

Radix and Suffix Sorting

- Radix Sort
- Suffix Sort

Philip Bille

Radix and Suffix Sorting

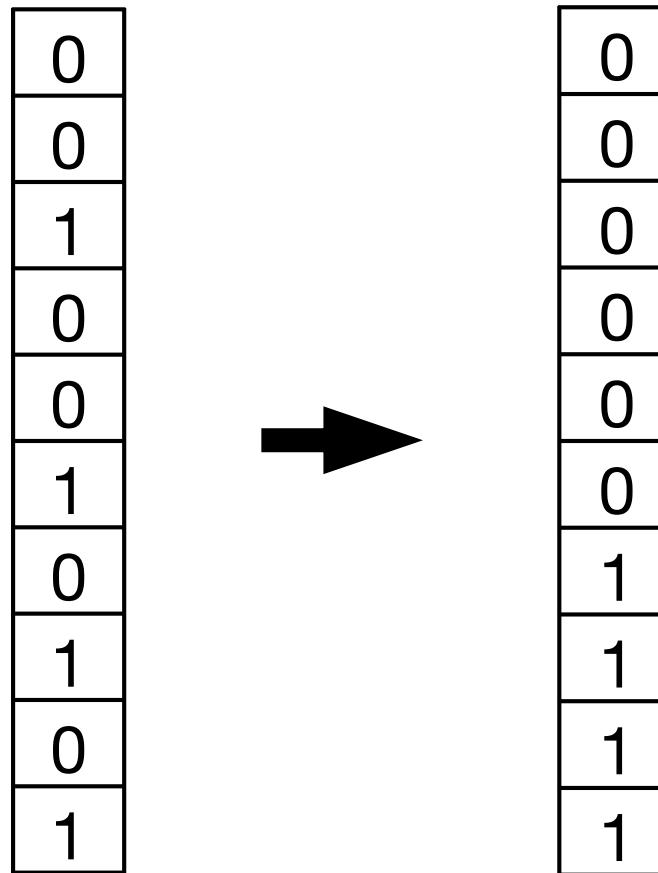
- Radix Sort
- Suffix Sort

Radix Sort

- **Sorting small universes.** Given a sequence of n integers from a universe $U = \{0, 1, \dots, u-1\}$.
- How fast can we sort sequence if the size of the universe is not too big?

