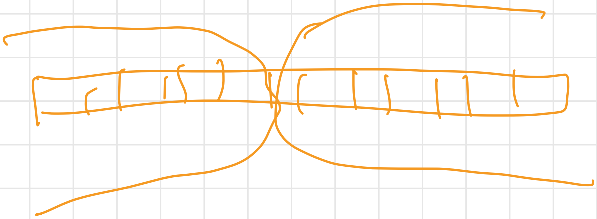
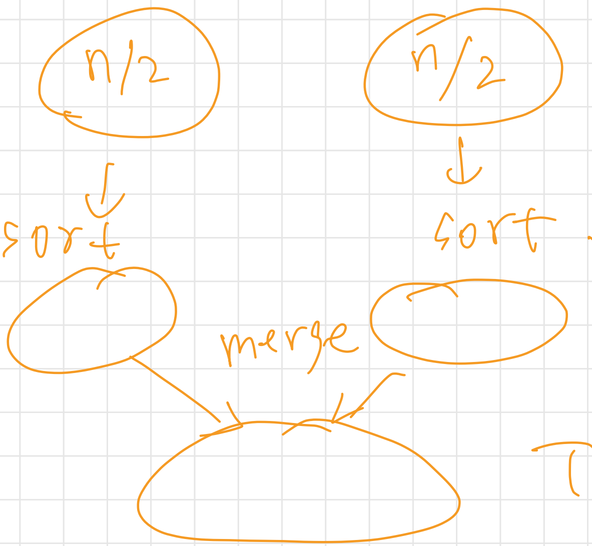
# Review Merge Sort



Divide.




$$T(n) = 2 \cdot T(n/2) + n$$

$$\Downarrow$$

$$T(n) = O(n \cdot \log n)$$

# Skew Algorithm For Suffix Sorting

- Let $S_0$, $S_1$, $S_2$, …, $S_{n-1}$ be all the n suffixes. $S_i$ starts at i-th position.
- Skew algorithm uses divide and conquer. But it divides the problem into unequally sized parts.
- Two sets $SA^0 = \{S_i : i = 0 \bmod 3\}$ and $SA^{12} = \{S_i : i=1 \text{ or } 2 \bmod 3\}$.

# Skew Algorithm Example

- Example: mississippi

0 1 2 3 4 5 6 7 8 9 10

SA$^0$ | SA$^{12}$ (SA$^1$ / SA$^2$)

| SA$^0$ | | SA$^1$ | | SA$^2$ | |
|---|---|---|---|---|---|
| 0: | mississippi | 1: | ississippi | 2: | ssissippi |
| 3: | sissippi | 4: | issippi | 5: | ssippi |
| 6: | sippi | 7: | ippi | 8: | ppi |
| 9: | pi | 10: | i | | |

✓ ① Sort SA$^{12}$ recursively

✓ ② Sort SA$^0$ in linear time

✓ ③ merge in linear time

$$T(n) = T(\tfrac{2}{3}n) + n + n = O(n).$$

# Skew Algorithm Example

- Example: mississippi

length $\approx \frac{2n}{3}$ $\neq$ 3mer

| $SA^0$ | $SA^{12}$ | |
|---|---|---|
| mississippi0 | ississippi00 | ssissippi |
| sissippi0 | issippi00 | ssippi |
| sippi0 | ippi00 | ppi |
| pi0 | i00 | |

Renaming:

3-mer → integer

Now we can do recursion.

except 3-mer.

# Skew Algorithm Example

- Example: mississippi

0 1 2 3 4 5 6 7 8 9 10

SA$^{12}$

| SA$^0$ | | SA$^1$ | | SA$^2$ |
|---|---|---|---|---|
| 0: mississippi | → | 1: mississippi | | 2: ssissippi |
| 3: sissippi | → | 4: sissippi | | 5: ssippi |
| 6: sippi | → | 7: sippi | | 8: ppi |
| 9: pi | → | 10: pi | | |

one-pass in radix sorting

m < s < p

1    4, 7    10

# Skew Algorithm Example

- Example: mississippi

  0 1 2 3 4 5 6 7 8 9 10

| SA$^0$ | | SA$^{12}$: SA$^1$ | | SA$^2$ |
|---|---|---|---|---|
| 0: | mississippi | 1: | ississippi | 2: ssissippi |
| 3: | sissippi | 4: | issippi | 5: ssippi |
| 6: | sippi | 7: | ippi | 8: ppi |
| 9: | pi | 10: | i | |

