DSA MASTERY

Session 1

CP Wing

Dhanya & Indra kumar

KICKSTART YOUR CP JOURNEY!

Platform	Best For	Highlights
LeetCode	Interviews & DSA practice	Topic-wise problems, contests, company tags
Codeforces	Competitive programming & contests	Regular rated contests, editorial support
CodeChef	Practice & long contests	Star-based rating system, beginner-friendly
AtCoder	Algorithmic contests	Clean problem statements, fast contests
HackerRank	Interview prep & basics	Beginner-friendly, certifications available

Sheets & Roadmaps:

- Striver's A2Z DSA Sheet
- Fraz DSA Sheet
- LeetCode 75 & Blind 75
- CP- 31

YouTube Channels:

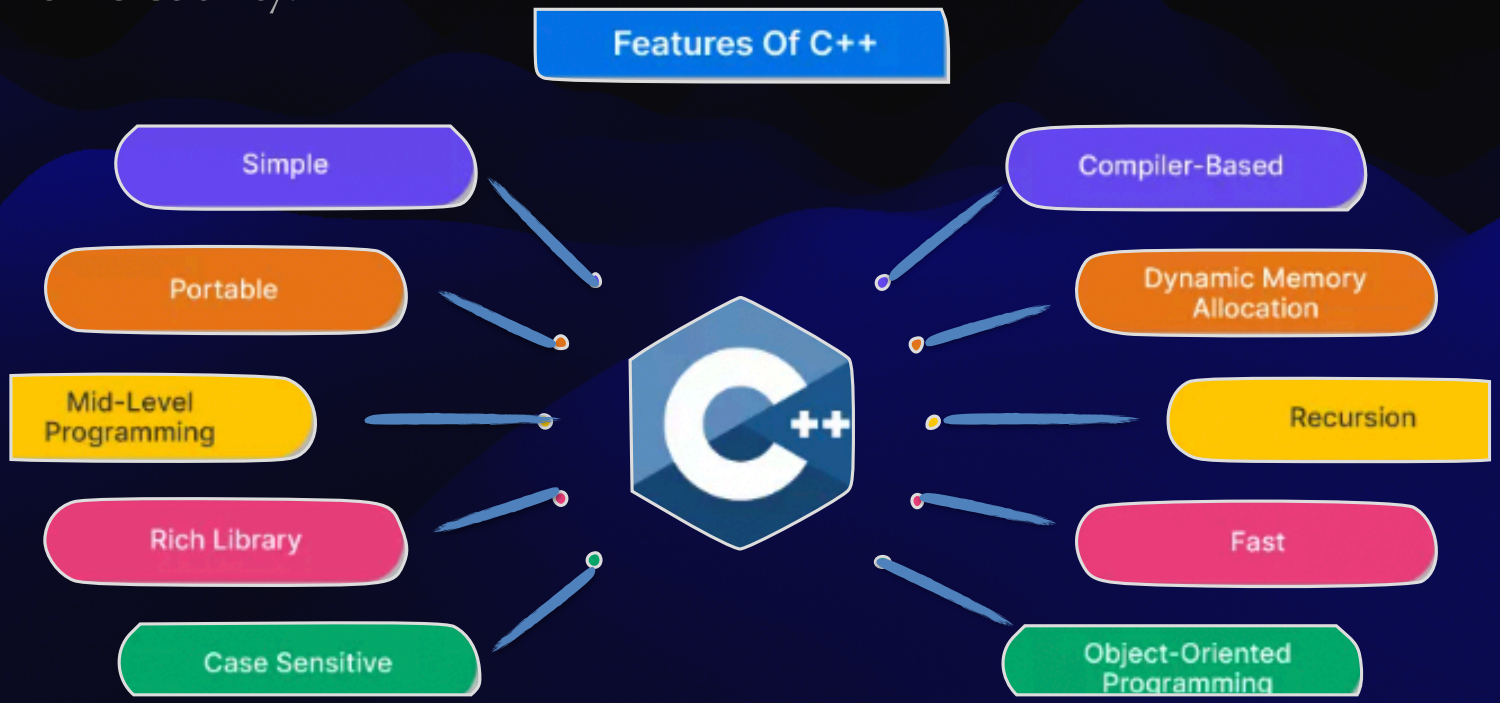
- NeetCode (DSA for interviews)
- take U forward (Striver's SDE Sheet)

