Z - Algorithm

We first define the Z function of the string - The Z function of a string S is an array Z of length same as that of S such that Z[i] denotes the length of the largest prefix that matches from the substring starting at position i.

For example :
For the pattern “AAAABAA”,
Z[] is [0, 3, 2, 1, 0, 2, 1]
Z[0] is 0 by definition. The longest prefix that is also the prefix of string s[1..6] which is “AAABAA” is 3(This is equal to “AAA”). Similarly, For the whole string AAABAAA it is 1, hence the Z[6] is 1 since s[6.. 6] is ‘A’ and that is the longest possible prefix we can match.

Algorithm for Computing the Z array.
The idea is to maintain an interval [L, R] which is the interval with max R such that [L, R] is a prefix substring (substring which is also prefix).

- if i > R, no larger prefix-substring is possible.
- Compute the new interval by comparing S[0] to S[i] i.e. string starting from index 0 i.e. from start with substring starting from index i and find z[i] using z[i] = R - L + 1.
- Else if, i ≤ R, [L, R] can be extended to i.
- For k = i - L, Z[i] ≥ min(Z[k] , R - i + 1).
- If Z[k] < R - i + 1, no longer prefix substring s[i] exist.
- Else Z[k] ≥ R - i + 1, then there can be a longer substring.
- update [L, R] by changing L = i and changing R by matching from S[R+1]

```
function ZArray(s)
        //   initialize to all zeroes
        Z = array[n];
        //   set the current window to the first character
        l = 0
        r = 0

        for i from 1 to n - 1
                //   first case i <= r
                if i <= r
                        z[i] = min (r - i + 1, z[i - l]);

                //   increase prefix length while they are matching
                while i + z[i] < n and s[z[i]] == s[i + z[i]]
                        z[i] += 1;

                //   update the window if i + z[i] crosses the window
                if i + z[i] - 1 > r
                l = i
                r = i + z[i] - 1;
    //   return the array
    return z
```

Time Complexity: O(N), where N is the length of the pattern.

Algorithm for searching the pattern.

Now consider a new string S’ = pattern + ‘#’ + ‘text’ where + denotes the concatenation operator. Now, what is the condition that pattern appears at position [i. …i + M - 1] in the string text. The Z[i] should be equal to M for the corresponding position of i in S’. Note that Z[i] cannot be larger than M because of the ‘#’ character.

- Create S’ = pattern + ‘#’ + ‘text’
- Compute the lps array of S’
- For each i from M + 1 to N + 1 check the value of lps[i].
- If it is equal to M then we have found an occurrence at the position i - M - 1 in the string text.

```
function StringSearchZ_Algo(text, pattern)
    //   construct the new string
    S' = pattern + '#' + text

    //   compute its prefix array
    Z = ZArray(S')
    N = text.length
    M = pattern.length

    for i from M + 1 to  N + 1
        //   longest prefix match is equal to the length of pattern
        if Z[i] == M
            //   print the corresponding position
            print the occurrence i - M - 1
    return
```

Time Complexity: O(N + M), where N is the total length of the pattern and M is the length of the pattern we need to search.