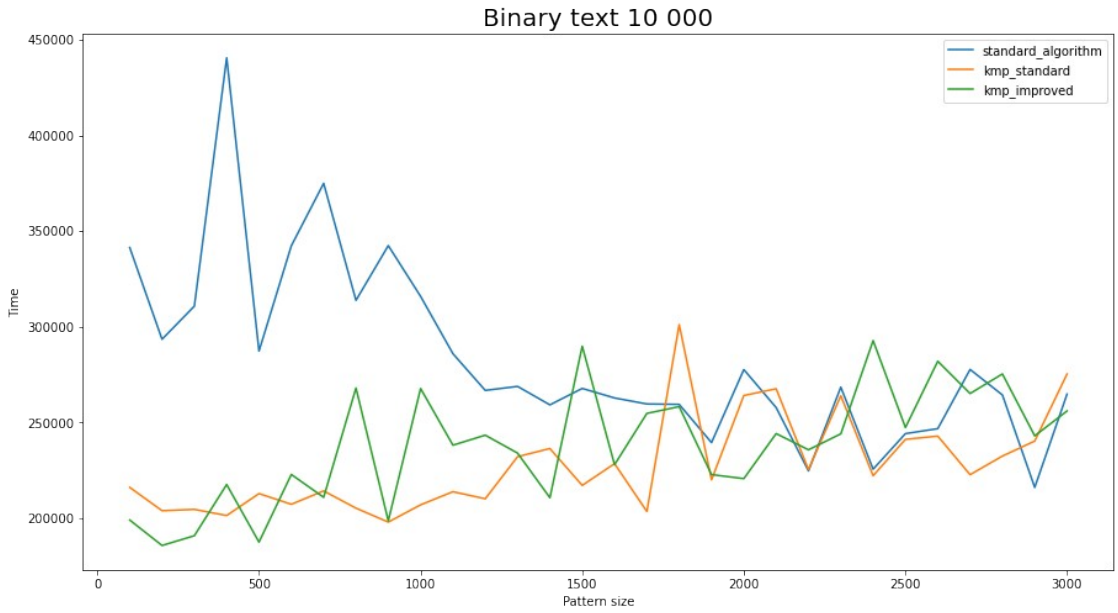
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Binary text 10 000:

```
pattern_size = []
for i in range(100, 3001, 100):
    pattern_size.append(i)
standard_algorithm = []
kmp_standard = []
kmp_improved = []
current_algorithm = ""
with open("Binary 10 000.txt") as file:
    for nums in file:
        if len(nums) > 4 and nums[4] == "d":
            current_algorithm = "StandardAlgorithm"
            continue
        elif len(nums) > 4 and nums[4] == "S":
            current_algorithm = "KMP_Standard"
            continue
        elif len(nums) > 4 and nums[4] == "I":
            current_algorithm = "KMP_Improved"
            continue
        if current_algorithm == "StandardAlgorithm":
            k = nums.split(" ")
            standard_algorithm.append(int(k[1]))
        elif current_algorithm == "KMP_Standard":
            k = nums.split(" ")
            kmp_standard.append(int(k[1]))
        elif current_algorithm == "KMP_Improved":
            k = nums.split(" ")
            kmp_improved.append(int(k[1]))
fig, ax = plt.subplots()
fig.set_size_inches(15,8)
ax.set_xlabel("Pattern size")
ax.set_ylabel("Time")
ax.plot(pattern_size, standard_algorithm, label =
'standard_algorithm')
ax.plot(pattern_size, kmp_standard, label = 'kmp_standard')
ax.plot(pattern_size, kmp_improved, label = 'kmp_improved')
ax.set_title("Binary text 10 000", fontsize= 20)
plt.legend(loc='best')
plt.show()
```