CONTENTS

INTRODUCTION TO C++

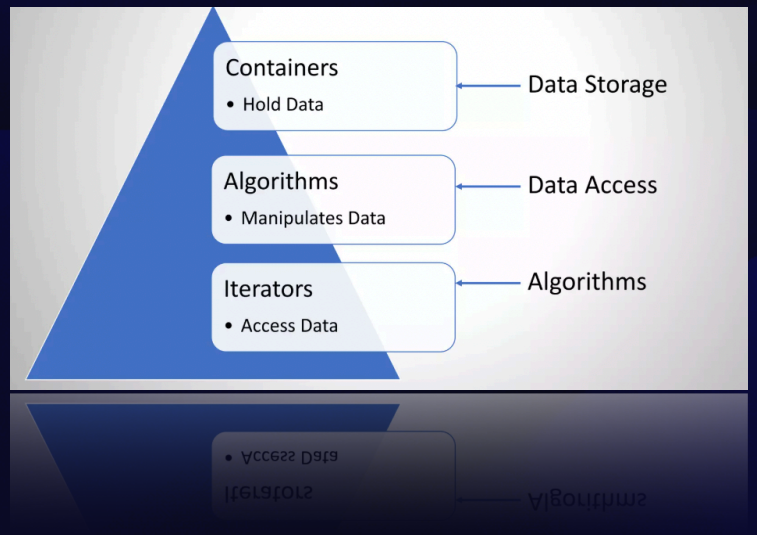
WHY C++?

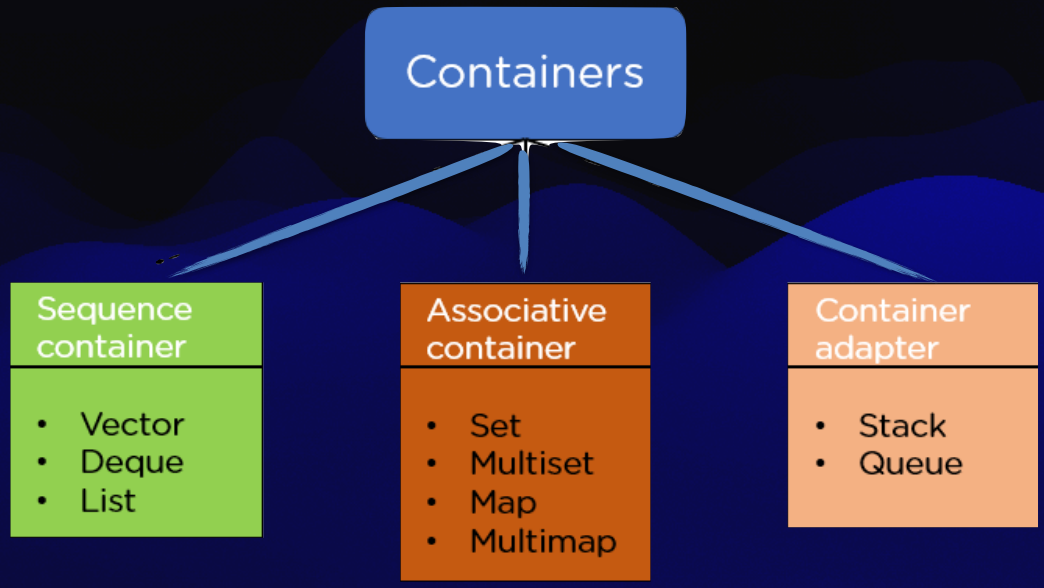
C++ remains a widely used language due to its powerful combination of performance, control, and versatility.



C++ Standard Template Library (STL)

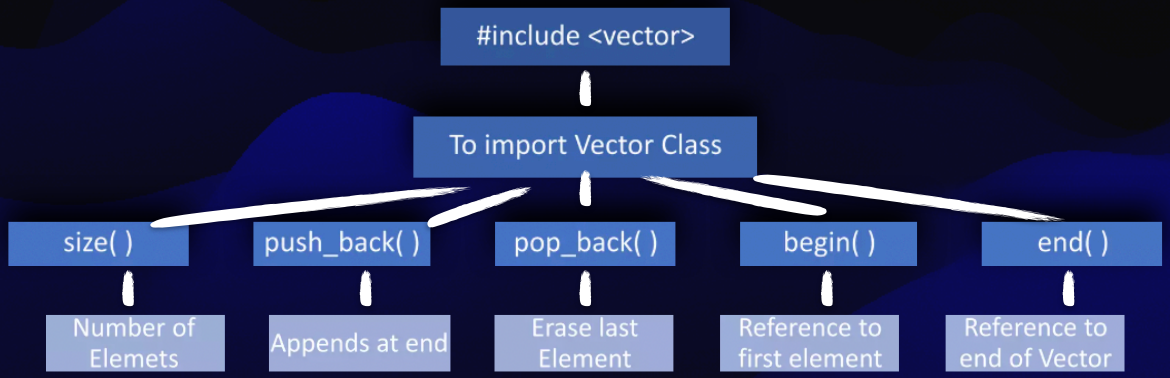
STL is a library that consist of different data structures and algorithms to effectively store and manipulate data.

