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

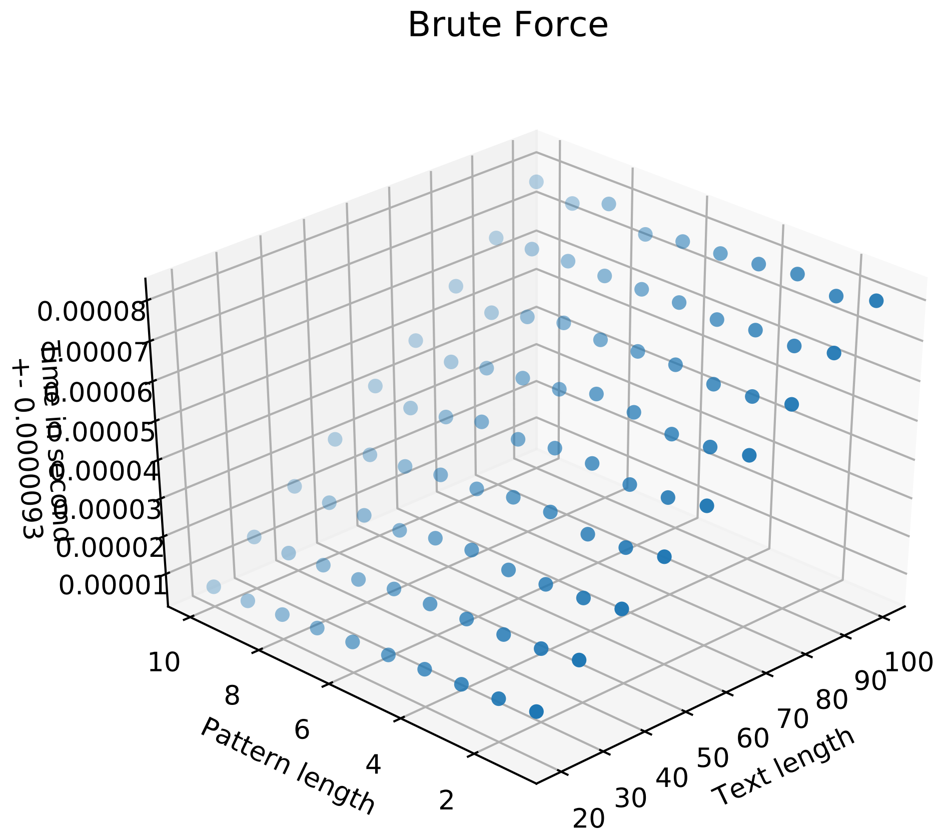
This different substring search algorithms testing shows dependencies of a string preprocessing and other methods of search optimizing

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    'abcddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
pattern 'abc
Best case:
0.0181100256428895З S
70.5625 B
```

