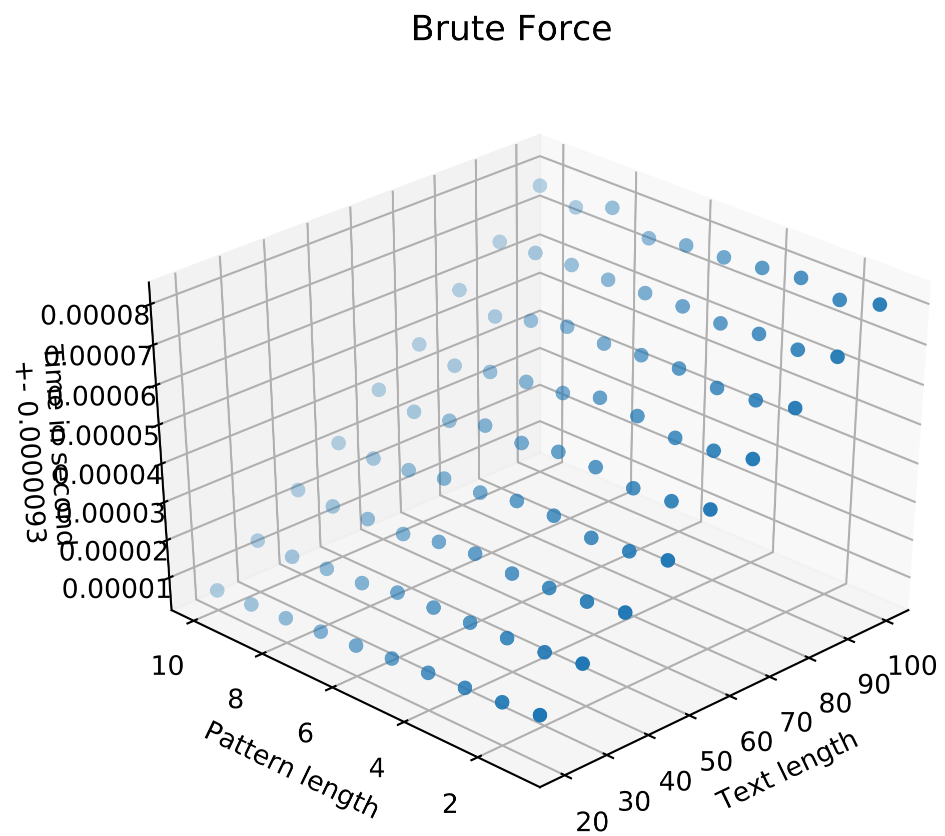
This different substring search algorithms testing shows dependencies of a string preprocessing and other methods of search optimizing

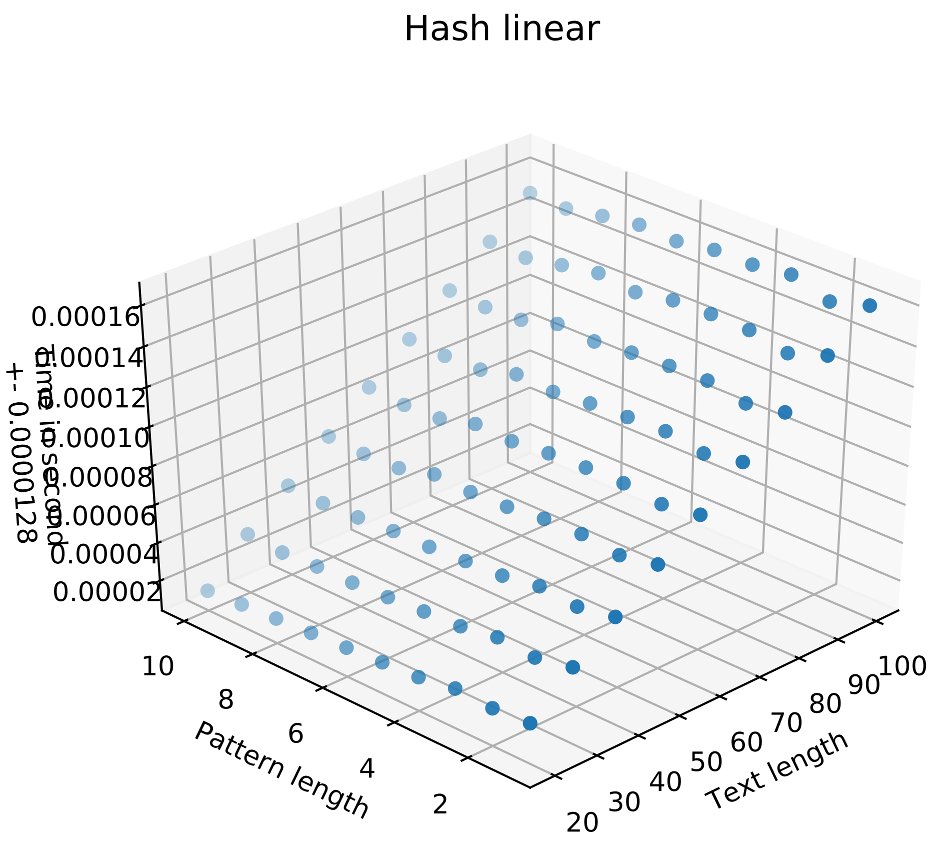
Processor: 3.4GHz Inter Core i5
Memory: 8 GB 1600 MHz DDR3
System: osx

This different substring search algorithms testing shows dependencies of a string preprocessing and other methods of search optimizing

```
Brute Force algorithm associated simply comparing every     substring char
and pattern char until match
Brute Force is pretty stable alghorithm, works every time with O(n^2)
asymptotic

text    'abcdddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
pattern 'abc
Best case:
0.018089590183323404 S
71.44140625 B

