🤝 GCD (Greatest Common Divisor) / HCF (Highest Common Factor) in C++ 🧮

```cpp
#include <bits/stdc++.h>
using namespace std;
```

🧠 What Are We Solving?

We're finding the **largest number that divides two or more numbers** without leaving a remainder. This number is called the **GCD or HCF** — both mean the same.

Think of it as the **biggest brick size** you can use to build both walls of different lengths without cutting.

🏋️ [1] Naive Method — "Manual Laborer" 🛠️

💭 Analogy:

Imagine testing every brick size starting from `1` up to `min(a, b)` to see if it fits both walls perfectly. Slow but works!

```cpp
int gcdNaive(int a, int b) {
    int gcd = 1;
    for (int i = 1; i <= min(a, b); i++) {
        if (a % i == 0 && b % i == 0) {
            gcd = i;
        }
    }
    return gcd;
}
```

🧮 **Time:** `O(min(a, b))` 📦 **Space:** `O(1)` 🐌 **Downside:** Too slow for big numbers.

🔄 [2] Euclidean Algorithm — "Smart Divider" 📐

💭 Analogy:

If two lengths differ, subtract the smaller from the larger and repeat. Eventually, you'll reach the GCD. Efficient and clever!

```cpp
int gcdEuclidean(int a, int b) {
    while (b != 0) {
```

```cpp
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

🧮 **Time:** `O(log(min(a, b)))` 📦 **Space:** `O(1)` 🚀 **Much faster and widely used**!

🔁 [3] Recursive Euclidean — "Recursive Magician" 🧙

💭 Analogy:

Similar to the smart divider, but calls itself — like a **magic mirror** that reflects until it reaches the base.

```cpp
int gcdRecursive(int a, int b) {
    return b == 0 ? a : gcdRecursive(b, a % b);
}
```

🧮 **Time:** `O(log(min(a, b)))` 📦 **Space:** `O(stack)` due to recursion ✨ **Elegant and compact**

✖️ [4] GCD of Array — "Group Harmonizer" 🎼

💭 Analogy:

You want to find a **common beat** that all drums in a band can follow — you GCD the entire group!

```cpp
int gcdArray(vector<int>& nums) {
    int result = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        result = gcdEuclidean(result, nums[i]);
    }
    return result;
}
```

🧮 **Time:** `O(N log A)` where A is the largest element 📦 **Space:** `O(1)` 🎶 **Perfect for multiple numbers**

🏃 [5] Master Runner — Testing All Approaches 🧪

```cpp
void runAllGCDMethods(int a, int b) {
    cout << " 🔸 Naive GCD            : " << gcdNaive(a, b) << " 
(O(min(a,b)))\n";
    cout << " 🔸 Euclidean GCD        : " << gcdEuclidean(a, b) << "  (O(log
min(a,b)))\n";
```

```
    cout << " 🔸 Recursive Euclidean  : " << gcdRecursive(a, b) << "  (O(log
min(a,b)))\n";
}

void runGCDArrayDemo() {
    vector<int> nums = {24, 36, 48, 60};
    cout << "\n🎼 GCD of Array: ";
    for (int num : nums) cout << num << " ";
    cout << "\n 🔸 GCD of Array: " << gcdArray(nums) << endl;
}
```

🔚 [6] Main Function — Time to Run the Show 🎬

```
int main() {
    int a = 48, b = 18;

    runAllGCDMethods(a, b);
    runGCDArrayDemo();

    return 0;
}
```

📓 Summary Table

💡 **Method**	🕒 **Time Complexity**	💾 **Space**	⚙️ **Use Case**
Naive Method	O(min(a, b))	O(1)	Small values, understanding basics
Euclidean Algorithm	O(log min(a, b))	O(1)	Fast and standard for all inputs
Recursive Euclidean	O(log min(a, b))	O(stack)	Shorter, readable, recursive flavor
GCD of Array	O(N log A)	O(1)	Finding GCD in large data sets

✸ Bonus Tips

- 💯 Use **Euclidean GCD** in real-life programming or coding contests.
- 🔁 `__gcd(a, b)` is a built-in function in C++ (but **understand how it works**).
- 🤝 GCD helps in simplifying fractions, computing LCM, and many number theory problems.
- ✨ You can find **LCM using GCD** with this formula: `LCM(a, b) = (a * b) / GCD(a, b)`

🔁 LCM (Least Common Multiple) in C++ 💡