```
text    'abaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaba
pattern 'abc
Worst case:
0.021376972400094018 S
70.56640625 B
```

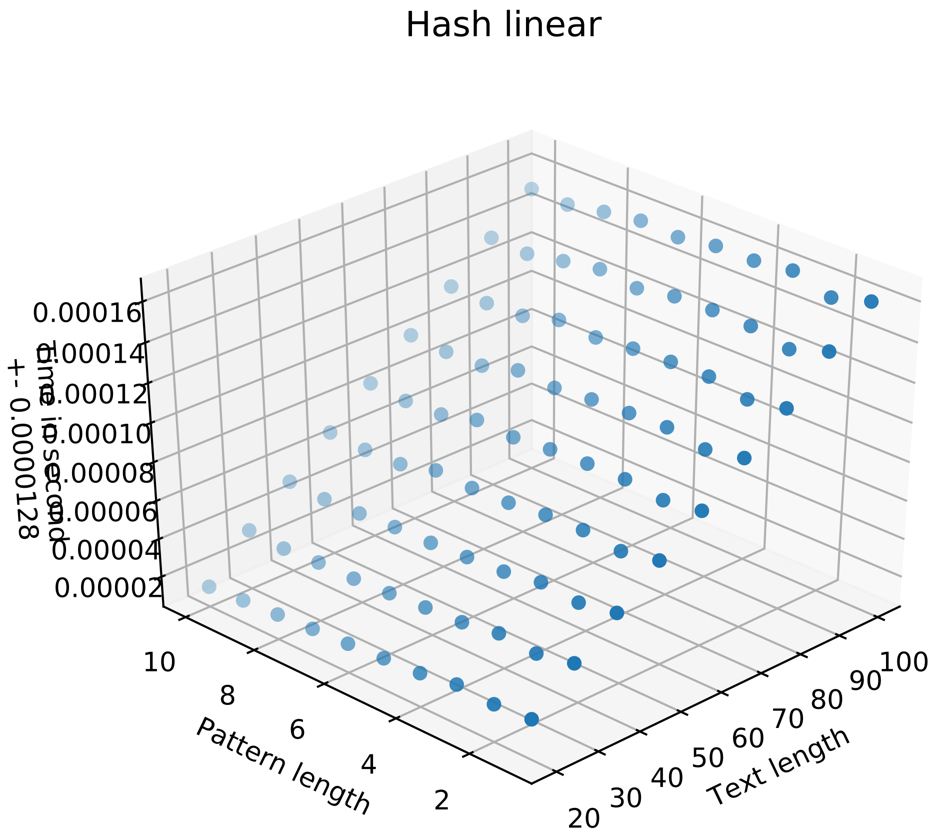


Hashing algorithm based on summarizing chars ordinals   to simplify strings comparison

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:
0.018540316711288245 S
74.78125 B
```

```
text    'ZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhdZhd
pattern 'abc
Worst case:
0.03584653500014004 S
74.7265625 B
```

