Finding Array of Longest Prefix Sum

1. Problem Statement

Given a string `s`, compute the **Longest Prefix Suffix (LPS) Array**, where `LPS[i]` stores the length of the **longest proper prefix** of `s[0...i]` that is also a **suffix**.

A **proper prefix** is a prefix that is **not equal to the full string**.

Example:

Input: `"abacab"`

Output LPS Array: `[0, 0, 1, 0, 1, 2]`

---

2. Approach

1. **Initialize:**
    - Create an `lps` array of size `n`, initialized to `0`.
    - Use a variable `j = 0` to track the length of the previous longest prefix-suffix.
2. **Iterate over the string (`i = 1` to `n-1`)**
    - If `s[i] == s[j]`, increment `j` and set `lps[i] = j`.
    - Otherwise, reduce `j` using `lps[j-1]` until a match is found or `j = 0`.
3. **Return the computed `lps` array**

---

3. Complexity Analysis

- **Time Complexity:** `O(n)`
    - Each character is processed at most **twice**, making it linear.
- **Space Complexity:** `O(n)`
    - The `lps` array takes **O(n) space**.

---

4. Example Execution

Input: `"abacab"`

LPS Calculation Step by Step:

```
makefile
CopyEdit
Index:   0  1  2  3  4  5
String:  a  b  a  c  a  b
LPS:     0  0  1  0  1  2
```

Output:

```
javascript
CopyEdit
LPS Array: 0 0 1 0 1 2
```

---

5. Use Cases

- **Pattern Matching Algorithms** (e.g., KMP Algorithm).
- **Detecting String Repetitions** (e.g., checking cyclic patterns).
- **Lexical Analysis in Compilers** (e.g., substring preprocessing).
- **Text Processing in NLP** (e.g., finding repeated words or phrases).

---

6. Code Implementation (C++)

```
cpp
CopyEdit
#include <iostream>
#include <vector>

using namespace std;
```

```
vector<int> computeLPS(const string &s)
{
    int n = s.size();
    int j = 0;
    vector<int> lps(n, 0);

    for(int i = 1; i < n; i++)
    {
        while(j > 0 && s[i] != s[j])
        {
            j = lps[j - 1];
        }
        if(s[i] == s[j])
        {
            j++;
        }
        lps[i] = j;
    }
    return lps;
}

int main()
{
    string s = "abacab";
    vector<int> lps = computeLPS(s);
    cout << "LPS Array: ";
    for (int x : lps) cout << x << " ";
    cout << endl;
    return 0;
}
```

7. Summary

- This function computes the **LPS Array** in **O(n) time** using a **single pass** over the string.

- It is used in **string pattern matching, text processing, and automata design**.
- The approach ensures efficiency by **reusing previously computed LPS values** instead of brute force comparisons.