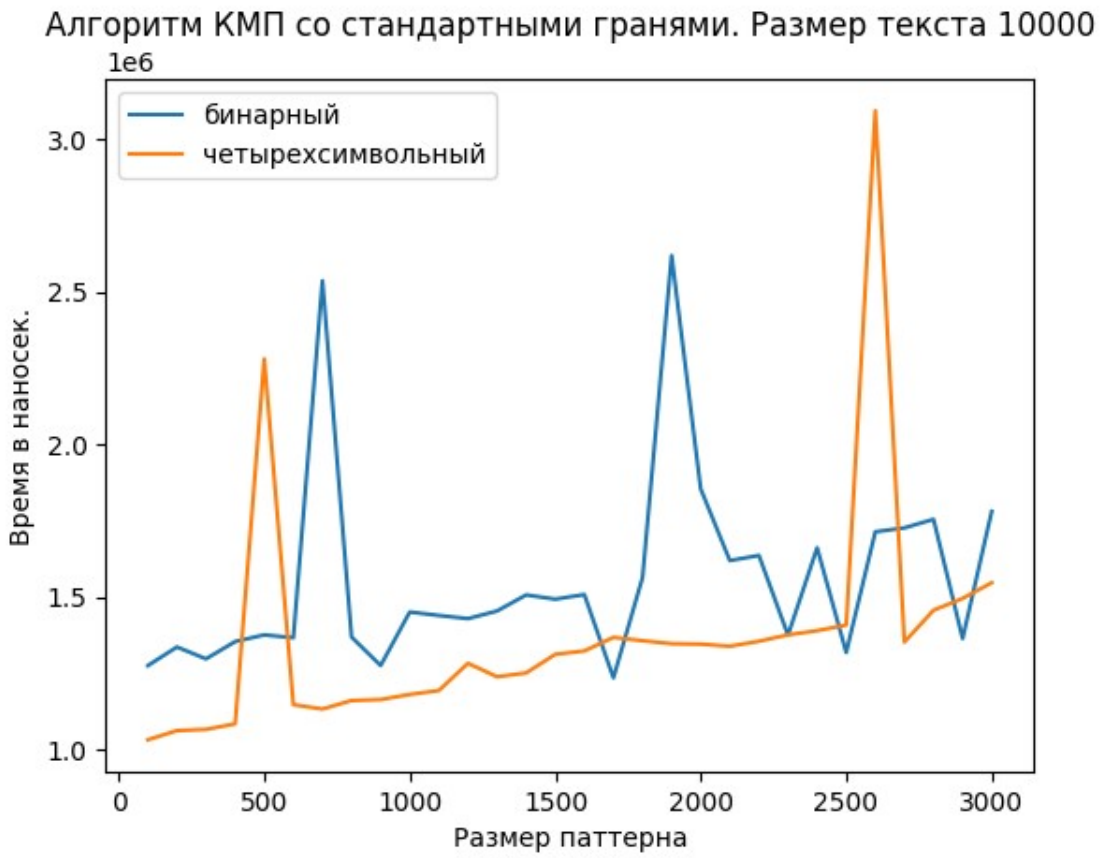
```
naive_algo = data[(data['Algorithm'] == 'наивный') & (data['Text
size'] == 10000)]

alphabet = naive_algo['Type of text'].unique()

for alpha in alphabet:
    rows = naive_algo[naive_algo['Type of text'] == alpha]
    plt.plot(rows['Pattern size'], rows['Time(ns)'], label=alpha)

plt.legend()
plt.title('Наивный алгоритм. Размер текста 10000')
plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.show()
```



Очень сложно сделать вывод, глядя на этот график (:

График для алгоритма КМП со стандартными гранями. Размер текста 10000

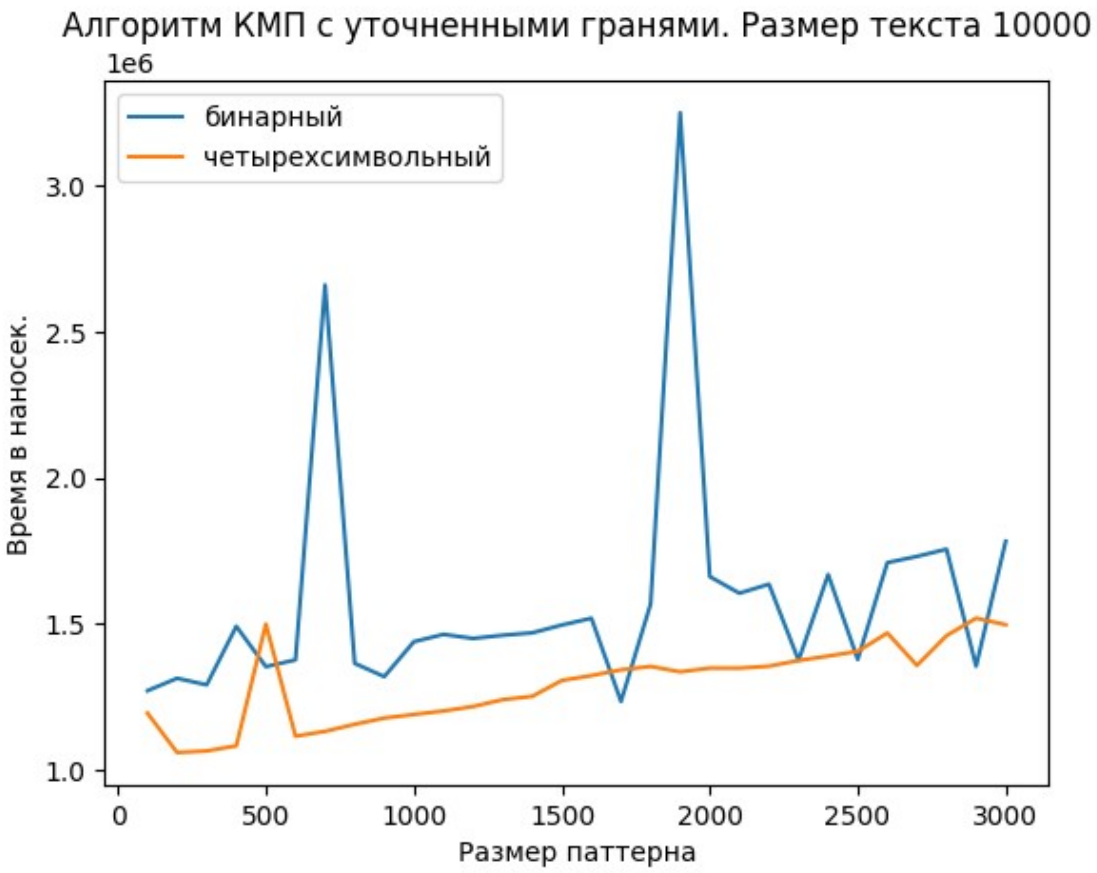
```
kmp = data[(data['Algorithm'] == 'КМП-обычный') & (data['Text size']
== 10000)]

alphabet = kmp['Type of text'].unique()

for alpha in alphabet:
    rows = kmp[kmp['Type of text'] == alpha]
    plt.plot(rows['Pattern size'], rows['Time(ns)'], label=alpha)

plt.legend()
plt.title('Алгоритм КМП со стандартными гранями. Размер текста 10000')
plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.show()
```



Судя по графику, можно сказать, что с четырехсимвольным алфавитом алгоритм работает быстрее, но по сути так быть не должно - связано это исключительно с невозможности провести замеры в изолированной среде

График для алгоритма КМП с уточненными гранями. Размер текста 10000

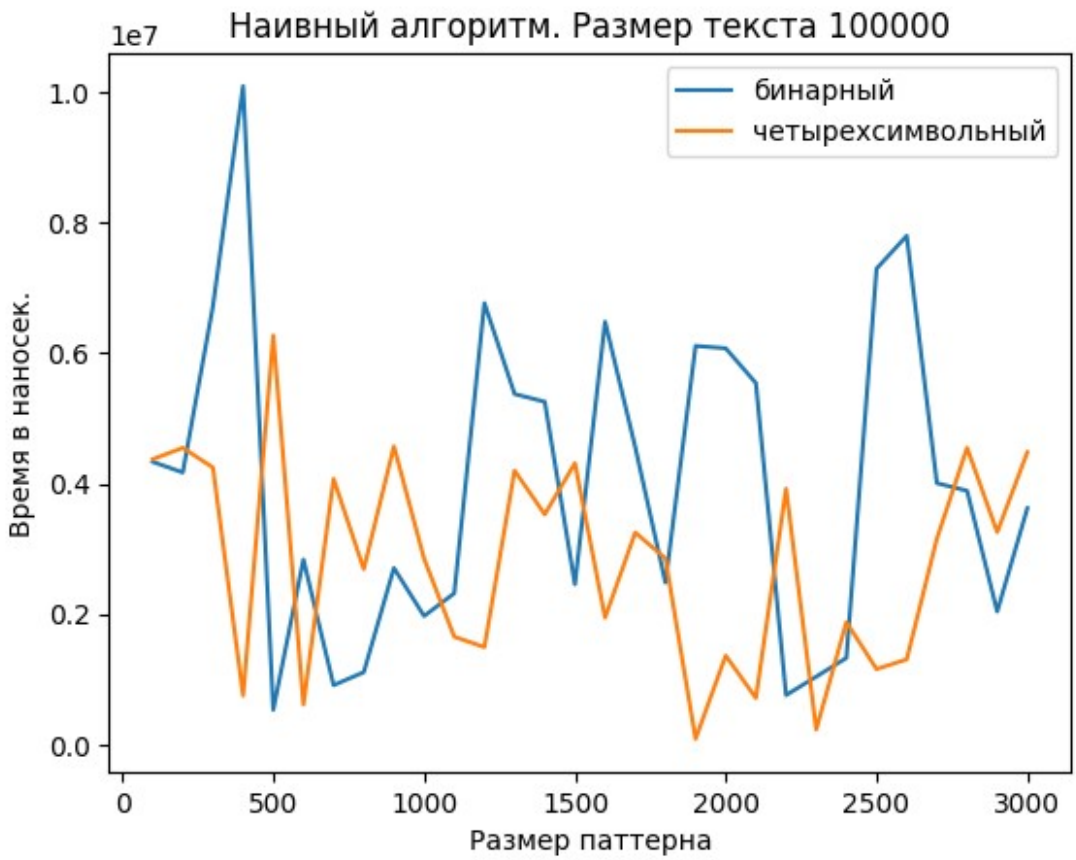
```
kmp = data[(data['Algorithm'] == 'КМП-оптимизированный') & (data['Text size'] == 10000)]

alphabet = kmp['Type of text'].unique()

for alpha in alphabet:
    rows = kmp[kmp['Type of text'] == alpha]
    plt.plot(rows['Pattern size'], rows['Time(ns)'], label=alpha)

plt.legend()
plt.title('Алгоритм КМП с уточненными гранями. Размер текста 10000')
plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.show()
```