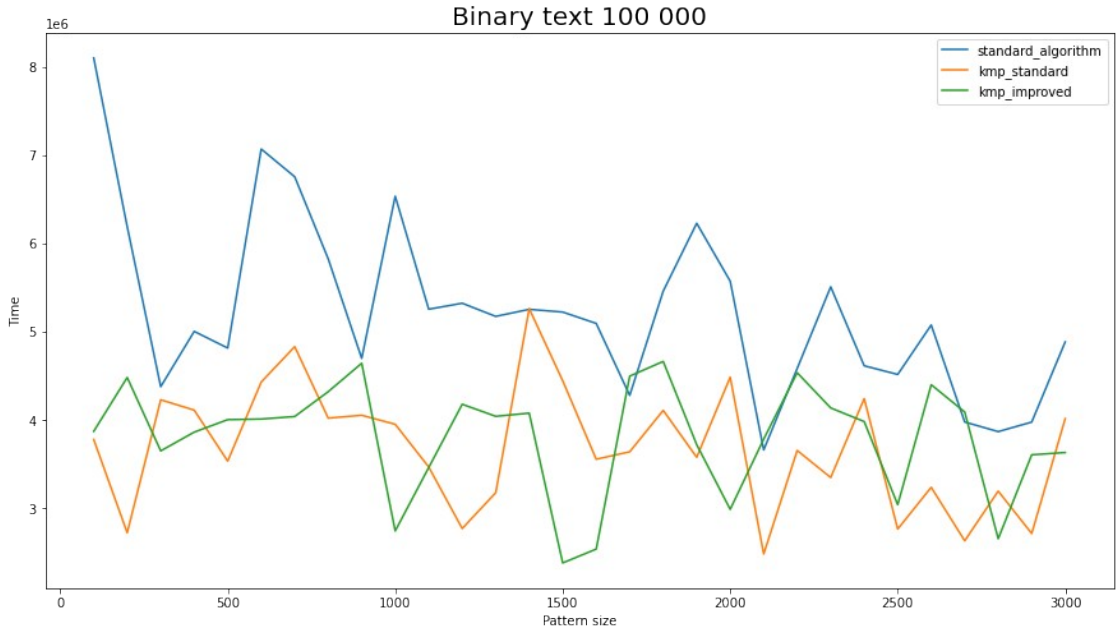
Вывод:

На более коротких шаблонах хорошо видно, что стандартный алгоритм работает крайне хуже двух КМП. А вот ближе к концу они начинают не так сильно друг от друга отставать.

Binary text 100 000:

```
pattern_size = []
for i in range(100, 3001, 100):
    pattern_size.append(i)
standard_algorithm = []
kmp_standard = []
kmp_improved = []
current_algorithm = ""
with open("Binary 100 000.txt") as file:
    for nums in file:
        if len(nums) > 4 and nums[4] == "d":
            current_algorithm = "StandardAlgorithm"
            continue
        elif len(nums) > 4 and nums[4] == "S":
            current_algorithm = "KMP_Standard"
            continue
        elif len(nums) > 4 and nums[4] == "I":
            current_algorithm = "KMP_Improved"
            continue
        if current_algorithm == "StandardAlgorithm":
            k = nums.split(" ")
            standard_algorithm.append(int(k[1]))
        elif current_algorithm == "KMP_Standard":
            k = nums.split(" ")
            kmp_standard.append(int(k[1]))
```

```
        elif current_algorithm == "KMP_Improved":
            k = nums.split(" ")
            kmp_improved.append(int(k[1]))
fig, ax = plt.subplots()
fig.set_size_inches(15,8)
ax.set_xlabel("Pattern size")
ax.set_ylabel("Time")
ax.plot(pattern_size, standard_algorithm, label =
'standard_algorithm')
ax.plot(pattern_size, kmp_standard, label = 'kmp_standard')
ax.plot(pattern_size, kmp_improved, label = 'kmp_improved')
ax.set_title("Binary text 100 000", fontsize= 20)
plt.legend(loc='best')
plt.show()
```