Radix Sort

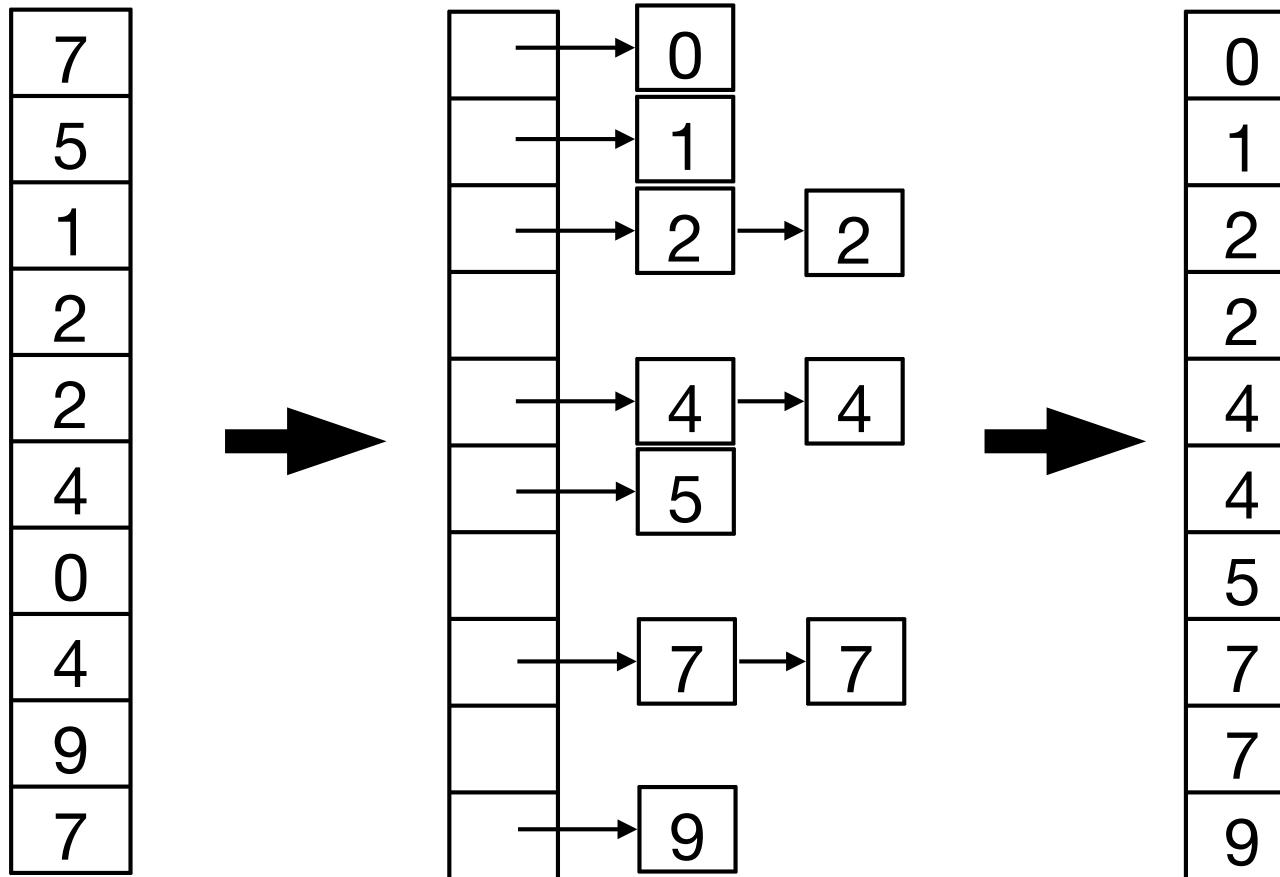
$$n = 10, U = \{0,1\}$$



- **Algorithm.** Count 0s and 1s.
- **Time.** $O(n)$.

Radix Sort

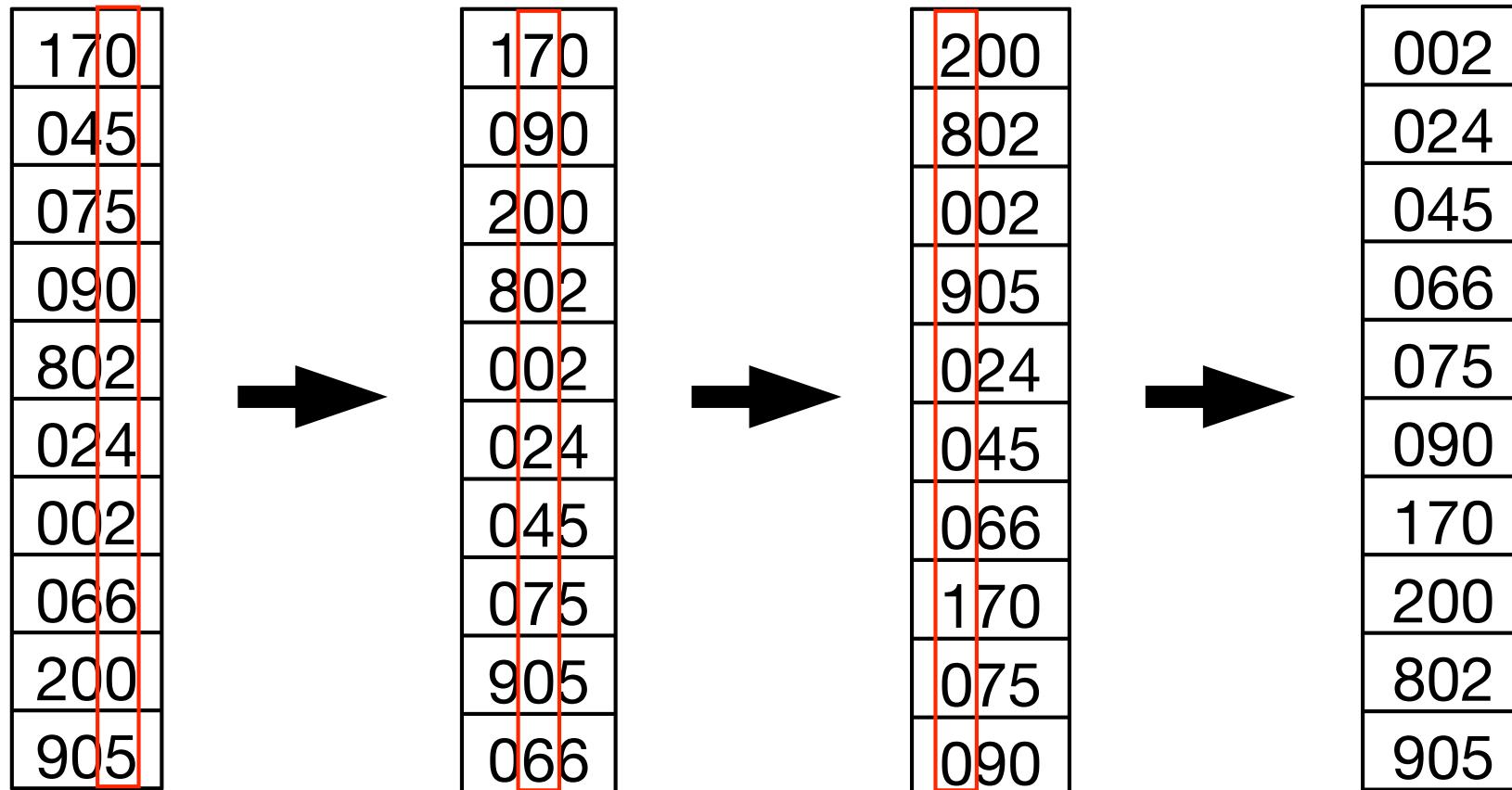
$$n = 10, U = \{0, 1, \dots, n-1 = 9\}$$



- **Algorithm.** Insert into array of linked list + traverse array of linked list.
- **Time.** $O(n + u) = O(n)$
- Sorting can be **stable**.

