# **COMPLEXITY ANALYSIS**

## **(BURROWS-WHEELER TRANSFORMATON ALGORITHM)**

### TIME COMPLEXITY ANALYSIS OF THE ALGORITHM:

**STEP-1: Generating Rotations**

* Generating cyclic rotations involves creating substrings of the input text.
* For each rotation, we create a substring by concatenating two substrings, which takes O(n) time, (where n is the length of the text).
* We do this for each character of the text, the overall time complexity for generating rotations is O(n^2).

**STEP-2: Sorting**

* After generating the rotations, the algorithm sorts them based on their suffixes.
* The ‘sort’ function has an average-case time complexity of O (n log n), where n is the number of rotations.
* Since there are n rotations, the sorting step takes O (n log n) time.

**STEP-3: Extracting Transformed Text**

* After sorting, the algorithm extracts the last characters of each rotation to form the Burrows-Wheeler Transform.
* This step requires iterating over the sorted rotations once, which takes O(n) time.

**Overall Time Complexity:**

* The dominant factor in the time complexity is the sorting step, which is O (n log n).
* Therefore, the overall time complexity of the Burrows-Wheeler Transform algorithm implemented in the provided code is O (n log n), (where n is the length of the input text).

### SPACE COMPLEXITY ANALYSIS OF THE ALGORITHM:

**1. Rotations Vector:**

* + The algorithm creates a vector of structures to store the cyclic rotations along with their indices. Each structure (`RotationIndex`) contains an integer (`index`) and a string (`rotation`).
  + The space required to store this vector depends on the number of rotations, which is equal to the length of the input text (n).
  + Since each rotation string has a length equal to the length of the input text, the space complexity of storing all rotations is O(n^2).

**2. Transformed Text String:**

* The algorithm constructs a string (`transformedText`) to store the Burrows-Wheeler Transform.
* This string contains only the last characters of each rotation and has a length equal to the length of the input text (n).
* Therefore, the space complexity of storing the transformed text string is O(n).

**3. Other Variables:**

* They do not significantly contribute to the overall space complexity.

**Overall Space Complexity:**

* The dominant factor in the space complexity is the vector of rotations, which requires O(n^2) space.
* Therefore, the overall space complexity of the Burrows-Wheeler Transform algorithm implemented in the provided code is O(n^2).