```cpp
#include <bits/stdc++.h>
using namespace std;
```

🧠 What Are We Solving?

We want the **smallest number that is a multiple of two (or more) numbers** — this is called **LCM**.

Think of LCM as the **first time two traffic lights turn green at the same time again!** 🚦

🧱 [1] Naive Method — "Try All Multiples" 🪜

💭 Analogy:

Imagine counting up from the bigger number until you find a number **both can divide evenly**. It's like looking for the **first common meeting day** on two calendars 📅.

```cpp
int lcmNaive(int a, int b) {
    int maxVal = max(a, b);
    while (true) {
        if (maxVal % a == 0 && maxVal % b == 0)
            return maxVal;
        maxVal++;
    }
}
```

🧮 **Time:** Can be up to `O(a*b)` 🐌 **Inefficient** for large numbers 📦 **Space:** `O(1)`

🧠 [2] LCM via GCD — "Mathemagician's Trick" ✨

💭 Analogy:

Use the formula:

> **LCM(a, b) = (a × b) / GCD(a, b)**

Why? Because LCM and GCD are deeply linked by this beautiful identity! 💍

```cpp
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

int lcmUsingGCD(int a, int b) {
    return (a / gcd(a, b)) * b; // To avoid overflow
}
```

🧮 **Time:** `O(log(min(a, b)))` ⚡ **Super fast** and clean 📦 **Space:** `O(stack)` if recursive GCD

🧩 [3] LCM of an Array — "Team Schedule Syncer" 🧭

💭 Analogy:

You're scheduling a meeting with a bunch of friends who are only free every `x`, `y`, `z`... days. When can everyone meet again? That's the LCM of the group!

```
int lcmOfArray(vector<int>& nums) {
    int result = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        result = (result / gcd(result, nums[i])) * nums[i];
    }
    return result;
}
```

🧮 **Time:** `O(N log A)` 👫 Works for large groups 📦 **Space:** `O(1)`

🎯 [4] Main Tester Functions 🧪

```
void runAllLCMMethods(int a, int b) {
    cout << " ◇  Naive LCM            : " << lcmNaive(a, b) << "  (O(a*b))\n";
    cout << " ◇  LCM Using GCD        : " << lcmUsingGCD(a, b) << "  (O(log
min(a,b)))\n";
}

void runLCMArrayDemo() {
    vector<int> nums = {3, 4, 5, 6};
    cout << "\n🎯 LCM of Array: ";
    for (int num : nums) cout << num << " ";
    cout << "\n ◇  LCM of Array: " << lcmOfArray(nums) << endl;
}
```

🏁 [5] Main Function — Ready to Roll 🚙

```
int main() {
    int a = 12, b = 18;

    runAllLCMMethods(a, b);
    runLCMArrayDemo();

    return 0;
}
```

🧾 Summary Table

💡 **Method**	🕒 **Time Complexity**	💾 **Space**	⚙️ **Use Case**
Naive Method	O(a*b)	O(1)	Simple understanding, small values
Using GCD	O(log min(a, b))	O(1)	Standard method, fast & clean
LCM of Array	O(N log A)	O(1)	Find LCM for N elements

💡 Bonus Section

🔗 Formula to Remember:

```
LCM(a, b) = (a * b) / GCD(a, b)
```

🤝 Relationship Between GCD & LCM:

```
GCD(a, b) * LCM(a, b) = a * b
```

✅ Built-in in C++17+:

```cpp
#include <numeric>
std::lcm(a, b);
std::gcd(a, b);
```

🎓 Real-Life Use Cases

- 🕒 Finding synchronized intervals (like clocks or timers)
- 🔢 Simplifying fractions
- 🎧 Audio/video sampling rates
- 📅 Scheduling common events
