// nums.erase(nums.begin() + i) remove the element of current index and all further elements shift     left by 1 and size of the vector is decreses by one ...

// so some rules regarding erase

// vector<int> vec = {1, 2, 3, 4, 5, 6, 7};

// Erase elements from index 2 to 4 (i.e., 3rd to 5th elements)

// vec.erase(vec.begin() + 2, vec.begin() + 5); // it is actually vec.erase(index = 2, index = 4)

// vec.erase(inclusive length, exclusive length);

//so a.erase(a.begin()+m, a.begin()+n-1) does not give expected output

a.erase(a.begin()+m , a.begin()+n); // But it gives expected outut

// How to use set

unordered\_set<int> unionSet(a.begin(), a.end()); // create set and Insert elements of `a`

unionSet.insert(b.begin(), b.end()); // Insert elements of `b`

return unionSet.size();

//

sum = 10

a = 5

sum += a # Equivalent to sum = sum + a

print(sum) # Output: 15

//

sum = 10

a = 5

sum = +a # Equivalent to sum = a (because +a has no effect on positive numbers)

print(sum) # Output: 5

//

**Limitations of int in Different Languages:**

| **Language** | **int Size** | **Approximate Max Value** |
| --- | --- | --- |
| **C / C++** | 32-bit | ~2.1 × 10⁹ (2^31 - 1) |
| **C / C++** | 64-bit (long long) | ~9.2 × 10¹⁸ (2^63 - 1) |
| **Java** | int (32-bit) | ~2.1 × 10⁹ |
| **Java** | long (64-bit) | ~9.2 × 10¹⁸ |
| **Python** | No fixed limit | Can store 100+ digit numbers |

//

No, \_\_uint128\_t **cannot** store a 100-digit number in C++.

**Why Not?**

* \_\_uint128\_t is a **128-bit unsigned integer**.
* The maximum value it can store is **2¹²⁸ - 1**, which is approximately **3.4 × 10³⁸**.
* A **100-digit number** is around **10⁹⁹**, which is much larger than \_\_uint128\_t can hold.
* **Alternative Solutions in C++**
* If you need to work with **100-digit numbers**, you must use a **big integer library**, because built-in integer types won’t be sufficient.
* **1. Use GMP Library (mpz\_t)**
* The **GMP (GNU Multiple Precision) library** supports arbitrarily large numbers.

#include <gmp.h> #include <iostream>

**2. Use boost::multiprecision::cpp\_int (Boost Library)**

If you don’t want to use GMP, Boost provides a **header-only** big integer solution.

#include <boost/multiprecision/cpp\_int.hpp> #include <iostream>