- Given a text
- Consider all suffixes
- sort them alphabetically

T = mississippi#

P = si

#
i#
ippi#
issippi#
ississippi#
mississippi#
pi#
ppi#
sippi#
sissippi#
ssippi#
ssissippi#

P = si

1) Suffixes that share a prefix are contiguos

2) Pattern is "positioned" before the occurencies

⇒ so just scan to find the occurrences

⇒ Suffix array : STORES for every suffix the offset of the 1^{st} character

	1	2	3	4	5	6	7	8	9	10	11	12
T =	m	i	s	s	i	s	s	i	p	p	i	#

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

length of the text ~~strings~~

SA = $\Theta(N \log \sigma)$ BITS

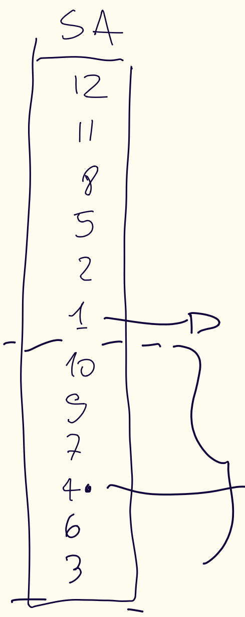
N = # characters of the text

• SEARCH A PATTERN on $\boxed{\text{SA}}$

=> Binary search

P = si

(1) => pick the middle element of SA



1 = mississippi

(2) COMPARE WITH P: si

=> S > M

=> P will be in the second part of the array

compare

(3) REPEAT

=> ~~(1)~~ pick middle element

=> I do $\boxed{\log_2 n \text{ steps.}}$

∀ step I compare $|P|$ characters at most

=> Cost of search $= O(p \log_2 n)$

OCCURRENCES

=> when I've found position of P

=> SCAN the SA and compare => BUT is COSTLY

=> search for P# and P\$ where $\# < \Sigma < \$$

as char of alphabet

=> 2 BINARY SEARCHES TO FIND BEGINNING and the end => I know ~~#~~ of occurrences

$$\Rightarrow O(\underbrace{\underbrace{p \cdot \log n}_{\text{only to count}} + \text{occurrences}}_{\text{To count and locate the occurrences}})\ \text{time}$$

=> Counting occurrences $O(p \cdot \log n)$ time

Retrival $O(p \cdot \log_2 n + occ)$ time

```
Building SA (char *T, int *n, char **SA)
    for (i=0; i<n; i++){
        SA[i] = T+i;
    }
    qsort(SA, n, sizeof(char *), suffix_cmp)
```

— o —

```
suffix_cmp(char **p, cha **q){
        return str_cmp(*p, *q) }
```

$$O(\overset{\text{SCAN}}{\frac{M}{B}} \cdot n \log n)\ \text{I/Os}$$

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

LCP-ARRAY

↳ longest common prefix

(total number of characters shared the longest prefix by adjacent suffixes)

	1	2	3	4	5	6	7	8	9	10	11	12
T =	m	i	s	s	i	s	s	i	p	p	i	#

LCP	SA	
0	12	#
0	11	i#
1	8	ippi#
4	5	issippi#
0	2	ississippi#
0	1	mississippi#
1	10	pi#
0	9	ppi#
2	7	sippi#
1	4	sissippi#
	6	ssippi#
3	3	ssissippi#

min

P = si

n-1 position

LCP(5, 9) = 0 = min(4, 0, 0) = 0

```
Build SUFFIX-ARRAY (char *T, char *m, char **SA){
        for( i=0; i<m; i++){
              SA[i]=T; }
              QSORT(SA, m, sizeof(char *), suffix_cmp)
    Suffix_cmp (char **p, char **q)
              return string_cmp(*p, *p)
```

cost $O\left(\frac{M}{B} n \log n\right)$ ✓

```
Buildg SA ( char *T, char *n, char **SA)

    for (i=0 ; i<n; i ++){
        SA[i] = T + i; }
      QSORT ( SA, n, sizeof(char *), suffix_cmp)
   }

suffix_cmp (char *p , char *q) {
        return strg--cmp(*p , *q)
```

=> Complexity $O(\frac{n}{B} n \log n$

SNOW_PLOW

```
- Require U array of unsorted items
- H = build a min-heap over U- items
- set U = ∅
while ( U ≠ 0 )
        min = extract the minimum item from H
        next = read next item from U
        if (next < min)
                put next in U
        else
                put next in the heap
    end if
end while
```

$$O\left(\frac{n}{B}\log_2\frac{n}{2M}\right)$$

=> snow plow produces sorted runs of length 2M

=> long with & if (~~min~~ next < min)

and assuming RANDOM distribution ~~on~~ fetching items

$$\left((k-M)+M=k\right)$$

=> M items ends up in U $P=\frac{1}{2}$ => I need k items

$M = \frac{K}{2}$ $\boxed{K = 2M}$

```
Buid SA (char *T, char *n, char **SA) {
        for (i=0; j<i; i++){
              SA[i] = T+i;
        Qsort (char SA, char n, size_of (char *), suffix_cmp)
        }
  }
suffix_cmp (char **p, char **q)
        return string_cmp (*p, *q)
```

$$O\left(\frac{n}{B} \cdot n \log_2 n\right)$$