ALGORITHM [EASY 1]: -

```
Begin
   // Initialize the index to the last character of the string
   // Initialize a variable to store the length of the last word
    while // Loop from the end of the string towards the beginning
          if // Check if the current character is an alphabet character
                    // If it is, increment the length of the last word and move to the previous character
                    End if
          else
                    if // If the current character is not an alphabet character
                              // If the length of the last word is already greater than 0, break the loop
                    End if
                     else
                               // If the length of the last word is 0, move to the previous character
                     End else
          End else
     End while
   // Return the length of the last word
End
```

EXPLANATION [EASY 1]: -

- The function starts by initializing the index ‘size’ to the last character of the string and the variable ‘x’ to 0.
- It then enters a loop that iterates from the end of the string towards the beginning.
- Inside the loop, it checks if the current character at index ‘size’ is an alphabet character using the ‘isalpha’ function.
- If the character is an alphabet character, it increments the length of the last word (‘x’) and moves to the previous character by decrementing the index ‘size’.
- If the character is not an alphabet character, it checks if the length of the last word is already greater than 0. If yes, it breaks out of the loop. Otherwise, it just moves to the previous character.
- Finally, the function returns the length of the last word (‘x’).

ALGORITHM [MEDIUM 2]: -

begin

// Counters for the potential majority elements

// Potential majority element candidates

// First pass to find potential majority elements

// Second pass to count occurrences of the potential majority elements.

// Check if the counts of potential majority elements are greater than n/3 and add them to the result.

//return result

End

EXPLANATION [MEDIUM 2]: -

- The algorithm uses two counters (‘count1’ and ‘count2’) and two candidate variables (‘candidate1’ and ‘candidate2’) to keep track of potential majority elements.
- In the first pass through the array, it updates the candidates and counts based on certain conditions. If a counter becomes zero, it updates the corresponding candidate. If the current number is the same as one of the candidates, it increments the corresponding count. If the current number is different from both candidates, it decrements both counts.
- After the first pass, the algorithm has two potential majority elements. It then performs a second pass through the array to count the occurrences of these potential majority elements.
- Finally, it checks if the counts of potential majority elements are greater than ⌊n/3⌋, where n is the size of the array. If yes, it adds them to the result vector.
- The main function takes user input for the size and elements of the array, calls the ‘majorityElement’ method, and prints the result.

ALGORITHM [HARD 2]: -

begin

// Reverse the string

// Concatenate the original and reversed strings with a special character "#"

// Create an array to store the Longest Prefix Suffix (lps) values

// Build the lps array using the KMP algorithm

// The length of the palindromic prefix is given by the last element of the lps array

// Append the non-palindromic part of the reversed string to the original string

End

EXPLANATION [HARD 2]: -

The algorithm starts by reversing the given string ‘s’ to create the reversed string ‘rev’.

It concatenates the original and reversed strings with a special character "#" to form a new string ‘s_new’.

The algorithm then initializes an array ‘lps’ of length ‘n_new’ (length of ‘s_new’) to store the Longest Prefix Suffix values.

The KMP algorithm is used to build the ‘lps’ array. The algorithm iterates through the characters of ‘s_new’ and updates the ‘lps’ values based on the pattern matching logic.

After constructing the ‘lps’ array, the length of the palindromic prefix is given by the last element of the array (‘lps[n_new - 1]’).

Finally, the algorithm appends the non-palindromic part of the reversed string to the original string, creating the shortest palindrome.

The main function takes user input, calls the ‘shortestPalindrome’ method, and prints the result.