Radix Sort

$$n = 10, U = \{0, \dots, n^3 - 1 = 999\}$$



- **Radix Sort.** Sort on each digit from right to left using stable sort.
- **Time.** $O(n + n + n) = O(n)$

Radix Sort

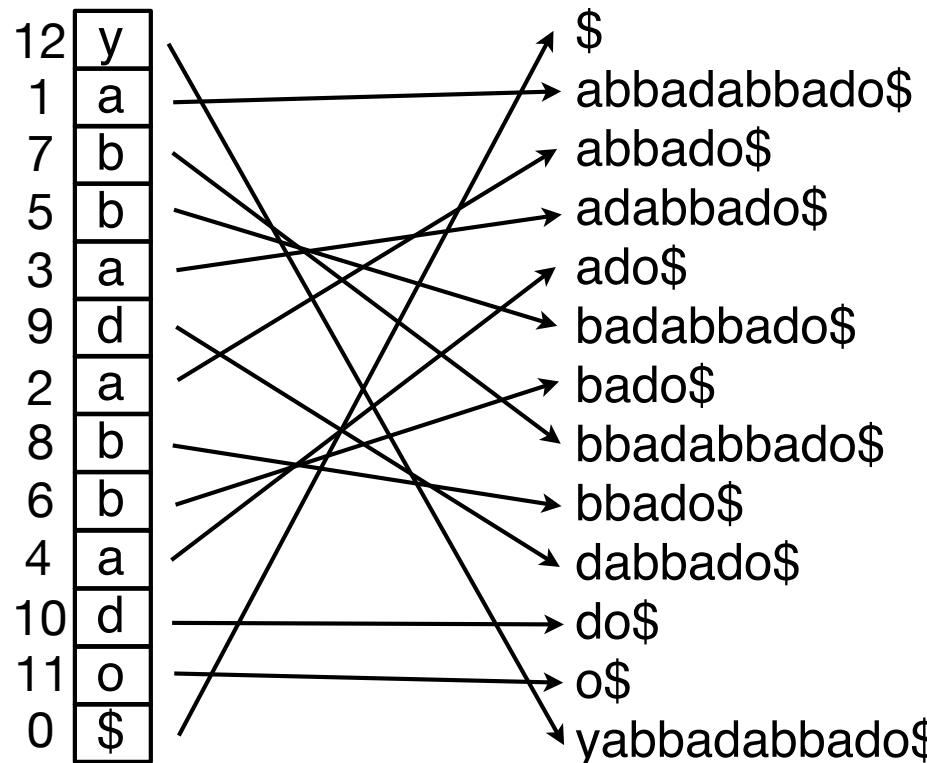
- Radix Sort [Hollerith 1887]. Sort sequence of n integers from $U = \{0, \dots, n^k - 1\}$.
 - Write each element in sequence as a base n integer $x = (x_1, x_2, \dots, x_k)$
 - Sort sequence according to each digit from right to left. Sorting should be **stable**.
- Time. $O(nk)$

Radix and Suffix Sorting

- Radix Sort
- Suffix Sort

Suffix Sort

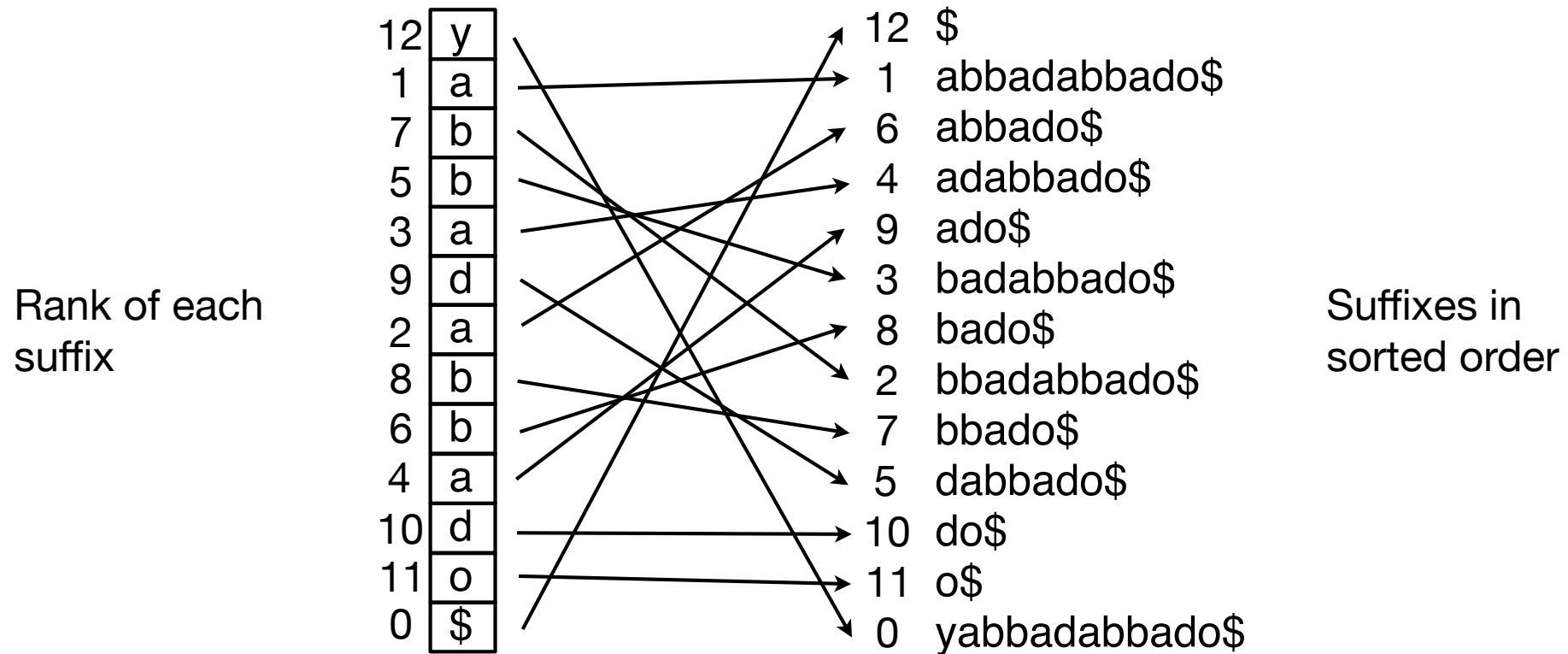
- **Suffix sorting.** Given string S of length n over alphabet Σ , compute the sorted lexicographic order of all suffixes of S .