text    'abaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaba
pattern 'abc
Worst case:
0.02308733718757594 S
71.44140625 B
```

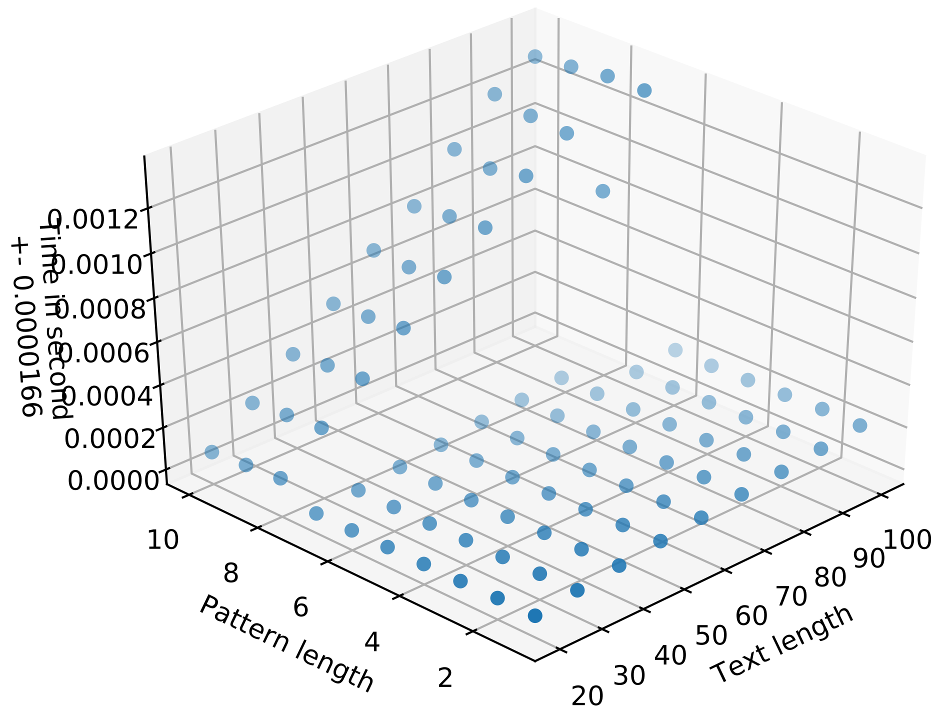


```
Hashing algorithm based on summarizing chars ordinals     to simplify
strings comparison
Hashes have a collision aspect that's why we can see an abrupt behavior change
and time growthon a bad substring

text    'dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
pattern 'abc
Best case:
0.01906160028295933 S
75.1953125 B

text    'ZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhd
pattern 'abc
Worst case:
0.038343771038713524 S
75.1875 B
```

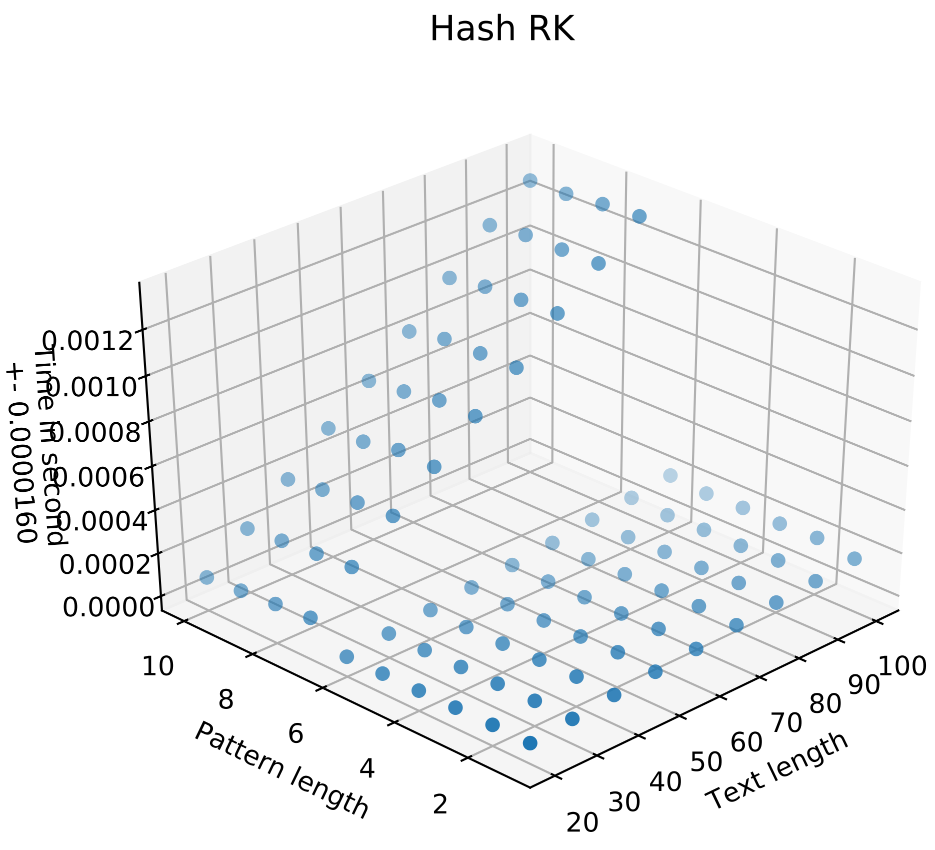


```
Hashing algorithm based on summarizing chars ordinals in square     to
simplify strings comparison