без комментариев

График для наивного алгоритма. Размер текста 100000

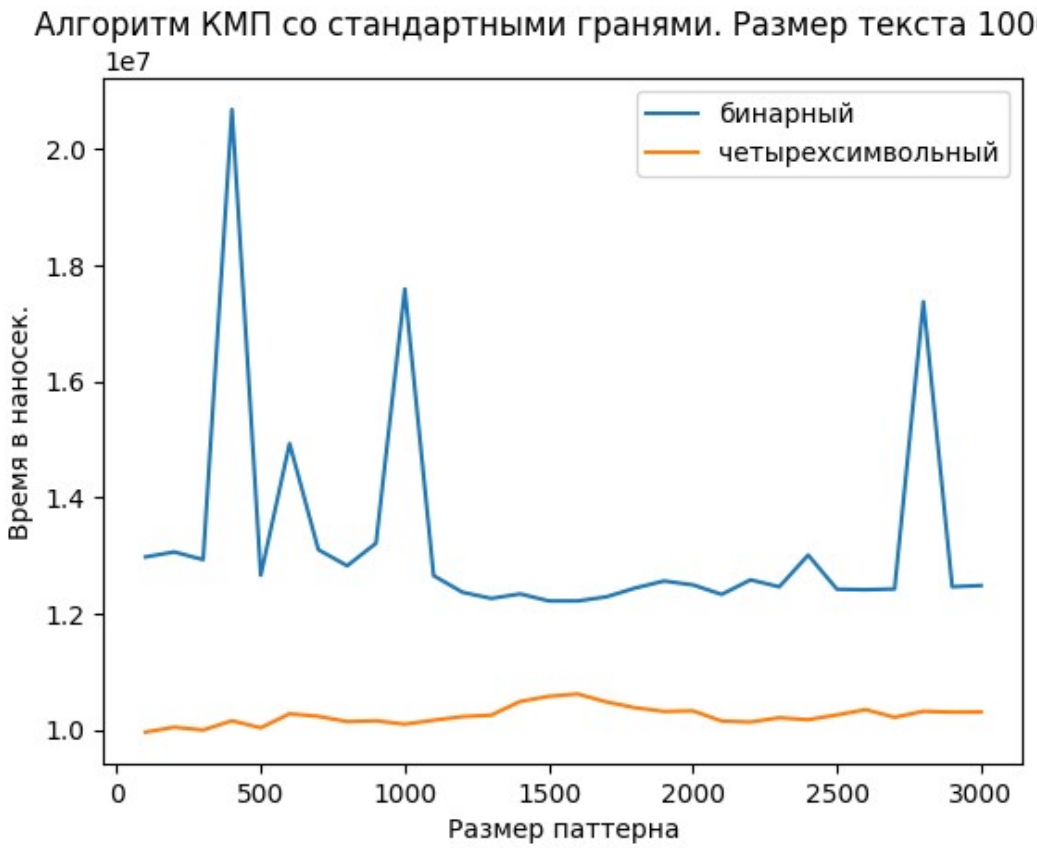
```
naive_algo = data[(data['Algorithm'] == 'наивный') & (data['Text
size'] == 100000)]

alphabet = naive_algo['Type of text'].unique()

for alpha in alphabet:
    rows = naive_algo[naive_algo['Type of text'] == alpha]
    plt.plot(rows['Pattern size'], rows['Time(ns)'], label=alpha)

plt.legend()
plt.title('Наивный алгоритм. Размер текста 100000')
plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.show()
```