- **Theorem [Kasai et al. 2001].** Given the sorted lexicographic order of suffixes of S , we can construct the suffix tree for S in linear time.

Suffix Sort

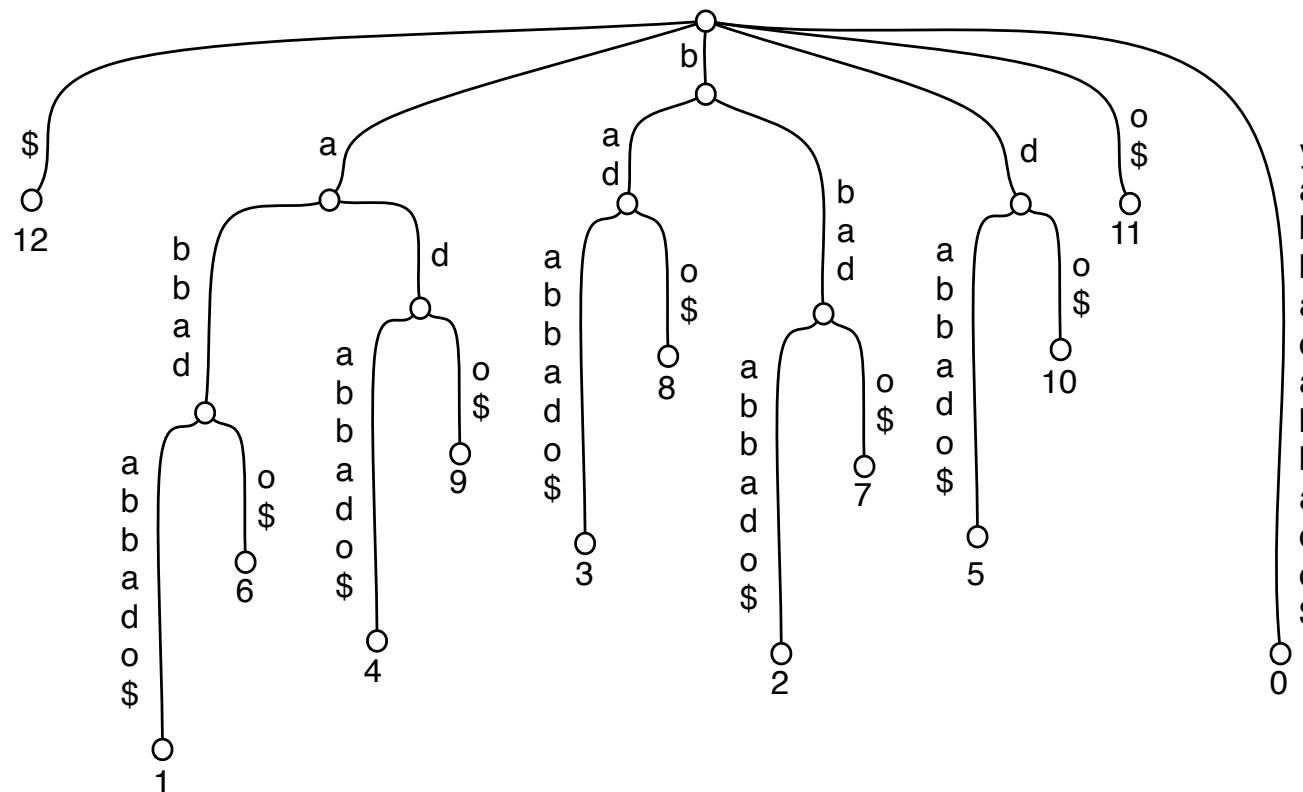
- **Suffix sorting.** Given string S of length n over alphabet Σ , compute the sorted lexicographic order of all suffixes of S .