Hashes have a collision aspect that's why we can see an abrupt behavior change
and time growthon a bad substring

text    'dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
pattern 'abc
Best case:
0.04316059674985289 S
77.57421875 B

text    'cbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacba
pattern 'abc
Worst case:
0.057410513349714164 S
77.54296875 B
```

Hash quad



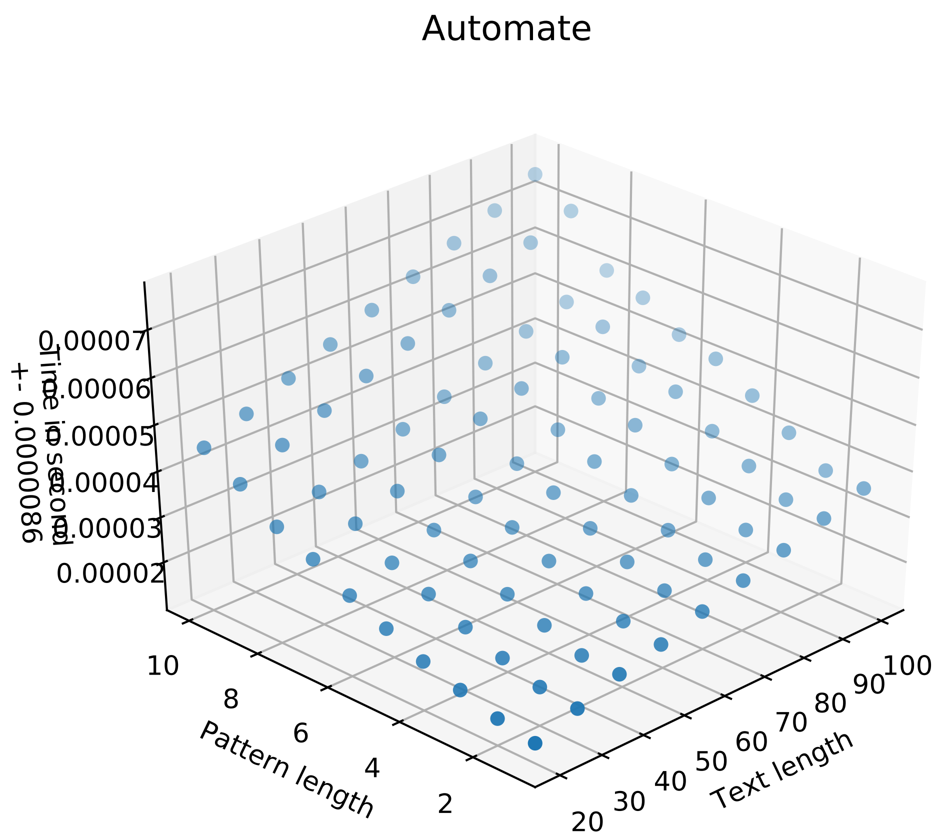
```
Hashing algorithm based on summarizing chars ordinals with special
Rabin Karph formula to simplify strings comparison
Hashes have a collision aspect that's why we can see an abrupt behavior change
and time growthon a bad substring

text    'dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
pattern 'abc
Best case:
0.03195713622724063 S