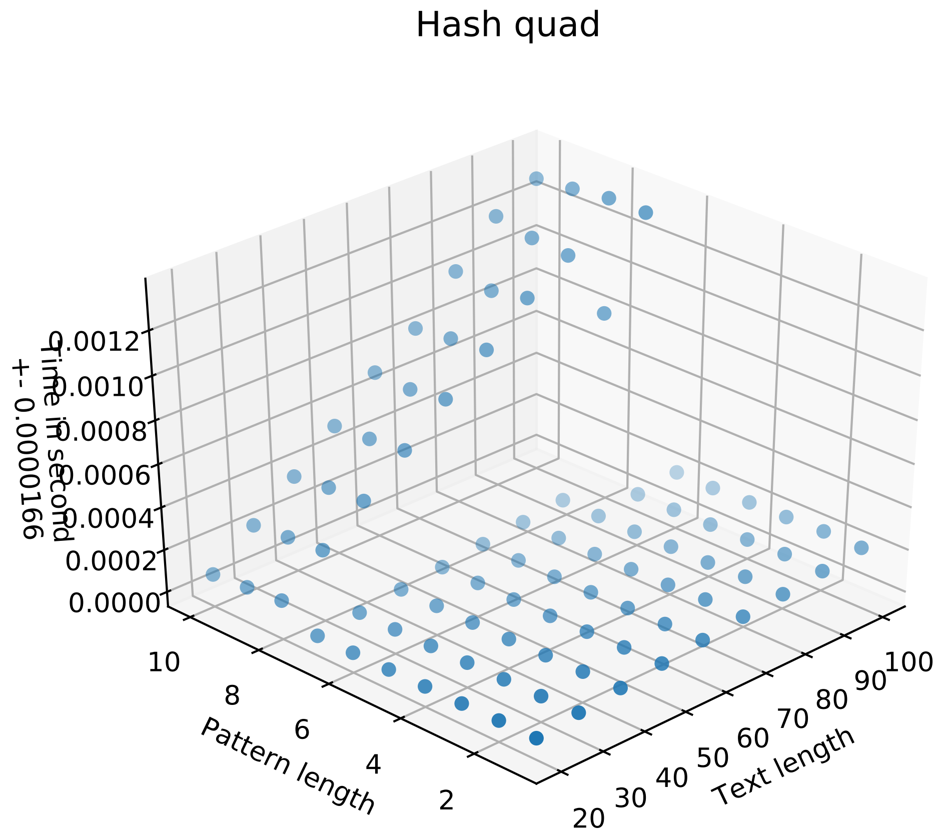


Hashing algorithm based on summarizing chars ordinals in square   to simplify strings comparison

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:
0.03533163470025708 S
77.09765625 B

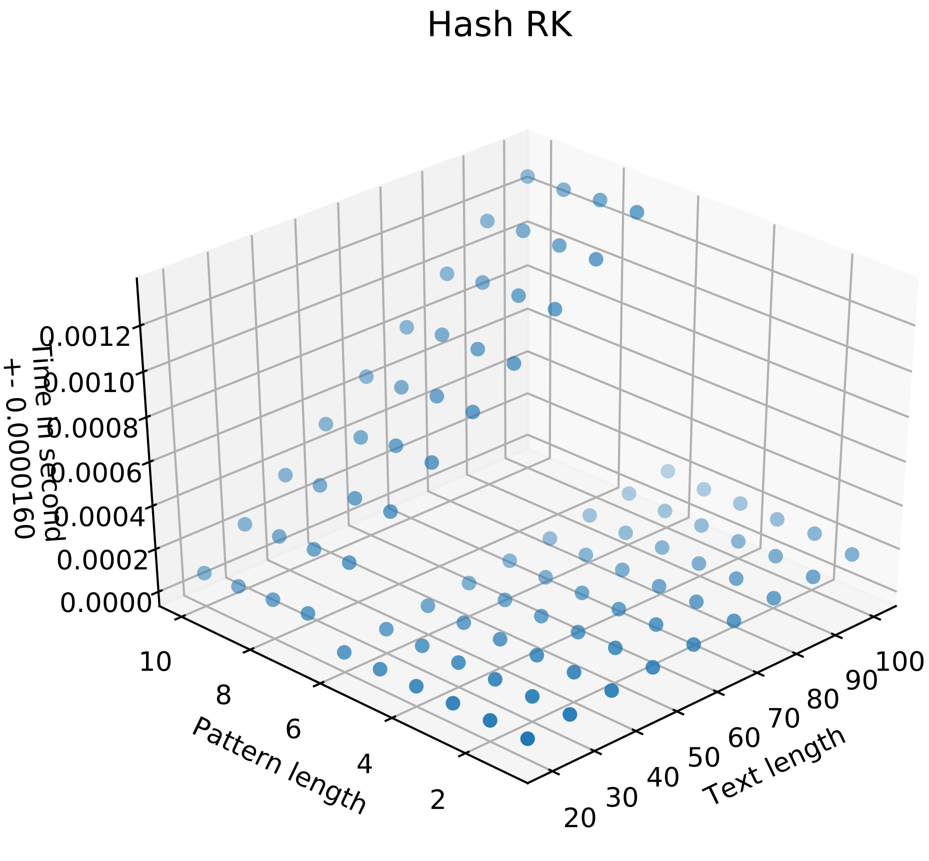
text   'cbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacbacba
pattern 'abc
Worst case:
0.05473548321051088 S
77.09765625 B



