Concept	Meaning
prefix common
Prefix Sum	Sum of elements before i
Prefix XOR	XOR of elements before i
Prefix Max/Min	Max/Min till i
Prefix Frequency	Count of elements till i
Prefix Hash	Hash of prefix (strings)
Prefix GCD	GCD of A[0..i]
Prefix AND/OR	Bitwise accumulation
Prefix Common Count	Shared items till i

Concept	Meaning
suffix common
Suffix Sum	Sum from i to end
Suffix Max/Min	Max/Min from i to end
Suffix Frequency	Count from i onward
Suffix GCD	GCD of A[i..n-1]
Suffix AND/OR	Bitwise from i to end
Suffix Hash	Hash of suffix

Remove one element problems
Technique	Used in
Prefix + suffix arrays	Split problems
Difference array	Range updates
Imos technique	Interval additions
Sliding window	Variable prefix
Two pointers	Prefix + suffix movement
Meet in middle	Left-right decomposition

explain all these concepts in details with proper theory

proper exmaples for each and proper codes for each concepts in very very details
give codes in cpp and python

Great table 👍 — this is basically the **entire "prefix–suffix toolkit" of competitive programming & array/string algorithms**.
I'll explain **each concept from first principles → intuition → example → code**.

I'll structure it cleanly so you can **revisit this as notes**.

0️⃣ What are Prefix & Suffix Concepts (Big Picture)

- **Prefix** = information from index `0` → `i`
- **Suffix** = information from index `i` → `n-1`

Why they exist:

> To answer queries or make decisions **without recomputing** things again and again.

Instead of O(n) per query → **O(1)** per query after preprocessing.

1️⃣ PREFIX CONCEPTS (Detailed)

Let array be:

```ini
A = [2, 4, 6, 8, 10]
```

Copy code

---

1. Prefix Sum

Meaning

`pref[i] = A[0] + A[1] + ... + A[i]`

Why useful

- Range sum queries
- Remove-one-element problems
- Subarray sum logic

Example

```ini
pref = [2, 6, 12, 20, 30]
```

Copy code

Sum of range `[l, r]`:

```css
pref[r] - pref[l-1]
```

Copy code

C++ Code

```cpp
vector<long long> prefixSum(const vector<int>& a) {
    int n = a.size();
    vector<long long> pref(n);
    pref[0] = a[0];
    for(int i = 1; i < n; i++)
        pref[i] = pref[i - 1] + a[i];
    return pref;
}
```

Copy code

Python Code

```python
def prefix_sum(a):
    pref = [0]*len(a)
    pref[0] = a[0]
    for i in range(1, len(a)):
        pref[i] = pref[i-1] + a[i]
    return pref
```

Copy code

---

2. Prefix XOR

Meaning

`prefX[i] = A[0] ^ A[1] ^ ... ^ A[i]`

Why useful

- XOR subarray problems
- Bitwise parity logic
- Remove-one problems

Key Property

```powershell
XOR(l, r) = prefX[r] ^ prefX[l-1]
```

Copy code

C++ Code

```cpp
vector<int> prefixXor(const vector<int>& a) {
    int n = a.size();
    vector<int> pref(n);
    pref[0] = a[0];
    for(int i = 1; i < n; i++)
        pref[i] = pref[i - 1] ^ a[i];
    return pref;
}
```

Copy code

Python Code

```python
def prefix_xor(a):
    pref = [0]*len(a)
    pref[0] = a[0]
    for i in range(1, len(a)):
        pref[i] = pref[i-1] ^ a[i]
    return pref
```

Copy code

---

3. Prefix Max / Min

Meaning

```lua
```

```
prefMax[i] = max(A[0..i])
prefMin[i] = min(A[0..i])
```

Copy code

Example

ini

```ini
A = [2, 5, 1, 6]
prefMax = [2, 5, 5, 6]
prefMin = [2, 2, 1, 1]
```

Copy code

Use cases

- Stock problems
- Remove-one max/min
- Partition problems

C++ Code

cpp

```cpp
vector<int> prefixMax(const vector<int>& a) {
    vector<int> pref(a.size());
    pref[0] = a[0];
    for(int i = 1; i < a.size(); i++)
        pref[i] = max(pref[i-1], a[i]);
    return pref;
}
```

Copy code

Python Code

python

```python
def prefix_max(a):
    pref = [0]*len(a)
    pref[0] = a[0]
    for i in range(1, len(a)):
        pref[i] = max(pref[i-1], a[i])
    return pref
```

Copy code

---

4. Prefix Frequency

Meaning

Count occurrences till index `i`.

For characters:

```perl
freq[i][c] = count of char c in s[0..i]
```

Copy code

Example

```perl
s = "ababa"
freq of 'a' till index 3 = 2
```

Copy code

C++ Code

```cpp
vector<vector<int>> prefixFreq(const string& s) {
    int n = s.size();
    vector<vector<int>> freq(n, vector<int>(26, 0));
    freq[0][s[0]-'a']++;
    for(int i = 1; i < n; i++) {
        freq[i] = freq[i-1];
        freq[i][s[i]-'a']++;
    }
    return freq;
}
```

Copy code

Python Code

```python
def prefix_freq(s):
    n = len(s)
    freq = [[0]*26 for _ in range(n)]
    freq[0][ord(s[0])-97] += 1
    for i in range(1, n):
        freq[i] = freq[i-1][:]
```

Copy code

```
        freq[i][ord(s[i])-97] += 1
    return freq
```

---

5. Prefix Hash (Strings)

Meaning

Hash of prefix to compare substrings fast.