79.96484375 B

text    'acaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaca
pattern 'abc
Worst case:
0.03736773121408987 S
79.96484375 B
```



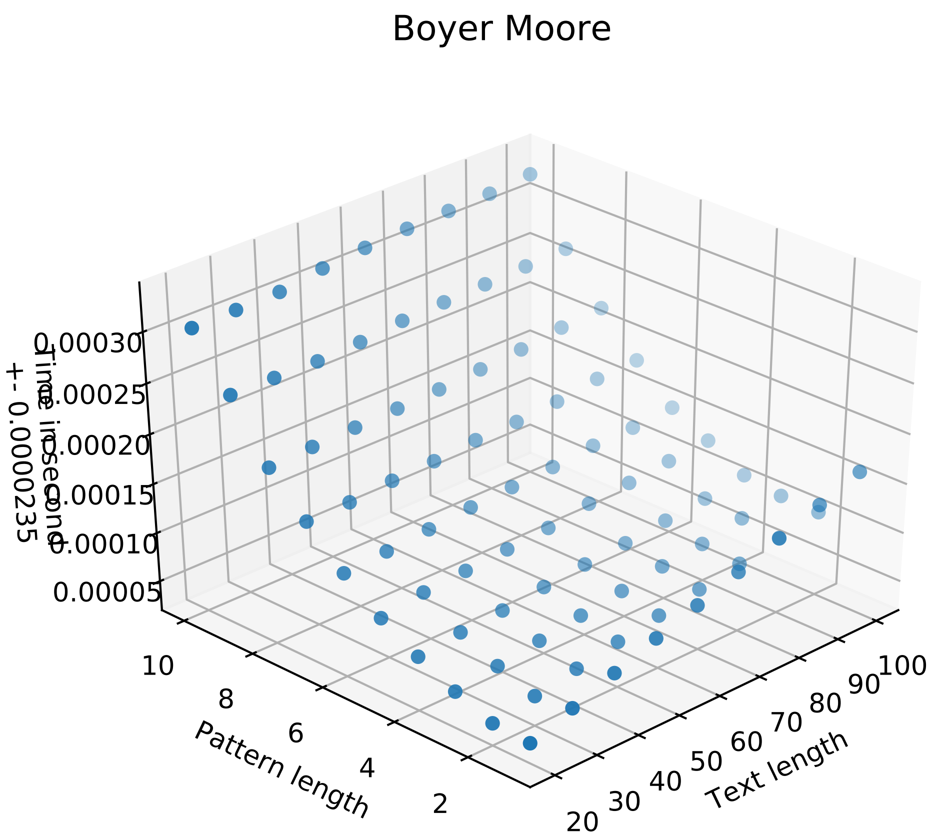
Automate algorithm has a preprocessing component it builds a table of shifts out of pattern which is used during method execution
Automate has a preprocessing aspect that's why we can see time growth

```
text    'cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
pattern 'abc
Best case:
0.007195785635815731 S
82.47265625 B
```

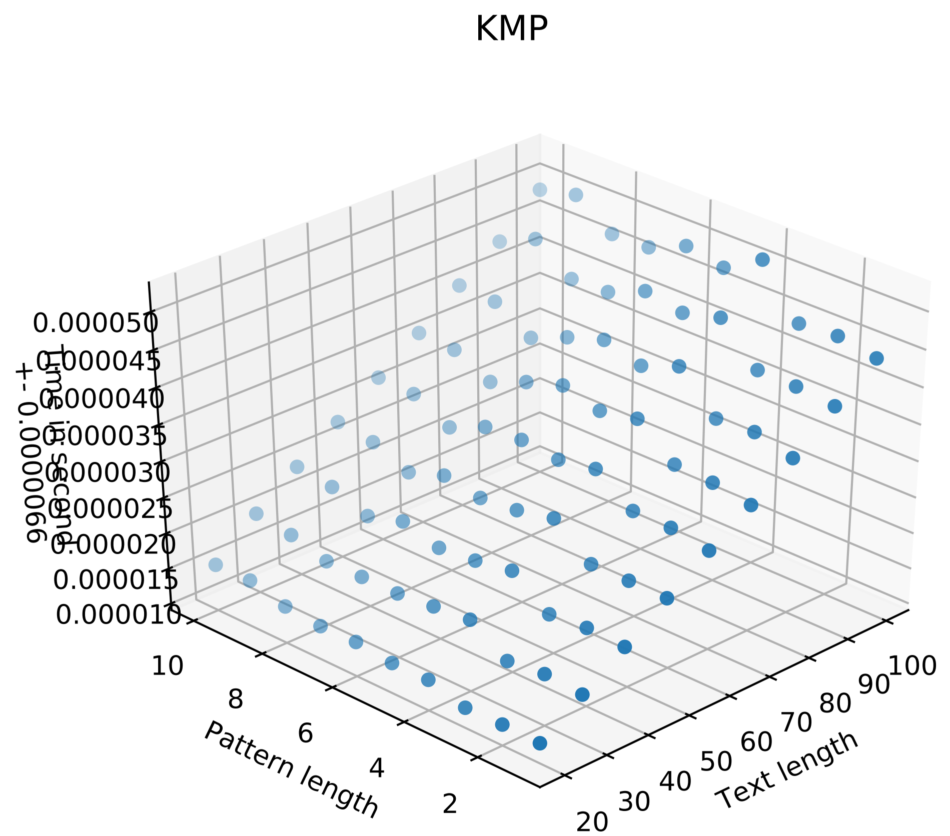
```
text    'abbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabb
pattern 'abc
Worst case:
0.007163999426532374 S
82.47265625 B
```



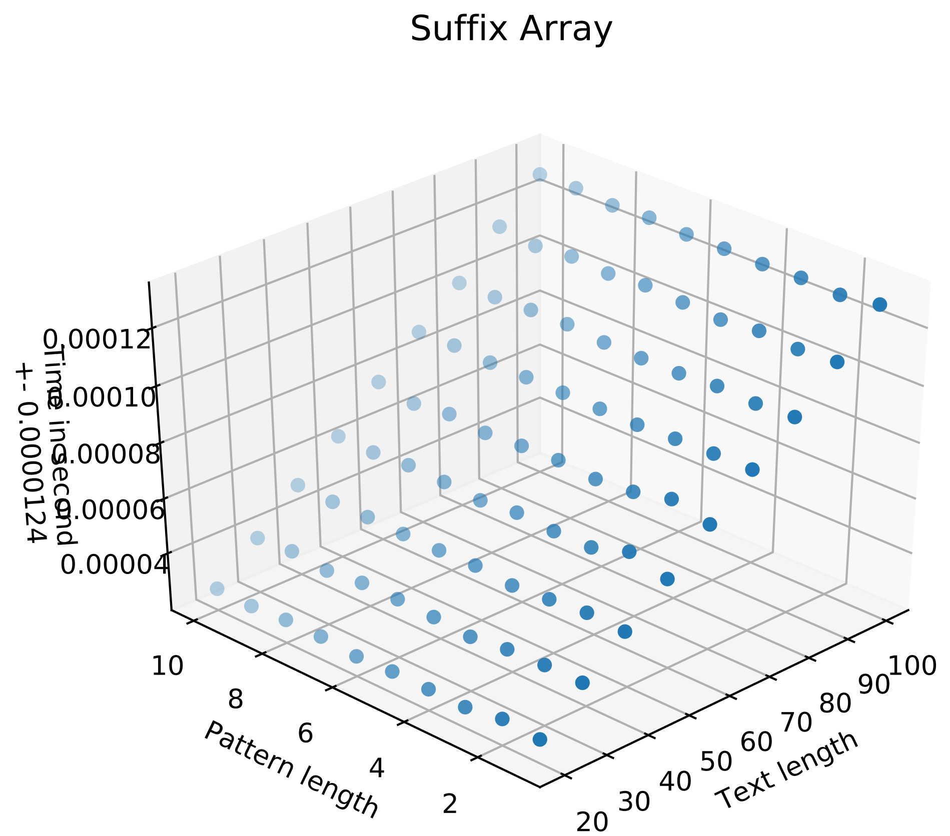
```