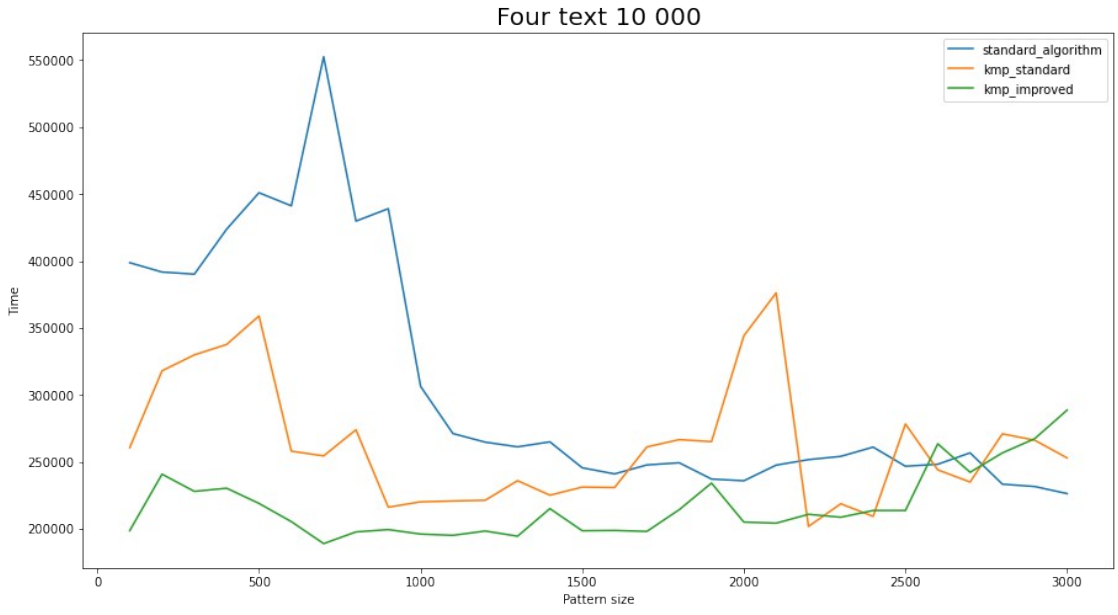
Вывод:

Здесь уже ситуация лучше, чем на прошлом графике. Видно, что почти везде стандартный алгоритм крайне медленно работает. Два КМП же борются за лидерство в течение всех длин шаблонов.

Four text 10 000:

```
pattern_size = []
for i in range(100, 3001, 100):
    pattern_size.append(i)
standard_algorithm = []
kmp_standard = []
kmp_improved = []
current_algorithm = ""
with open("Four 10 000.txt") as file:
    for nums in file:
```

```
        if len(nums) > 4 and nums[4] == "d":
            current_algorithm = "StandardAlgorithm"
            continue
        elif len(nums) > 4 and nums[4] == "S":
            current_algorithm = "KMP_Standard"
            continue
        elif len(nums) > 4 and nums[4] == "I":
            current_algorithm = "KMP_Improved"
            continue
        if current_algorithm == "StandardAlgorithm":
            k = nums.split(" ")
            standard_algorithm.append(int(k[1]))
        elif current_algorithm == "KMP_Standard":
            k = nums.split(" ")
            kmp_standard.append(int(k[1]))
        elif current_algorithm == "KMP_Improved":
            k = nums.split(" ")
            kmp_improved.append(int(k[1]))
fig, ax = plt.subplots()
fig.set_size_inches(15,8)
ax.set_xlabel("Pattern size")
ax.set_ylabel("Time")
ax.plot(pattern_size, standard_algorithm, label =
'standard_algorithm')
ax.plot(pattern_size, kmp_standard, label = 'kmp_standard')
ax.plot(pattern_size, kmp_improved, label = 'kmp_improved')
ax.set_title("Four text 10 000", fontsize= 20)
plt.legend(loc='best')
plt.show()
```