- **Theorem [Kasai et al. 2001].** Given the sorted lexicographic order of suffixes of S , we can construct the suffix tree for S in linear time.

Suffix Sort

- **Suffix trees and sorting.** The lexicographic order of the suffixes is the same ordering as suffixes in the leaves of the suffix tree.
- **Suffix array.** The array of the sorted order of the suffixes.



12	1	6	4	9	3	8	2	7	5	10	11	0
----	---	---	---	---	---	---	---	---	---	----	----	---

Suffix Sort

- **Goal.** Compute the lexicographic order of all suffixes of S fast.
- For simplicity assume $|\Sigma| = O(n)$
- **Solution in 3 steps.**
 - Solution 1: Radix sorting
 - Solution 2: Prefix doubling
 - Solution 3: Difference cover sampling

Solution 1: Radix Sort

- Radix Sort.

- Generate all suffixes (pad with \$).
- Radix sort.

yabbadabbado\$
abbadabbado\$\$
bbadabbado\$\$\$
badabbado\$\$\$\$
adabbado\$\$\$\$\$
dabbado\$\$\$\$\$\$
abbado\$\$\$\$\$\$\$
bbado\$\$\$\$\$\$\$\$
bado\$\$\$\$\$\$\$\$\$
ado\$\$\$\$\$\$\$\$\$\$
do\$\$\$\$\$\$\$\$\$\$\$
o\$\$\$\$\$\$\$\$\$\$\$\$
\$\$\$\$\$\$\$\$\$\$\$\$\$\$

- Time. $O(n^2)$

Solution 2: Prefix Doubling

- **Prefix doubling [Manber and Myers 1990].** Sort substrings (padded with \$) of lengths 1, 2, 4, 8, ..., n. Each step uses radix sort on pair from previous step.

5	y
1	a
2	b
2	b
1	a
3	d
1	a
2	b
2	b
1	a
3	d
4	o
0	\$

8	51	ya
1	12	ab
4	22	bb
3	21	ba
2	13	ad
5	31	da
1	12	ab
4	22	bb
3	21	ba
2	13	ad
6	34	do
7	40	o\$
0	00	\$\$

10	84	yabb
1	13	abba
6	42	bbad
4	35	bada
2	21	adab
7	54	dabb
1	13	abba
6	42	bbad
5	36	bado
3	27	ado\$
8	60	do\$\$
9	70	o\$\$\$
0	00	\$\$\$\$

.....

- **Time.** $O(n \log n)$

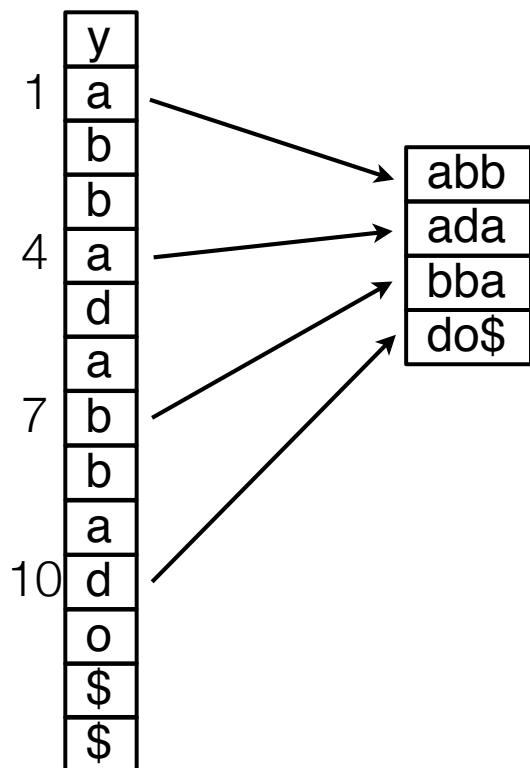
Solution 3: Difference Cover Sampling

- DC3 Algorithm [Karkkainen et al. 2003]. Sort suffixes in three steps:
 - Step 1. Sort sample suffixes.
 - Sample all suffixes starting at positions $i = 1 \bmod 3$ and $i = 2 \bmod 3$.
 - Recursively sort sample suffixes.
 - Step 2. Sort non-sample suffixes.
 - Sort the remaining suffixes (starting at positions $i = 0 \bmod 3$).
 - Step 3. Merge.
 - Merge sample and non-sample suffixes.