Formula

powershell

```
H[i] = (H[i-1] * base + s[i]) % mod
```

Copy code

Why needed

- Substring comparison
- Palindrome checking
- String matching

C++ Code

cpp

```
vector<long long> prefixHash(const string& s) {
    const long long base = 31, mod = 1e9+7;
    vector<long long> h(s.size());
    h[0] = s[0] - 'a' + 1;
    for(int i = 1; i < s.size(); i++)
        h[i] = (h[i-1] * base + (s[i]-'a'+1)) % mod;
    return h;
}
```

Copy code

Python Code

python

```
def prefix_hash(s):
    base, mod = 31, 10**9+7
    h = [0]*len(s)
    h[0] = ord(s[0]) - 96
    for i in range(1, len(s)):
```

```
        h[i] = (h[i-1]*base + ord(s[i])-96) % mod
    return h
```

Copy code

---

6. Prefix GCD

Meaning

matlab

```
prefG[i] = gcd(A[0..i])
```

Copy code

Use case

- Remove-one GCD problems
- Subarray GCD logic

C++ Code

cpp

```
vector<int> prefixGCD(const vector<int>& a) {
    vector<int> pref(a.size());
    pref[0] = a[0];
    for(int i = 1; i < a.size(); i++)
        pref[i] = gcd(pref[i-1], a[i]);
    return pref;
}
```

Copy code

Python Code

python

```
import math
def prefix_gcd(a):
    pref = [0]*len(a)
    pref[0] = a[0]
    for i in range(1, len(a)):
        pref[i] = math.gcd(pref[i-1], a[i])
    return pref
```

Copy code

---

7. Prefix AND / OR

Meaning

```
prefAnd[i] = A[0] & ... & A[i]
prefOr[i]  = A[0] | ... | A[i]
```

Copy code

Use case

- Bitwise constraints
- Nice subarray problems

---

8. Prefix Common Count

Meaning

Count common elements between two arrays up to `i`.

Example

makefile

```
A = [1,2,3,4]
B = [2,1,4,3]
Prefix common at i=1 → {1,2} → 2
```

Copy code

C++ Code

cpp

```cpp
vector<int> prefixCommon(const vector<int>& A, const vector<int>& B) {
    unordered_set<int> sa, sb;
    vector<int> res(A.size());
    int cnt = 0;
    for(int i = 0; i < A.size(); i++) {
        sa.insert(A[i]);
        sb.insert(B[i]);
        if(sa.count(B[i])) cnt++;
        if(sb.count(A[i])) cnt++;
        res[i] = cnt;
    }
```

```
    return res;
}
```

Copy code

2️⃣ SUFFIX CONCEPTS

Suffix = same ideas but from **right to left**.

Example (Suffix Sum)

```
suf[i] = A[i] + A[i+1] + ... + A[n-1]
```

Copy code

C++ Template

cpp

```cpp
vector<long long> suffixSum(const vector<int>& a) {
    int n = a.size();
    vector<long long> suf(n);
    suf[n-1] = a[n-1];
    for(int i = n-2; i >= 0; i--)
        suf[i] = suf[i+1] + a[i];
    return suf;
}
```

Copy code

Python Template

ChatGPT 5.2

```python
def suffix_sum(a):
    n = len(a)
    suf = [0]*n
    suf[n-1] = a[n-1]
    for i in range(n-2, -1, -1):
        suf[i] = suf[i+1] + a[i]
    return suf
```

Copy code

All other suffix concepts (Max, Min, GCD, Hash, AND/OR) follow **same logic** but reverse traversal.

3️⃣ REMOVE ONE ELEMENT PROBLEMS (MOST IMPORTANT)

Pattern

Remove `A[i]`, combine:

```css
prefix[i-1] + suffix[i+1]
```

Example: Remove one to maximize GCD

```cpp
ans = max(gcd(pref[i-1], suf[i+1]))
```

4️⃣ Difference Array / Imos Technique

Meaning

Efficient range updates.

Instead of updating each element:

```powershell
diff[l] += x
diff[r+1] -= x
```

Build actual array via prefix sum.

C++ Code

```cpp
void rangeUpdate(vector<int>& diff, int l, int r, int val) {
    diff[l] += val;
    if(r+1 < diff.size()) diff[r+1] -= val;
}
```

5️⃣ Sliding Window = Variable Prefix

Prefix but with **moving left pointer**.

Used when:

- Window size varies
- Constraints like sum ≤ k

6️⃣ Two Pointers = Prefix + Suffix Movement

Left builds prefix
Right shrinks suffix

Classic problems:

- Sorted arrays
- Palindromes
- Pair sum

7️⃣ Meet in the Middle

Split array:

```css
Left half → prefix
Right half → suffix
```

Used when:

- n ≈ 40
- brute force impossible

FINAL MENTAL MODEL 🧠

> **Prefix = precompute everything you might need from the left**
> **Suffix = same from the right**
> **Most array problems = clever combination of prefix & suffix**

If you want next:

- 🔥 20 **classic CP problems** using prefix/suffix
- 🔥 Visual dry runs
- 🔥 Interview patterns

Just tell me.