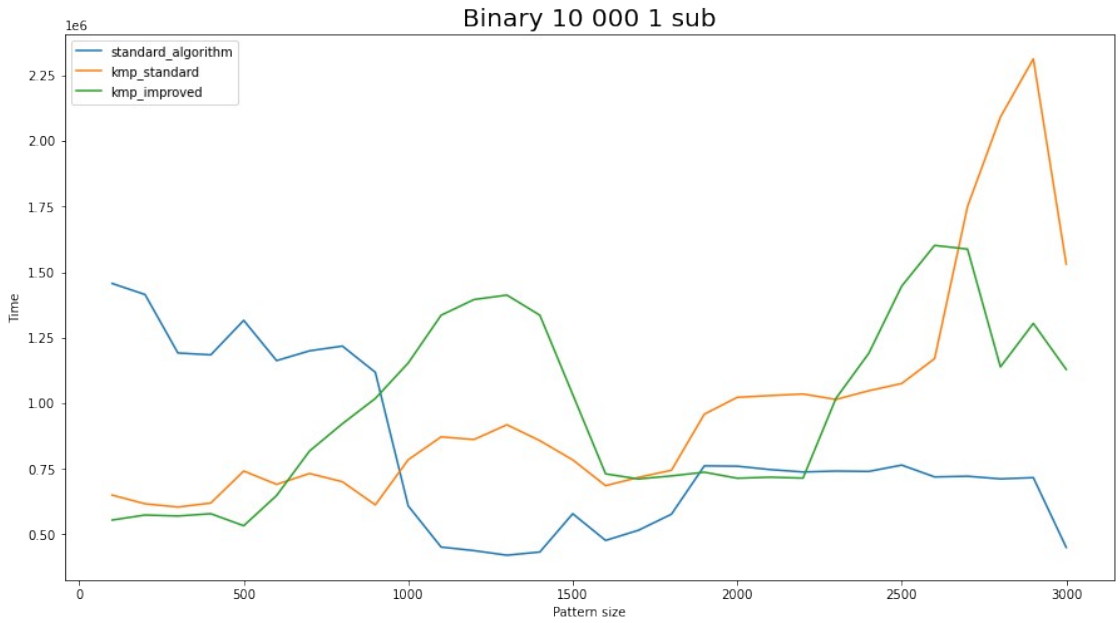
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Binary text 10 000 (1 sub):

```
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  1.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 10 000 1 sub", fontsize= 20)
plt.legend(loc='best')
plt.show()
```



Вывод:

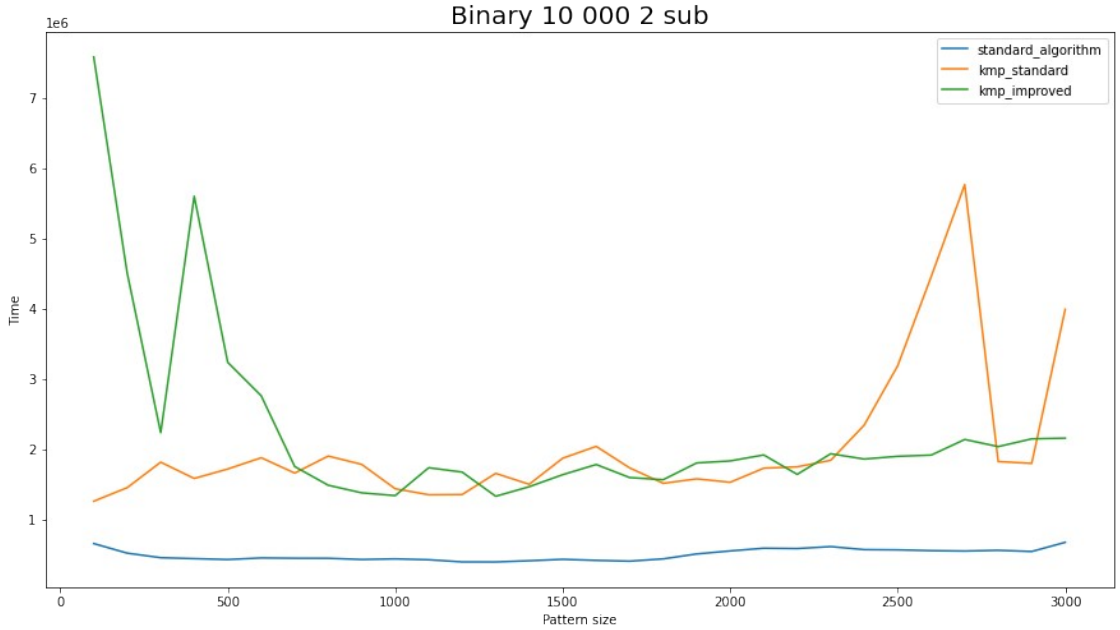
Для начала напишу, как я делала, используя символы подстановки. При стандратном алгоритме я просто проверяю, если совпадают символы или же мы наткнулись на вопросительный знак (по сути одинаковые ситуации). В случае с 2 КМП я составляла всевозможные комбинации символов, которые можн получить при данном числе подстановок.

Тут картина интересная, на самых коротких стандартный алгоритм самый долгий, а вот ближе к концу уже нет. Причём можно обратить внимание, что стандартный алгоритм где-то с длины 1600 работает примерно за одинаковое время.