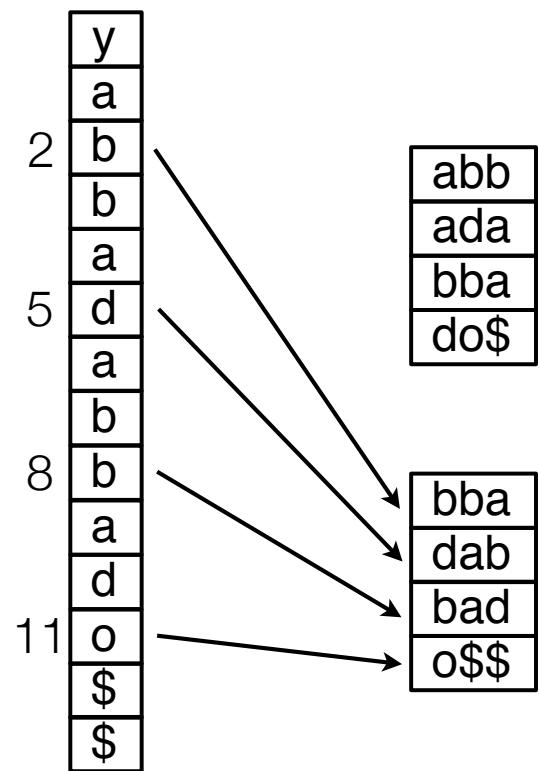
Step 1: Sort Sample Suffixes

y
a
b
b
a
d
a
b
b
a
d
o
\$
\$

Step 1: Sort Sample Suffixes



Step 1: Sort Sample Suffixes



Step 1: Sort Sample Suffixes

y
a
b
b
a
d
a
b
b
a
d
o
\$
\$

abb
ada
bba
do\$

bba
dab
bad
o\$\$

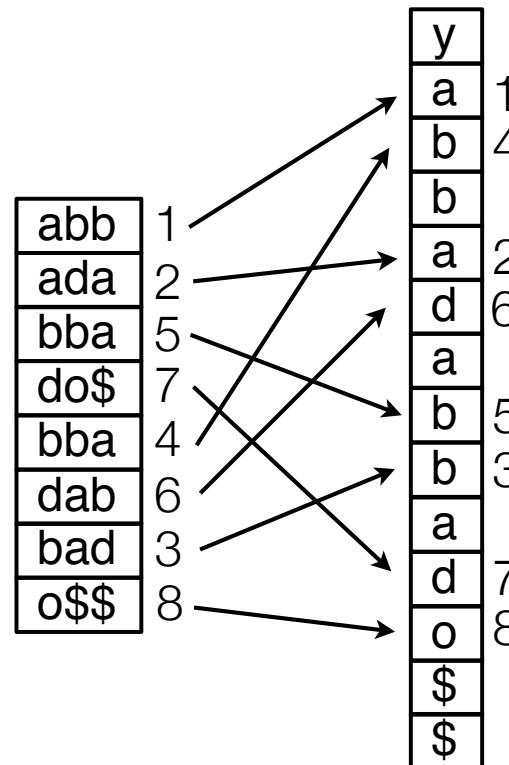
abb	1
ada	2
bba	5
do\$	7
bba	4
dab	6
bad	3
o\$\$	8