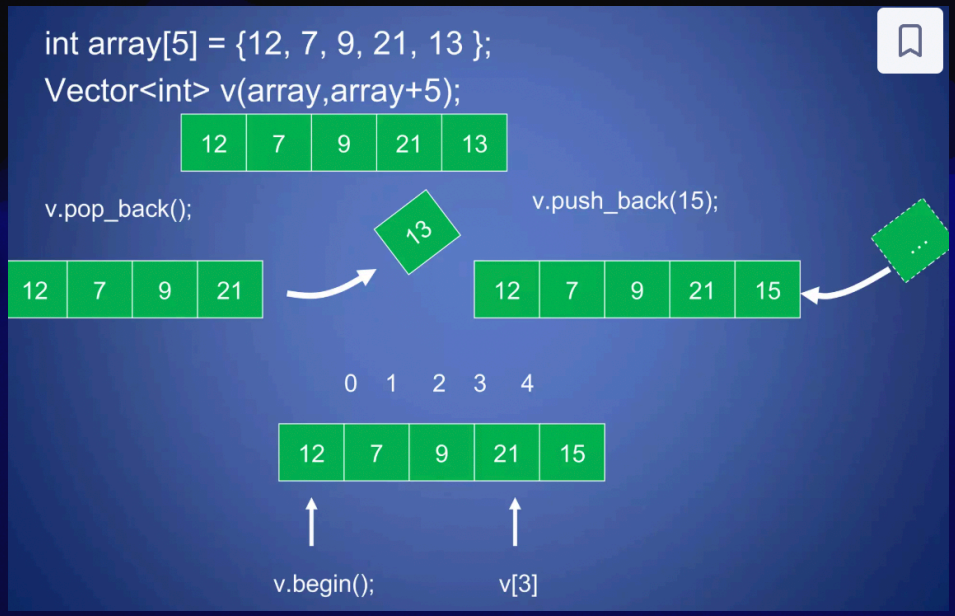




What is a Vector Class?

In C++, vectors are used to store elements of similar data types. However, unlike arrays, the size of a vector can grow dynamically.





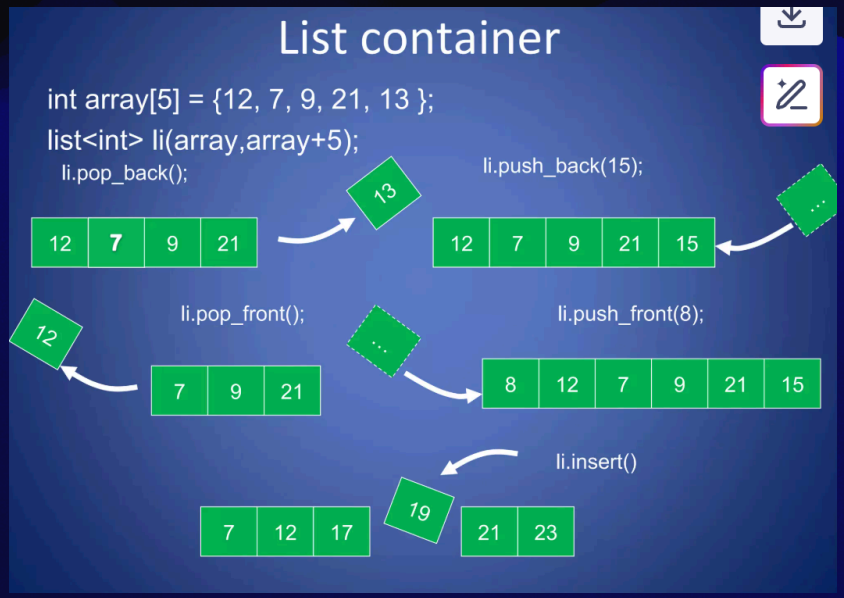
What is a List Class?

