

Suffix Array construction

```
·miss·missing  
·missing  
g  
ing  
iss·miss·missing  
iss·missing  
issing  
miss·missing  
missing  
ng  
s·miss·missing  
s·missing  
sing  
ss·miss·missing  
ss·missing  
ssing  
swiss·miss·missing  
wiss·miss·missing
```

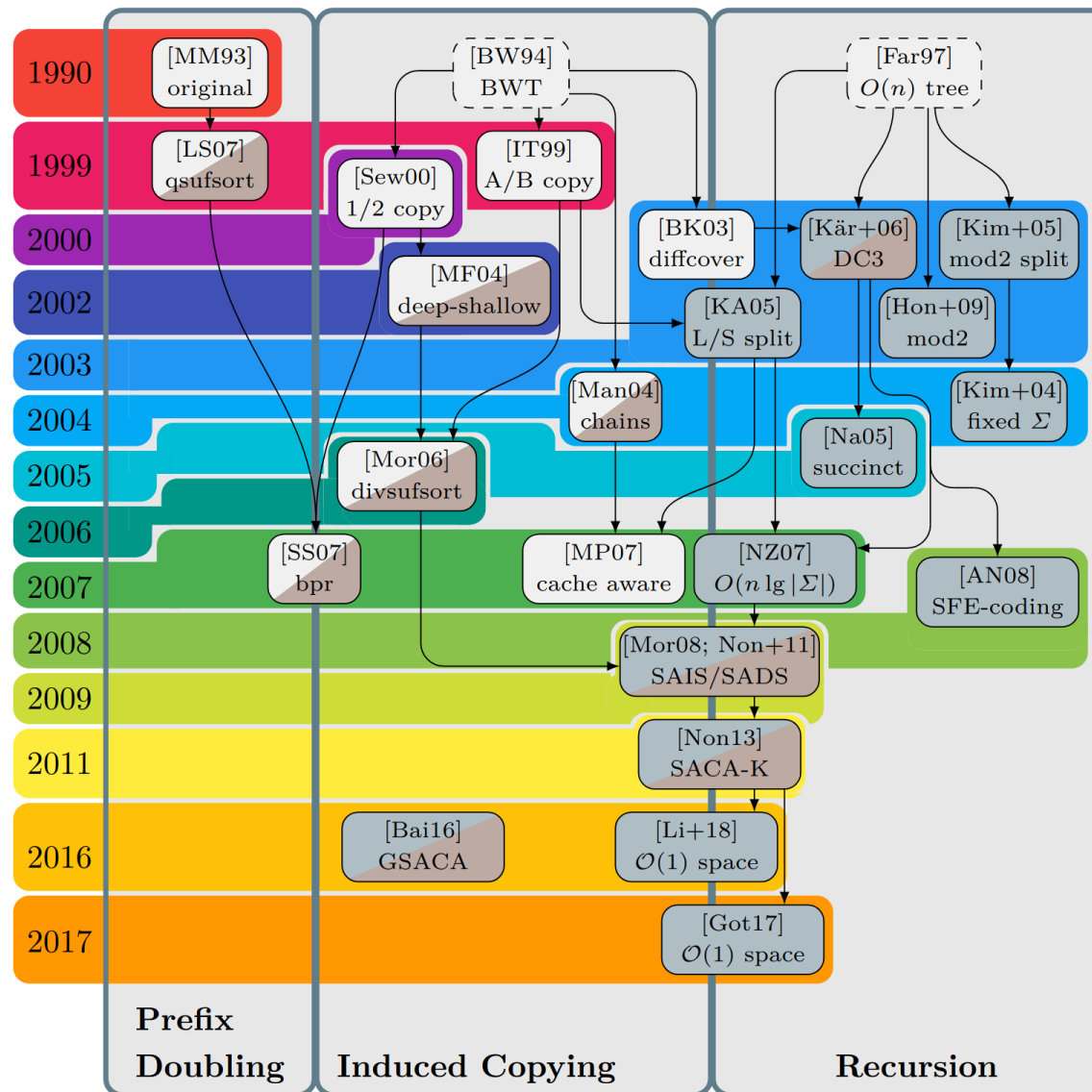
To build the Suffix Array we need to sort all the suffixes of the input text T

If $n=|T|$ a traditional sorting algorithm would require $\Theta(n \log n)$ comparisons between strings of length $O(n)$.