Step 1: Sort Sample Suffixes

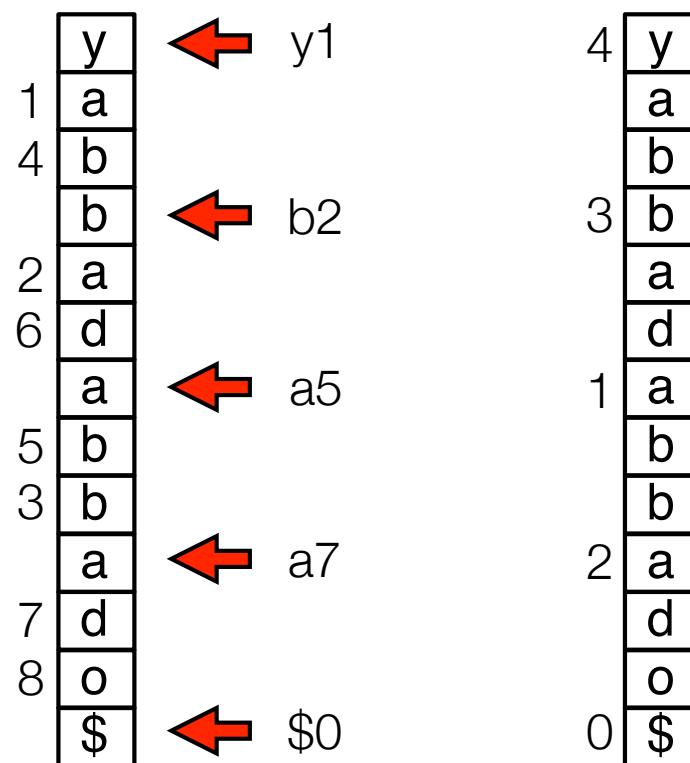
y
a
b
b
a
d
a
b
b
a
d
o
\$
\$

abb
ada
bba
do\$

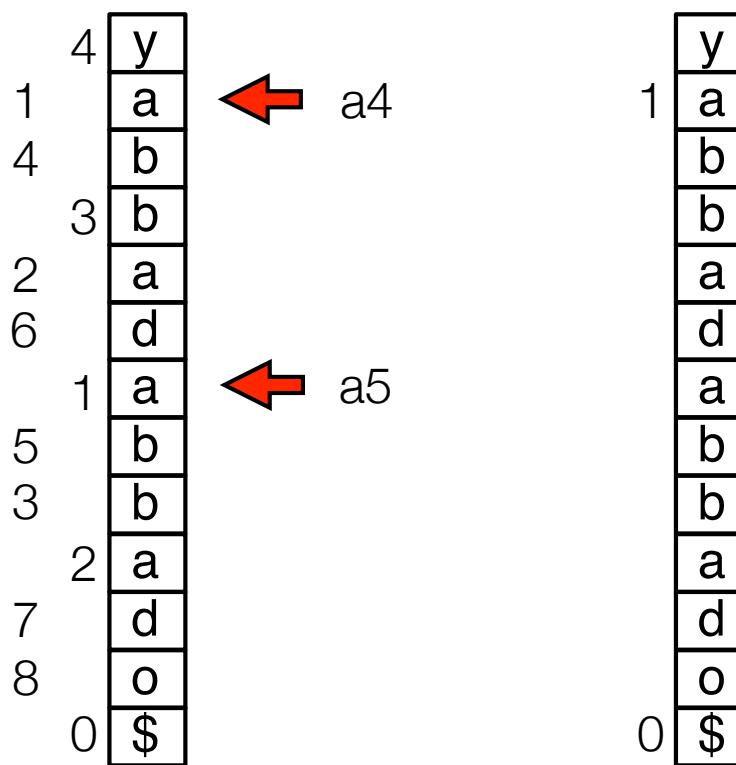
bba
dab
bad
o\$\$



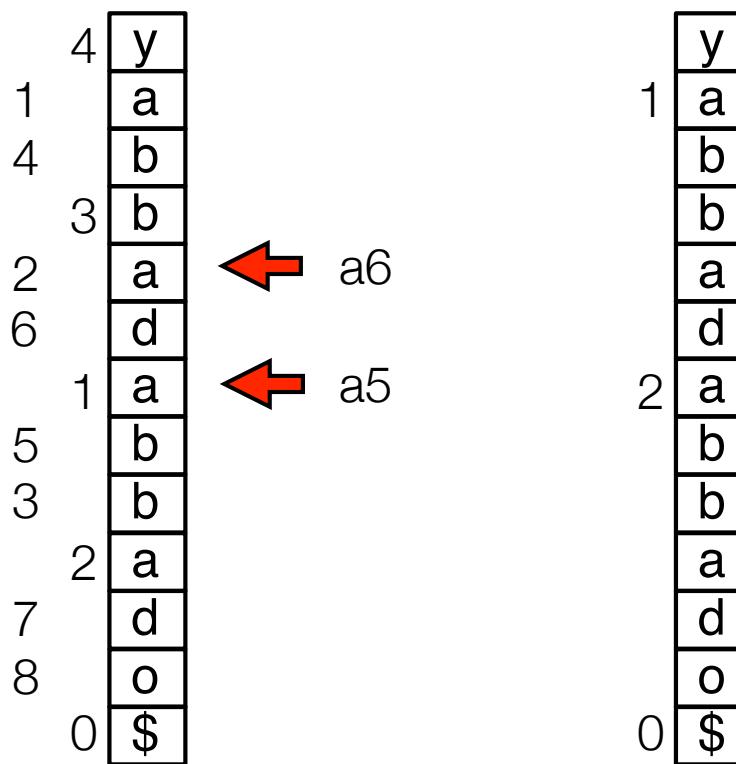
Step 2: Sort Non-Sample Suffixes



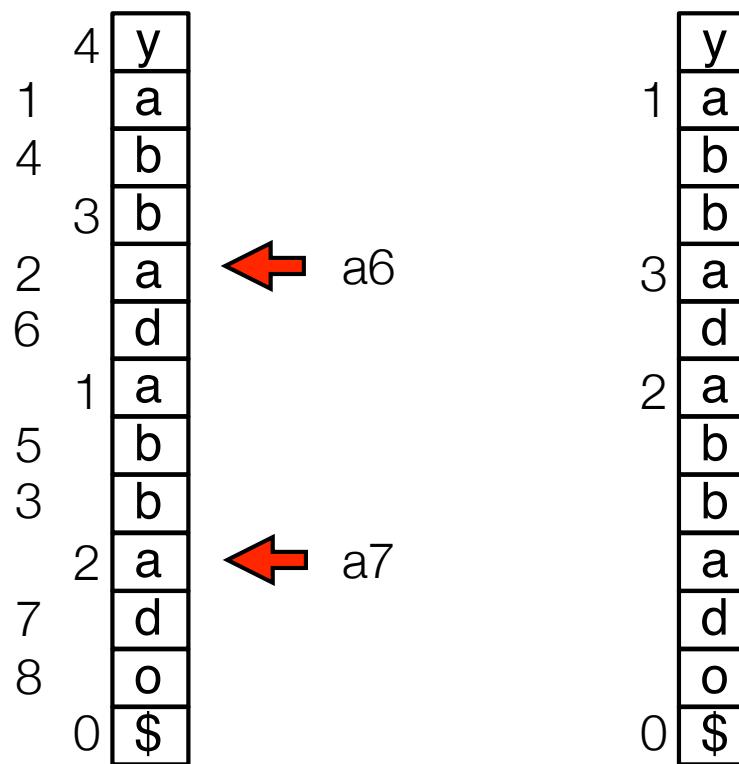
Step 3: Merge



Step 3: Merge



Step 3: Merge



Step 3: Merge

	4	y	
1		a	
4		b	
3		b	
2		a	
6		d	
1		a	
5		b	
3		b	
2		a	
7		d	
8		o	
0		\$	

	1	y	
1		a	
2		b	
3		b	
4		a	
5		d	
6		a	
7		b	
8		b	
9		a	
10		d	
11		o	
12		\$	

Step 3: Merge

	4	y	
1		a	
4		b	
3	b		
2		a	
6		d	
1		a	
5		b	
3	b		
2		a	
7		d	
8		o	
0	\$		

ba6 ba7

	1	y	
1		a	
5		b	
3	b		
2		a	
6		d	
2		a	
4		b	
4	b		
4		a	
0		d	
0		o	
0	\$		

Step 3: Merge

The diagram illustrates the merge step of a merge sort algorithm. It shows two arrays side-by-side.

Left Array:

4	y
1	a
4	b
3	b
2	a
6	d
1	a
5	b
3	b
2	a
7	d
8	o
0	\$

Right Array:

1	y
1	a
5	b
3	b
2	a
2	b
6	b
4	a
0	\$

Two specific elements are highlighted with red arrows pointing from the left array to the right array:

- A red arrow labeled "ya4" points to the element "y" at index 1 in the right array.
- A red arrow labeled "ba7" points to the element "b" at index 3 in the right array.

Step 3: Merge

	4	y	
1		a	
4		b	
3		b	
2		a	
6		d	
1		a	
5		b	
3		b	
2		a	
7		d	
8		o	
0		\$	

ya4

bb2

	1	y	
1		a	
7		b	
5		b	
3		a	
2		d	
2		a	
6		b	
6		b	
4		a	
0		d	
0		o	
0		\$	

Step 3: Merge

	4	y	
1		a	
4		b	
3		b	
2		a	
6		d	
1		a	
5		b	
3		b	
2		a	
7		d	
8		o	
0		\$	

\leftarrow y1 \leftarrow b3

	1	y	
1		a	
7		b	
5		b	
3		a	
2		d	
8		a	
6		b	
4		b	
4		a	
0		d	
0		o	
0		\$	

Step 3: Merge

The diagram illustrates the merge step in a merge sort algorithm. It shows two arrays, ya_4 and da_5 , being merged into a single array.