A list is similar to a vector in that it can store multiple elements of the same type and dynamically grow in size.

However, two major differences between lists and vectors are:

1. You can add and remove elements from both the beginning and at the end of a list, while vectors are generally optimized for adding and removing at the end.
2. Unlike vectors, a list does not support random access, meaning you cannot directly jump to a specific index, or access elements by index numbers.

Operations	Uses
sort	Sort elements in a list
splice	Transfer elements from 1 list to another list
merge	Merge sorted lists
unique	Removes duplicate elements
swap	Exchange the content of 1 list with that of another list.
assign	Assigns new contents to the list
remove	Erase all instances of value

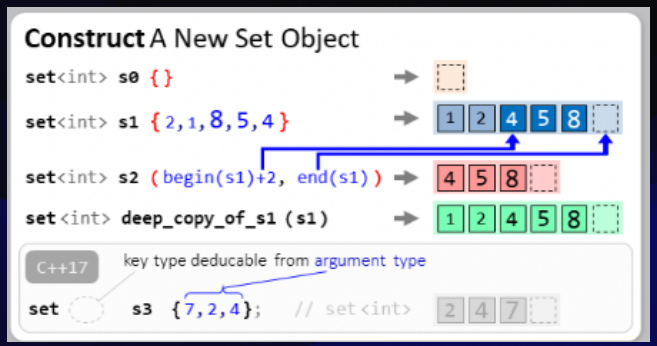


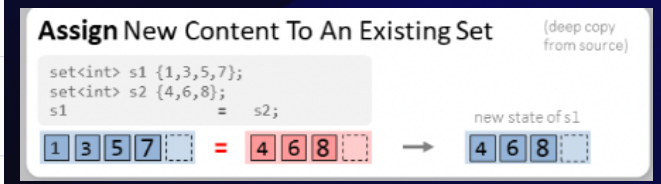
What is a Set Class?

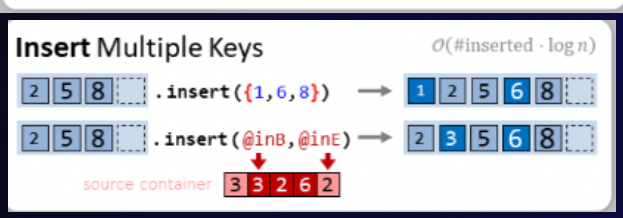
Set STL Function	Work	Time Complexity
begin()	Returns the iterator pointing to the first element of the set	O(1)
end()	Returns the pointer to the location which is next to the last element of the set.	O(1)
rbegin()	Returns the reverse iterator pointing to the last element	O(1)
rend()	Returns the reverse iterator pointing to the location before the first element	O(1)

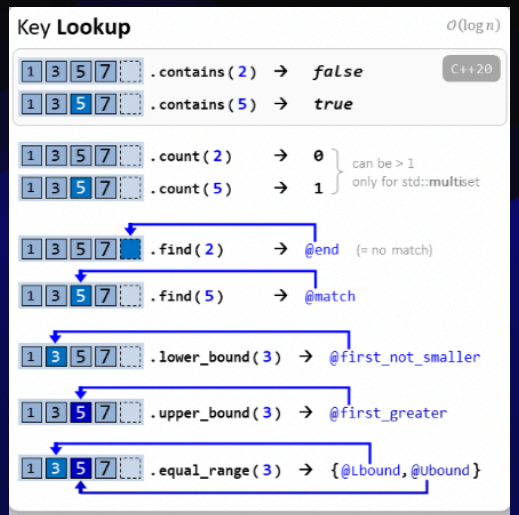
Set STL Function	Work	Time Complexity
size()	Returns the size/number of element of the set	O(1)
empty()	Checks if the set is empty or not	O(1)
max_size()	Returns the maximum allowed size/length of the set	O(1)

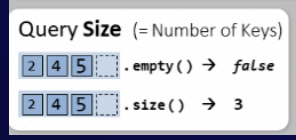
A set in c++ is an associative(STL) container used to store unique elements that are stored in a specific sorted order(increasing or decreasing).