For the text $T = a^{n-1}b$ this would mean $\Theta(n^2 \log n)$ character comparisons .

We can do better than that: there are algorithms that run in optimal $O(n)$ time.

Suffix sorting algorithms have been extensively studied in the last 20 years:



[Kurpicz '19]

The DC3 algorithm [Kark. et al 2003]

Run in linear time

Simple and elegant

Can be made lightweight (use space close to the SA and the original text)

Can be adapted to several models (external memory, cache oblivious, etc.)

The idea

We distinguish between the suffixes starting at positions $\equiv 0 \pmod 3$ (red) and the others (black).

swiss · miss · missing

Algorithm DC3

1. sort the black suffixes (use recursion, see later)
2. sort the red suffixes (use ordering of black suffixes)
3. merge the suffixes (use ordering of black suffixes)

To compare two red suffixes (Step 2):

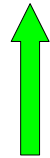
If the first two characters differ it is easy:

sw**i**s**s** · **m**i**s**s · m**i**s**s****i****n**g



If the first two characters are equal we look at the following black suffixes (whose order is known):

sw**i**s**s** · **m**i**s**s · m**i**s**s****i****n**g

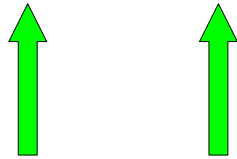


We can use Radix Sort so Step 2 takes $O(n)$ time

To compare a red and a black suffix (Step 3):

After at most 2 steps we land into two black suffixes whose order is known

swi**s**s · m**i**ss · m**i**ss**i**ng



swi**s**s · m**i**ss · m**i**ss**i**ng



Step 3 takes $O(n)$ time too!

How to sort the black suffixes

We consider all the triplets of characters starting at black positions

SA(T') =
9,4,13,8,3,...

swiss·miss·missing

wis s·m iss ·mi ssi ng\$
iss ·mi ss· mis sin g\$\$

We sort the $2n/3$ triplets and compute their rank:

0	1	2	3	4	5	6	7	8	9
·mi	g\$\$	iss	mis	ng\$	s·m	sin	ss·	ssi	wis

We build a new string whose SA gives the lex order of the black suffixes

T' = 952084#207361\$

Summing up

We have the recurrence:

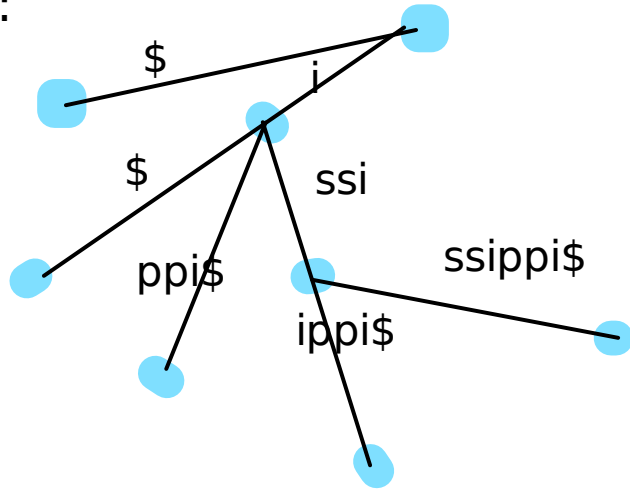
$$T(n) = T(2n/3) + O(n)$$

so DC3 runs in $O(n)$ time.

Building the ST from SA+LCP in linear time

Example for mississippi: see Sect. 10.3.2 of
the textbook for a more formal argument

Example:



SA LCP

12	-	\$
11	0	i\$
8	1	ippi\$
5	1	issippi\$
2	4	issippi\$
1	0	issippi\$
10	0	ppi\$
9	1	ppi\$
7	0	sippi\$
4	2	sissippi\$
6	1	ssippi\$
3	3	ssissippi\$

Short version: we add suffixes (ie leaves) in lex order, from the most recently added leaf we go up in the tree as dictated from the LCP values and the down to add a new leaf.