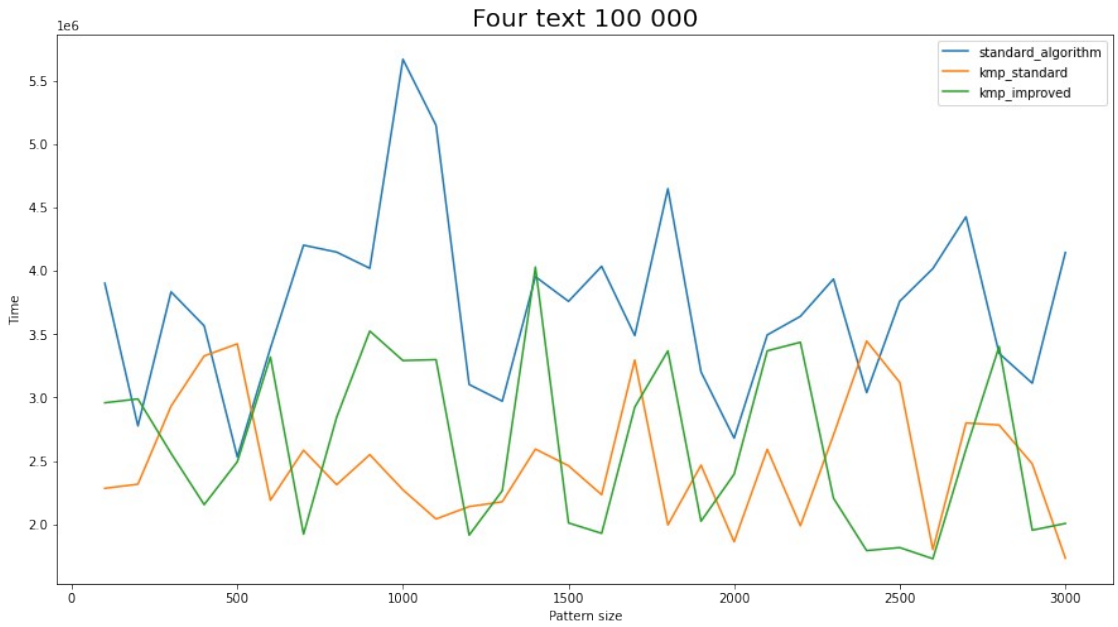


Вывод:

Ситуация похожа на самый первый график, ближе к концу графики показывают примерно одинаковое время обработки строк.

Four text 100 000:

```
pattern_size = []
for i in range(100, 3001, 100):
    pattern_size.append(i)
standard_algorithm = []
kmp_standard = []
kmp_improved = []
current_algorithm = ""
with open("Four 100 000.txt") as file:
    for nums in file:
        if len(nums) > 4 and nums[4] == "d":
            current_algorithm = "StandardAlgorithm"
            continue
        elif len(nums) > 4 and nums[4] == "S":
            current_algorithm = "KMP_Standard"
            continue
        elif len(nums) > 4 and nums[4] == "I":
            current_algorithm = "KMP_Improved"
            continue
        if current_algorithm == "StandardAlgorithm":
            k = nums.split(" ")
            standard_algorithm.append(int(k[1]))
        elif current_algorithm == "KMP_Standard":
            k = nums.split(" ")
            kmp_standard.append(int(k[1]))
        elif current_algorithm == "KMP_Improved":
            k = nums.split(" ")
            kmp_improved.append(int(k[1]))
fig, ax = plt.subplots()
fig.set_size_inches(15,8)
ax.set_xlabel("Pattern size")
ax.set_ylabel("Time")
ax.plot(pattern_size, standard_algorithm, label =
'standard_algorithm')
ax.plot(pattern_size, kmp_standard, label = 'kmp_standard')
ax.plot(pattern_size, kmp_improved, label = 'kmp_improved')
ax.set_title("Four text 100 000", fontsize= 20)
plt.legend(loc='best')
plt.show()
```



Вывод:

Стандартный алгоритм работает явно хуже остальных. А вот 2 кмп сильно прыгают.

Общий вывод:

Хотелось бы пояснить, почему в некоторых случаях КМП может работать медленнее стандартного алгоритма. Дело в том, что порой префикс вектор может быть не супер удачным и символы могут часто не совпадать, в таком случае мы будем двигаться, как и в станд. алгоритме + ещё время на подсчёт префикс вектора.