$i \rightarrow \quad \leftarrow j$

$a \in SA^0$ v.s. $b \in SA^1$

$a[0] \quad \in SA^1$

$b[0] \quad SA^2$

$a \in SA^0$ v.s. $b \in SA^2$

$a[0]\; a[1] \quad SA^2$

$b[0]\; b[1] \quad SA^1$

# Skew Algorithm For Suffix Sorting

- 1. Sort $SA^{12}$ recursively.
- 2. Sort $SA^{0}$ in linear time.
- 3. Merge sort $SA^{0}$ and $SA^{12}$ in linear time.

# How to sort $SA^{12}$ recursively

mississippi

$SA^{12}$

mod 1

```
ississippi
   issippi
      ippi
         i
```

mod 2

```
ssissippi
   ssippi
      ppi
```

- We need to know the order of these suffixes.
- In order to solve it recursively, we need to reduce the problem to a smaller suffix sorting problem.

# Reduction to a smaller suffix sorting problem

$SA^{12}$

mod 1

```
ississippi00
   issippi00
      ippi00
         i00
```

mod 2

```
ssissippi
   ssippi
      ppi
```

| | | | |
|---|---|---|---|
| i s s | i s s | i p p | i 0 0 |

| | | |
|---|---|---|
| s s i | s s i | p p i |

- Pad 0 to make their length multiple of 3. Then treat each string as a string of "triplets". Each subset is the suffixes of the "triplet string".
- We connect the two "triplet strings" together to make a longer string. We put the one with padding at the left.

| | | | | | | |
|---|---|---|---|---|---|---|
| i s s | i s s | i p p | i 0 0 | s s i | s s i | p p i |

# Reduction

SA$^{12}$

mod 1

```
ississippi00
   issippi00
      ippi00
         i00
```

mod 2

```
ssissippi
   ssippi
      ppi
```

```
ississippi00ssissippi
   issippi00ssissippi
      ippi00ssissippi
         i00ssissippi
            ssissippi
               ssippi
                  ppi
```

- Now check all the suffixes of the concatenated triplet string. Their relative order can be used to build the relative order of SA$^{12}$ easily.
- We are almost there, except that keeping tripling the size (number of bytes) of the "character" is a problem.

# Renaming

- We solve the unlimited expansion problem by a trick called renaming. It maps each unique triplet to a single unique integer.
- To rename, we first sort the triplets, and then assign integer values sequentially to unique triplets. Sorting triplets can be done in linear time by radix sort.
- This ensures
  - The max value is always bounded by the length of array.
  - The suffix order is unchanged.

```
i00 -> 0
ipp -> 1
iss -> 2
ppi -> 3
ssi -> 4
```

```
ississippi00ssissippi
          ⇓
 2  2  1  0  4  4  3
```

# Renaming Example

```
i00  ->  0
ipp  ->  1
iss  ->  2
ppi  ->  3
ssi  ->  4
```

```
ississippi00ssissippi

2  2  1  0  4  4  3
```

```
ississippi00ssissippi       2 2 1 0 4 4 3
   issippi00ssissippi         2 1 0 4 4 3
      ippi00ssissippi   =>      1 0 4 4 3
         i00ssissippi             0 4 4 3
            ssissippi               4 4 3
               ssippi                 4 3
                  ppi                   3
```