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    'ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
pattern 'abc
Best case:
0.028594915685607 7 S
79.46875 B
```

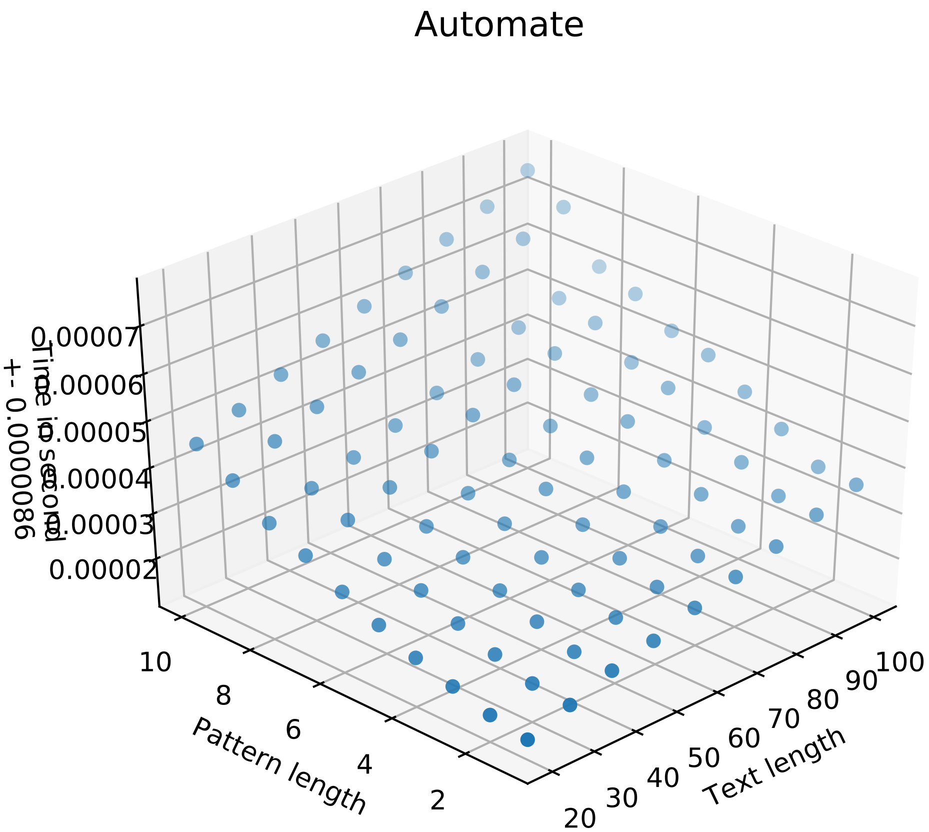
```
text    'acaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaacaaca
pattern 'abc
Worst case:
0.03709414020710843 S
79.46875 B
```



Automate algorithm has a preprocessing component it builds a table of shifts out of pattern which is used during method execution

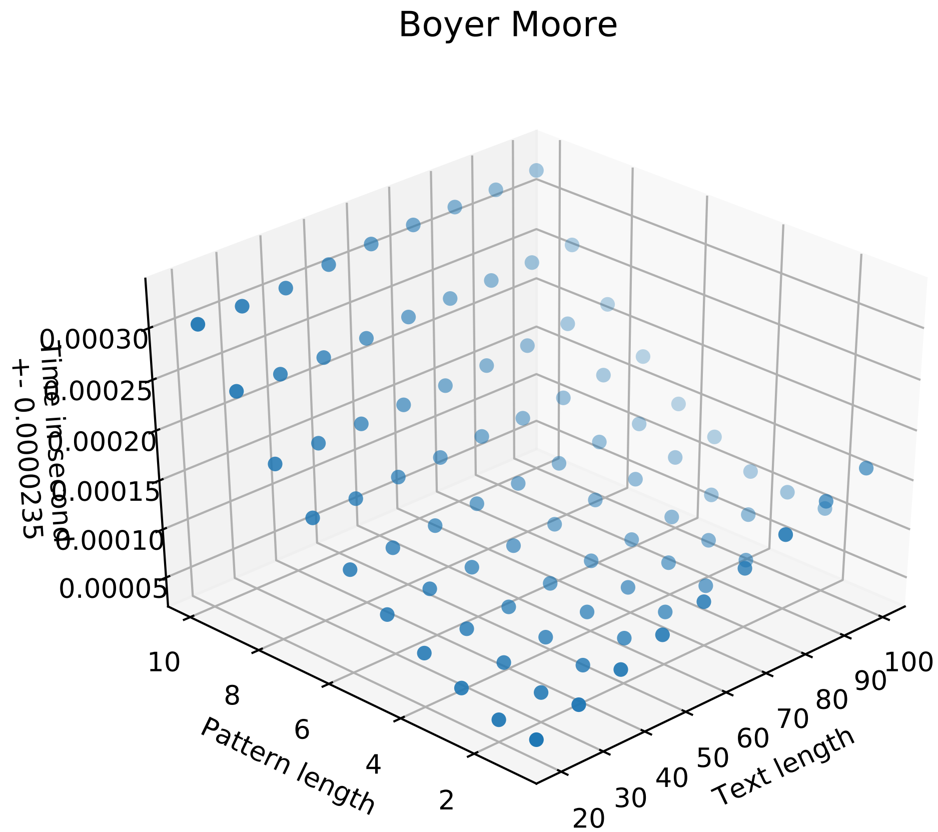
Automate has a preprocessing aspect that's why we can see time growth