Boyer Moore algorithm preprocess a pattern and builds shift tables for
"bad char" anb "good suffix" heuristics
Boyer Moore algorithm works perfectly with prepared pattern on every text but
as we can see there is recounting shift tables time delay

text    'DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
pattern 'ABCDABD
Best case:
0.003592011156017037 S
84.78515625 B

text    'ABCDAB ABCDABCDABABCDAB ABCDABCDABABCDAB ABCDABCDABABCDAB ABCDABCDABABCD
pattern 'ABCDABD
Worst case:
0.04170046699990053 S
84.78515625 B
```



```
KMP or Knuth-Morris-Pratt's algorithm builds shift table with least common
subsequence method then uses it during search
KMP works fine on average string its pretty stable

text    'DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
pattern 'ABCDABD
Best case:
0.002667955940582578 S
89.08203125 B

text    'ABC ABCDAB ABCDABCDABABC ABCDAB ABCDABCDABABC ABCDAB ABCDABCDABABC ABCDA
pattern 'ABCDABD
Worst case:
0.09227560774994951 S
89.08203125 B
```



```
Suffix Array algorithm preprocess a text string with making a suffix
array then uses a binary search to find left and right answer borders
Suffix Array has a text preprocessing that allows to work fast with O(n*log(n))
asymptotic on every pattern

text    'abcaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
pattern 'abc
Best case:
0.006291592505947912 S
129.73828125 B

text    'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
pattern 'b
Worst case:
0.02797513444454159 S
152.34375 B
```


Suffix Array
Time in second
Pattern length
Text length