# Recursion

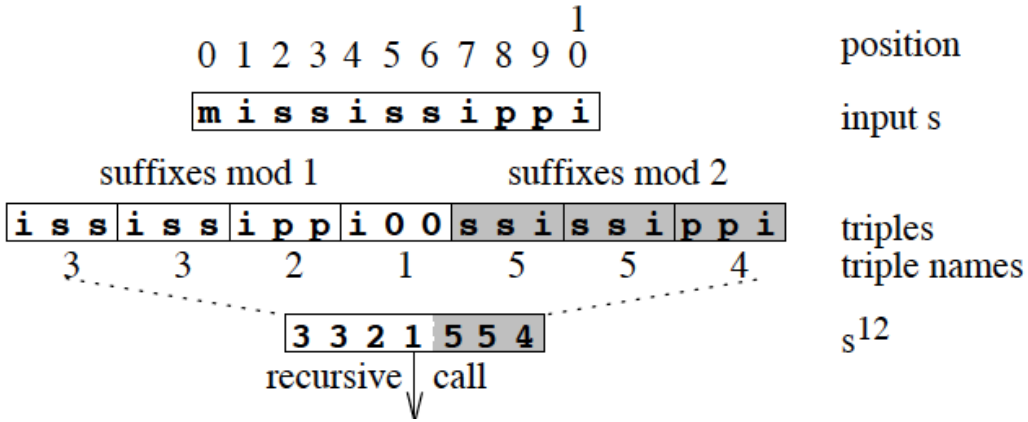
- After renaming, we just suffix sort the new integer string, which has length approximately $2n/3$. This can be done by recursion.
- The time complexity of renaming is dominated by sorting the triplets. This can be solved in linear time with radix sort.

# Radix Sort

- **Radix Sort**: Multiple passes. Each pass stable sorts according to one digit. From the least to the most significant digit.
- original: its, iss, ipp, abc, att
- pass1: abc, ipp, its, iss, att
- pass2: abc, ipp, iss, its, att
- pass3: abc, att, ipp, iss, its

- Radix sorting requires O(k) space, where k is the size of the alphabet.
- Each pass takes linear time. And only 3 passes needed in our case. So it is linear time.

# Recap Sort $S^{12}$ recursively




1. Padding and concatenation to get string of triplets.
2. Radix sort the triplets to get an ID (name) of each triple.
3. Recursion to get the suffix order on the string of IDs.

# Skew Algorithm For Suffix Sorting

- We assume $SA^{12}$ is sorted already, and learn the other two steps first.
- 1. Sort $SA^{12}$ recursively.
- 2. Sort $SA^{0}$ in linear time.
- 3. Merge sort $SA^{0}$ and $SA^{12}$ in linear time.

# Sort $S^0$ in linear time

- $S_i = s[i]\ S_{i+1}$.
- For all $S_i$ in $SA^0$, $S_{i+1}$ has been sorted already. Use s[i] to do another pass of radix sorting will give us the right order of $SA^0$. This takes linear time.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| m | i | s | s | i | s | s | i | p | p | i |

Sorted SA12

```
10: i
 4: issippi
 1: ississippi
 7: ippi
 8: ppi
 5: ssippi
 2: ssissippi
```

To sort SA0

```
0: mississippi
3: sissippi
6: sippi
9: pi
```

# Skew Algorithm For Suffix Sorting

- 1. Sort $SA^{12}$ recursively.
- 2. Sort $SA^{0}$ in linear time.
- 3. Merge sort $SA^{0}$ and $SA^{12}$ in linear time.

# Merge

Sorted SA12

```
10: i
 4: issippi
 1: ississippi
 7: ippi
 8: ppi
 5: ssippi
 2: ssissippi
```