```
text    'cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
pattern 'abc
Best case:
0.006662861530232183 S
82.22265625 B

text    'abbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabbabb
pattern 'abc
Worst case:
0.006932610507251608 S
82.22265625 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:
0.003530097134952324 S
84.3984375 B

text    'ABCDAB ABCDABCDABABCDAB ABCDABCDABABCDAB ABCDABCDABABCDAB ABCDABCDABABCD
pattern 'ABCDABD
Worst case:
0.039851759153932034 S
84.3984375 B
```

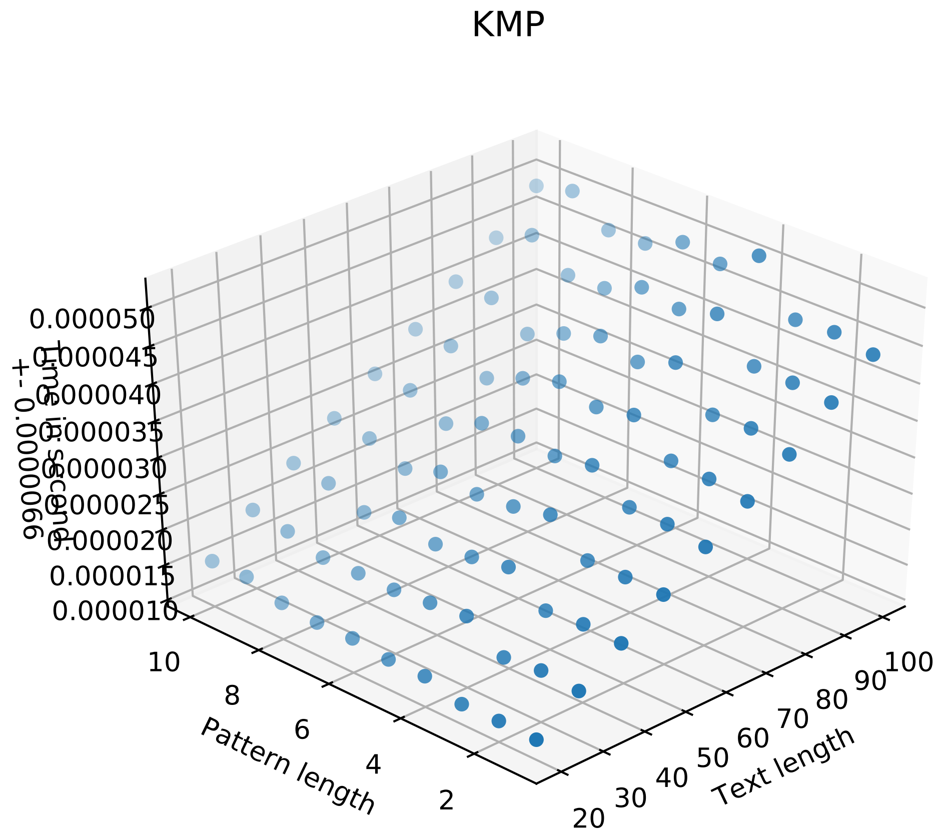


KMP or Knuth-Morris-Pratt's algorithm builds shift table with least common subsequence method then uses it during search

KMP works fine on average string its pretty stable

