Text (here a genomic sequence):

```
G A T T A C A T
0 1 2 3 4 5 6 7
```

- Be careful about indices: These ideas arose outside genomics, so algorithms here use 0-indices; you will need to convert to genomic coordinates later!

Text with end marker (here a $ symbol):

```
G A T T A C A T $
0 1 2 3 4 5 6 7 8   <-- because of 0-indexing, index of $ is length of original text (here 8)
```

Suffixes of text:

```
G A T T A C A T $    0   <-- the suffix beginning at index 0
A T T A C A T $      1
T T A C A T $        2
T A C A T $          3
A C A T $            4
C A T $              5
A T $                6
T $                  7
$                    8
```

Sorted suffixes ($ always sorts first):

```
$                    8   <-- suffix array (should always start with last index!)
A C A T $            4
A T $                6
A T T A C A T $      1
C A T $              5
G A T T A C A T $    0
T $                  7
T A C A T $          3
T T A C A T $        2
```

Cyclic permutations (or rotations):

```
G A T T A C A T $
A T T A C A T $ G
T T A C A T $ G A
T A C A T $ G A T
A C A T $ G A T T
C A T $ G A T T A
A T $ G A T T A C
T $ G A T T A C A
$ G A T T A C A T
```

Burrows-Wheeler Transform (BWT):

Sorted cyclic permutations (also called the Burrows-Wheeler matrix, BWM):

```
$ G A T T A C A T
A C A T $ G A T T
A T $ G A T T A C
A T T A C A T $ G
C A T $ G A T T A
G A T T A C A T $
T $ G A T T A C A
T A C A T $ G A T
T T A C A T $ G A  <-- last *column* of the BWM is the BWT of the text: TTCGA$ATA
```

* Why is the BWT the last column of the BWM and not some other column in the middle? After all, every single column is a permutation of the original text -- what's so special about the one in the last column?

================================

FM-index:

In this data structure, we name the *first* column of the BWM as F, and the *last* as L
We also keep around the suffix array from above:

SA	F		L	
8	$	...	T	
4	A	...	T	
6	A	...	C	
1	A	...	G	
5	C	...	A	
0	G	...	$	
7	T	...	A	<--
3	T	...	T	<--
2	T	...	A	<--

* Note: L is the same as the BWT, and F is nothing more than a sorted version of L (why?).
  Note also: L tells you the character that precedes the corresponding one in F (why?).

* This means we can search for queries "backwards". E.g., if we want to know all the places the query CAT appears as a substring within the text, we first need to identify all the locations of a T (i.e., all the suffixes that start with a T), and then narrow down to those that have an A before the T, and then narrow down to those that have a C before the A. Anything that remains must be an occurrence of the query substring CAT.

* We can use what we have above to find all the suffixes that start with a T (the ones indicated with arrows), and among those, we see that two of the three Ts are preceded by As in the text. We're almost there! All we need to know now is, "Which of those two As are preceded by a C in the text?" But how?

* It would be awesome if we had a mapping that could magically tell us that the specific character in row i of L is the same one as the one appearing in row j of F. We call this an LF mapping, because it maps a character in L back to the corresponding character in F.