Array ya_4 :

4	y
1	a
4	b
3	b
2	a
6	d
1	a
5	b
3	b
2	a
7	d
8	o
0	\$

Array da_5 :

1	a
7	b
5	b
3	a
9	d
2	a
8	b
6	b
4	a
0	d
0	o
0	\$

Red arrows point from the arrays to the merge operation, indicating the flow of data from the arrays into the final merged array.

Step 3: Merge

	4	y	
1		a	
4		b	
3		b	
2		a	
6		d	
1		a	
5		b	
3		b	
2		a	
7		d	
8		o	
0		\$	

y1

	1	y	
1		a	
7		b	
5		b	
3		a	
9		d	
2		a	
8		b	
6		b	
4		a	
10		d	
0		o	
0		\$	

d8

Step 3: Merge

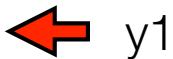
	4	y
1		a
4		b
3		b
2		a
6		d
1		a
5		b
3		b
2		a
7		d
8		o
0		\$

← y1

	1	y
1		a
7		b
5		b
3		a
9		d
2		a
8		b
6		b
4		a
10		d
11		o
0		\$

← o0

Step 3: Merge



	4	y	
1		a	
4		b	
3		b	
2		a	
6		d	
1		a	
5		b	
3		b	
2		a	
7		d	
8		o	
0		\$	

	12	y	
1		a	
7		b	
5		b	
3		a	
9		d	
2		a	
8		b	
6		b	
4		a	
10		d	
11		o	
0		\$	

Solution 3: Difference Cover Sampling

- DC3 Algorithm. Sort suffixes in three steps:
 - Step 1. Sort sample suffixes.
 - Sample all suffixes starting at positions $i = 1 \bmod 3$ and $i = 2 \bmod 3$. $O(n)$
 - Recursively sort sample suffixes. $T(2n/3)$
 - Step 2. Sort non-sample suffixes.
 - Sort the remaining suffixes (starting at positions $i = 0 \bmod 3$). $O(n)$
 - Step 3. Merge.
 - Merge sample and non-sample suffixes. $O(n)$
- $T(n) = \text{time to suffix sort a string of length } n \text{ over alphabet of size } n$
- Time. $T(n) = T(2n/3) + O(n) = O(n)$

Solution 3: Difference Cover Sampling

- **Theorem.** We can suffix sort a string of length n over alphabet Σ of size n in time $O(n)$.
- **Theorem.** We can suffix sort a string of length n over alphabet Σ $O(\text{sort}(n, |\Sigma|))$ time.

Radix and Suffix Sorting

- Radix Sort
- Suffix Sort