```
text    'DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
pattern 'ABCDABD
Best case:
0.0025387278885903005 S
89.01953125 B
```

```
text    'ABC ABCDAB ABCDABCDABABC ABCDAB ABCDABCDABABC ABCDAB ABCDABCDABABC ABCDA
pattern 'ABCDABD
Worst case:
0.09061917733333757 S
89.01953125 B
```

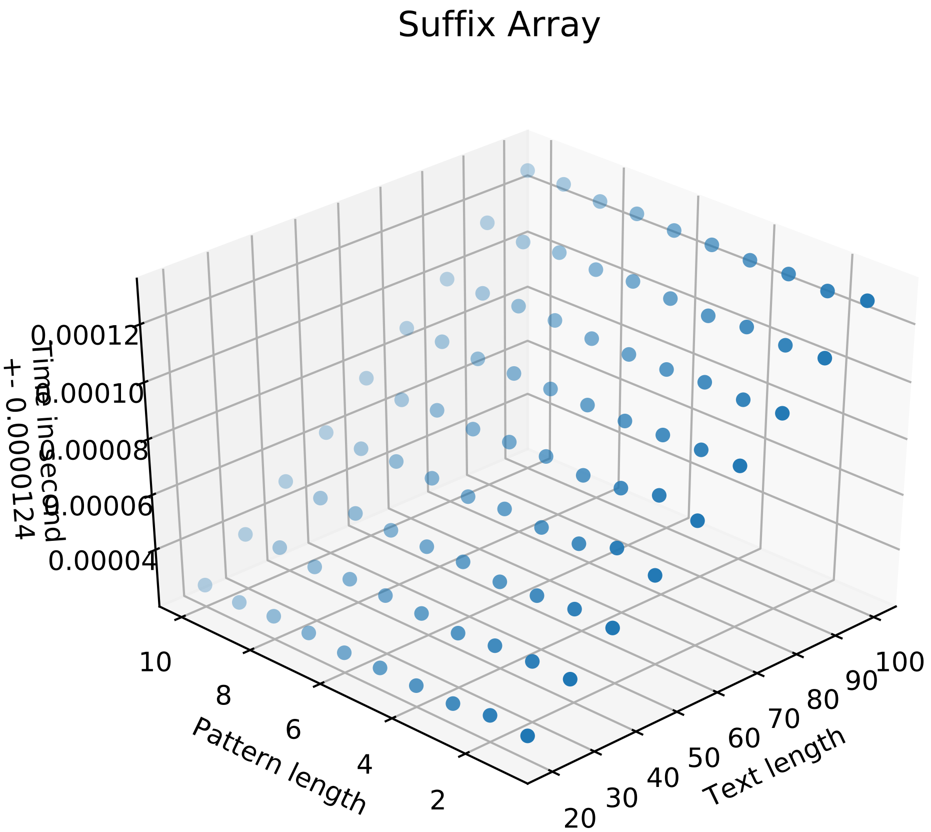


Suffix Array algorithm preprocess a text string with making a suffix array then uses a binary search to find left and right answer borders

Suffix Array has a text preprocessing that allows to work fast with O(n*log(n)) asymptotic on every pattern

```
text    'abcaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
pattern 'abc
Best case:
0.0060937853912933415 S
133.69921875 B
```

```
text    'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
pattern 'b
Worst case:
0.0345027188695499 S
151.24609375 B
```


Suffix Array
Time in second
Pattern length
Text length