Очень очень много выбросов. Работает алгоритм достаточно быстро в силу особенностей реализации

График для алгоритма КМП со стандартными гранями. Размер текста 100000

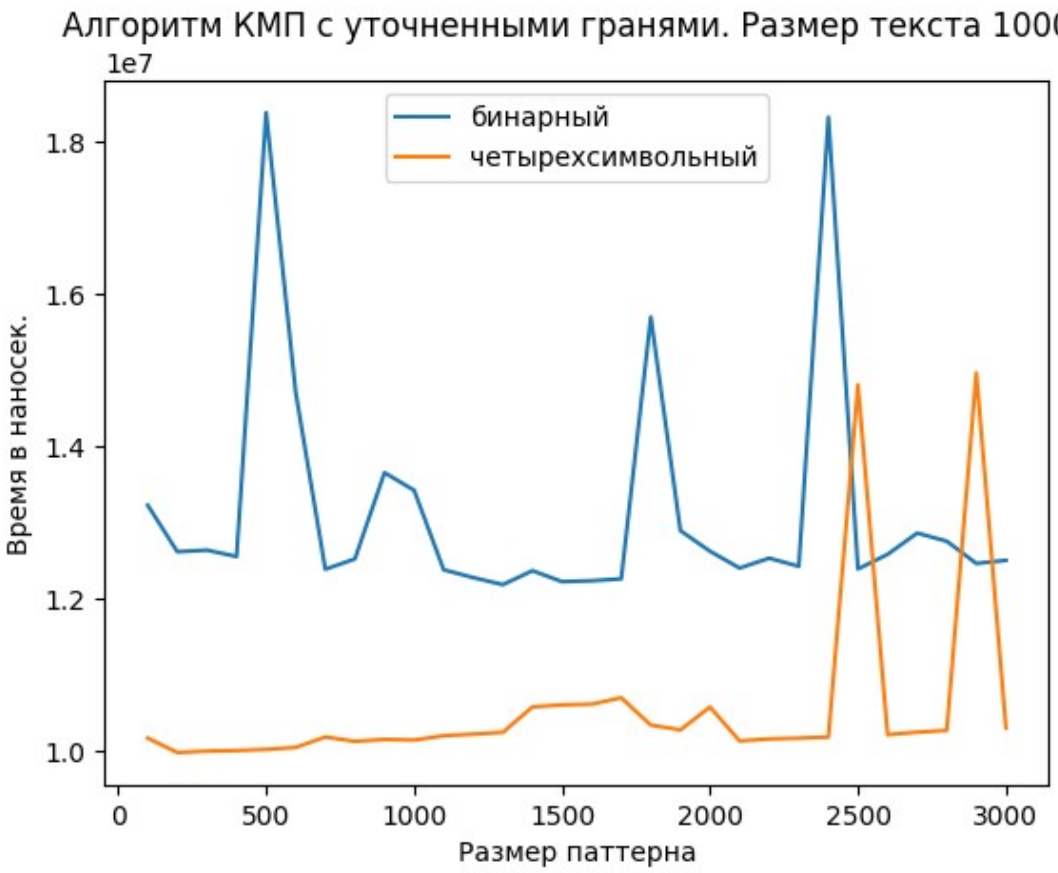
```
kmp = data[(data['Algorithm'] == 'КМП-обычный') & (data['Text size']
== 100000)]

alphabet = kmp['Type of text'].unique()

for alpha in alphabet:
    rows = kmp[kmp['Type of text'] == alpha]
    plt.plot(rows['Pattern size'], rows['Time(ns)'], label=alpha)

plt.legend()
plt.title('Алгоритм КМП со стандартными гранями. Размер текста
100000')
plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.show()
```



Судя по графику с четырехсимвольным алфавитом алгоритм справляется быстрее, но по сути могло быть и наоборот

График для алгоритма КМП с уточненными гранями. Размер текста 100000

```
kmp = data[(data['Algorithm'] == 'КМП-оптимизированный') & (data['Text size'] == 100000)]

alphabet = kmp['Type of text'].unique()

for alpha in alphabet:
    rows = kmp[kmp['Type of text'] == alpha]
    plt.plot(rows['Pattern size'], rows['Time(ns)'], label=alpha)

plt.legend()
plt.title('Алгоритм КМП с уточненными гранями. Размер текста 10000')
plt.xlabel('Размер паттерна')
plt.ylabel('Время в наносек.')
plt.show()
```



Оптимизированная версия алгоритма работает лучше базовой - этого следовало ожидать