Sorted SA0

```
0: mississippi
9: pi
6: sippi
3: sissippi
```

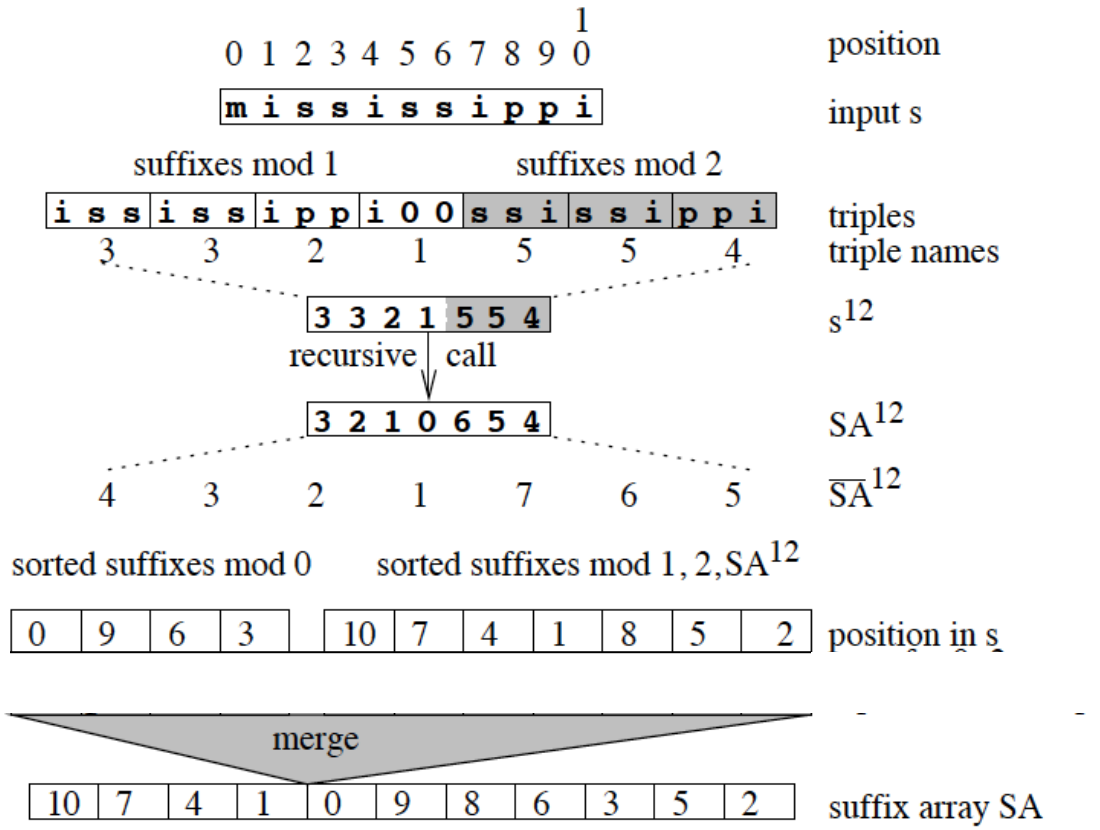
- Would be a simple merge if comparison of two takes constant time.
- Trouble is when two suffices share a long prefix, which takes more than constant time to compare.

E.g. what if S5 = aaaa… and S6=aaaa…

# Merge $S^0$ and $S^{12}$

- Merging only requires to compare a suffix $S_j$ with j mod 3 = 0 with a suffix $S_i$ with i mod 3 != 0. :
- Case 1: If i mod 3 = 1, we write $S_i$ as (s[i], $S_{i+1}$) and $S_j$ as (s[j], $S_{j+1}$).
  - Since (i +1) mod 3 = 2 and (j + 1) mod 3 = 1, the relative order of $S_{j+1}$ and $S_{i+1}$ can be determinded from their position in $SA^{12}$.
- Case 2: If i mod 3 = 2, we compare the triples (s[i], s[i + 1], $S_{i+2}$) and (s[j], s[j + 1], $S_{j+2}$).

# Recap



1
0 1 2 3 4 5 6 7 8 9 0
position
m i s s i s s i p p i
input s
suffixes mod 1
suffixes mod 2
i s s i s s i p p i 0 0 s s i s s i p p i
triples
3 3 2 1 5 5 4
triple names
3 3 2 1 5 5 4
s12
recursive call
3 2 1 0 6 5 4
SA12
4 3 2 1 7 6 5
SA12
sorted suffixes mod 0
sorted suffixes mod 1, 2, SA12
0 9 6 3
10 7 4 1 8 5 2
position in s
merge
10 7 4 1 0 9 8 6 3 5 2
suffix array SA

# C codes

- 50 lines of C++ codes were given in J.C.M. Baeten et al. (Eds.): ICALP 2003, LNCS 2719, pp. 943–955, 2003.

- http://www.mpi-inf.mpg.de/~sanders/programs/suffix/