Binary text 10 000 (2 sub):

```
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  2.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 10 000 2 sub", fontsize= 20)
plt.legend(loc='best')
plt.show()
```

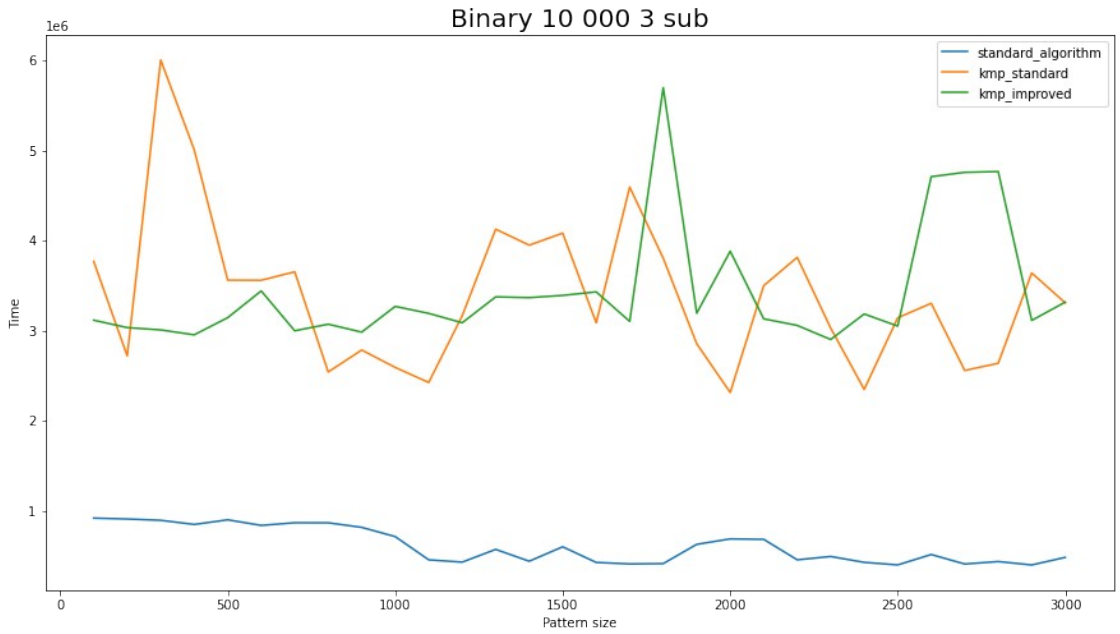


Вывод:

Тут интереснее, стандартный в целом стабилен и работает быстрее всего. Быстрота вполне объяснима, так как на КМП уже идёт больше вариантов на расстановку символов, все варианты нужно перебрать. Причём на промежутке от 700 до 2400 примерно алгоритмы КМП крайне стабильны. Судя по всему префикс вектор примерно одинаковый получается.

Binary text 10 000 (3 sub):

```
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  3.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 10 000 3 sub", fontsize= 20)
plt.legend(loc='best')
plt.show()
```



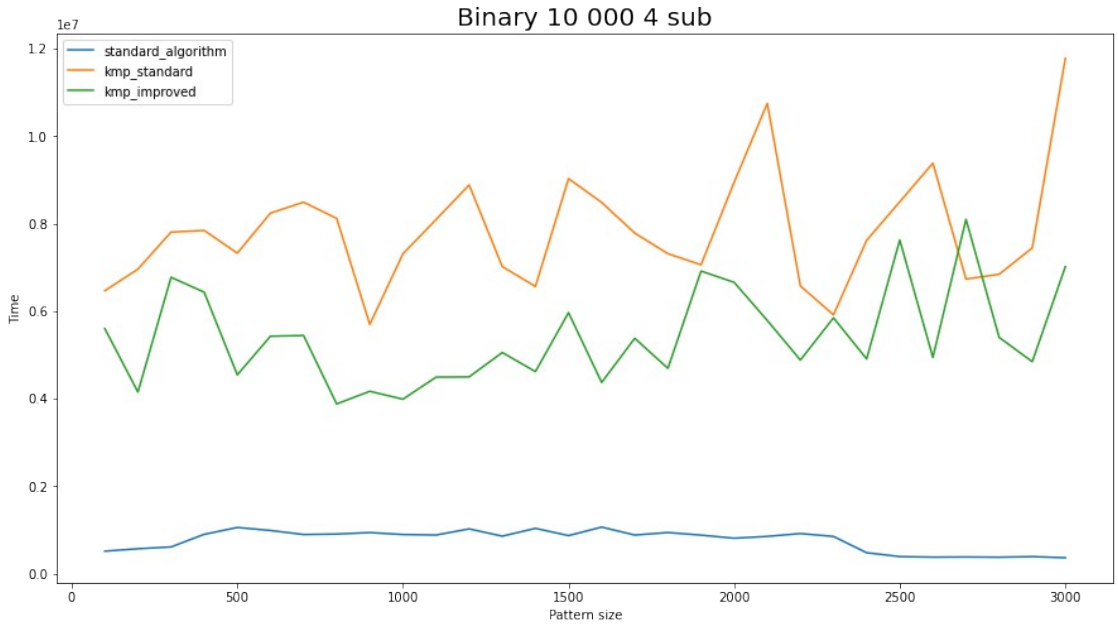
Вывод:

Тут стабильность только на стандартном, ну и опять по очевидным причинам он самый быстрый. Время работы двух кмп гуляет, где-то видны сильные скачки по времени. Что инетересно наблюдать, с 600 по примерно 1000 2 кмп достаточно стабильны.

Binary text 10 000 (4 sub):

```
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  4.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 10 000 4 sub", fontsize= 20)
plt.legend(loc='best')
plt.show()
```



Вывод:

Прям красиво как-то. Всё по ступенькам. kmp_improved имеет больше стабильных участков, чем обычный кмп. Скачки могут происходить из-за того, что + 100 символов к шаблону могут сильно