PATTERN MATCHING BASICS

→ first discussing the Most Naive approach.

s → b c a d c b c d

p → b c d

```
for( i=0 ; i <= len(s) - len(p) ; i++){
    bool isfound= true;
    for(j=0 ; j < len(p) ; i++){
        if(p[i] == s[i+j])
            continue;
    }
}
```

Time Complexity : $O(n*m)$

KMP ALGORITHM ≃ O(n)

- Time Complexity : O(string length) + O(Pattern length)

- we need a longest prefix which is also a suffix.

S : a b c z a b c g a b z a b c g a b c g a b c y

P : a b c g a b c y

LONGEST PREFIX SUFFIX EXPLAIN

eg :-

works in O(n)

0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10	11	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	d	a	d	a	b	a	d	a	d	a	b	a	d	a	d	a	d

Z- ALGORITHM PATTERN MATCHING

z - function → z(k) → length of longest substring starting at k which is also the prefix of the string.

Time Complexity for z-array is $O(m+n)$.

z-array

a	a	c	b	y	a	a	k	a	a	c
0	1	0	0	0	2	1	0	3	1	0

Now,

text = y a b c k a b d a b c

Pattern = a b c

we will generate, Pattern + '$' + Text

=

a	b	c	$	y	a	b	c	k	a	b	d	a	b	c
0	0	0	0	0	3	0	0	0	2	0	0	3	0	0

↑

i - p.length() - 1