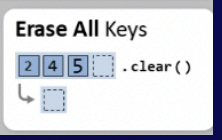


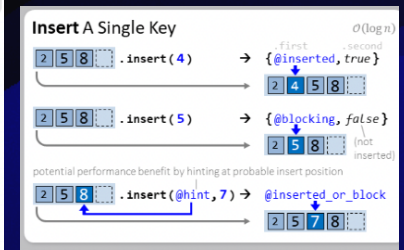


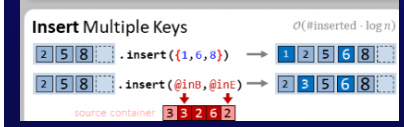








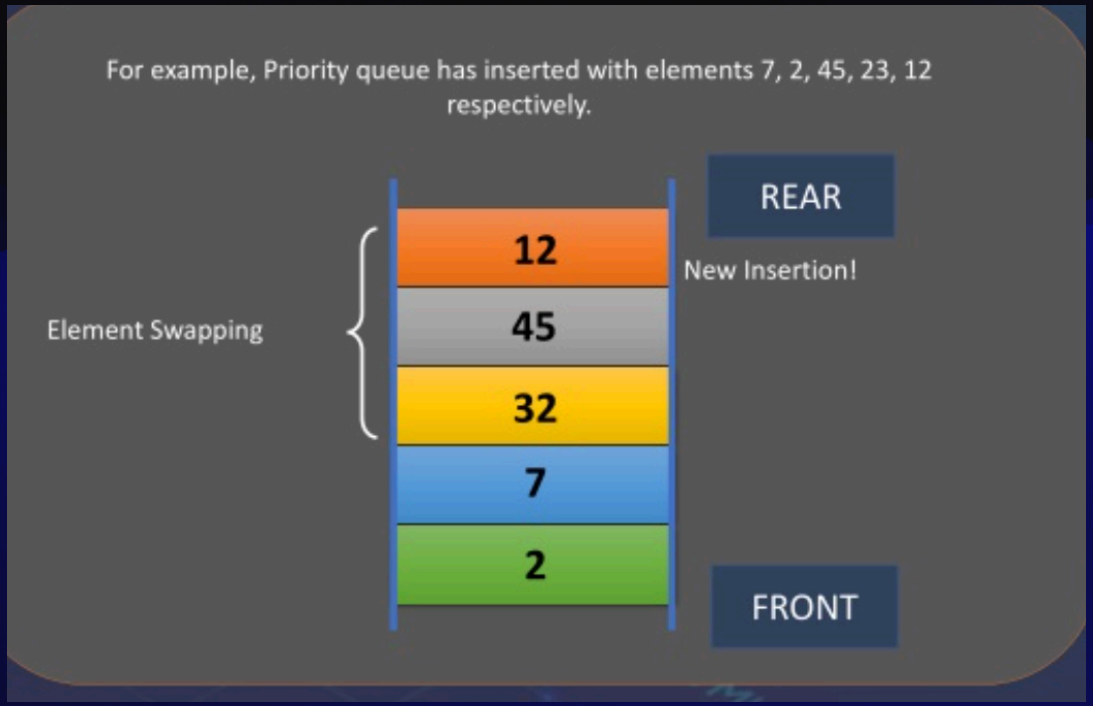




What is a Priority Queue?

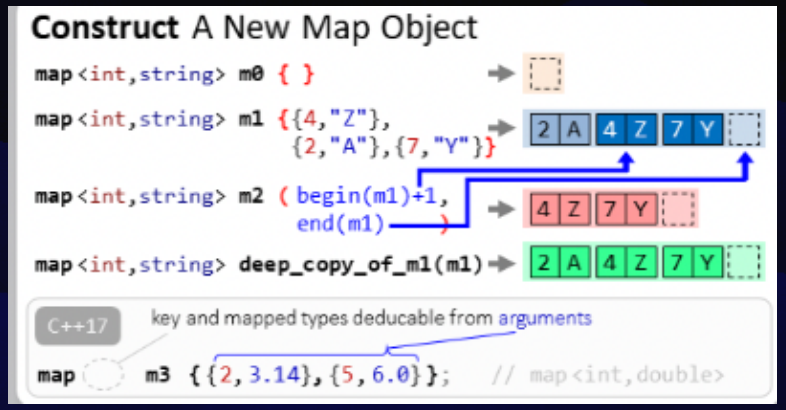
The priority queue is a container adaptor that provides constant time lookup of the largest (by default) element, at the expense of logarithmic insertion and extraction.

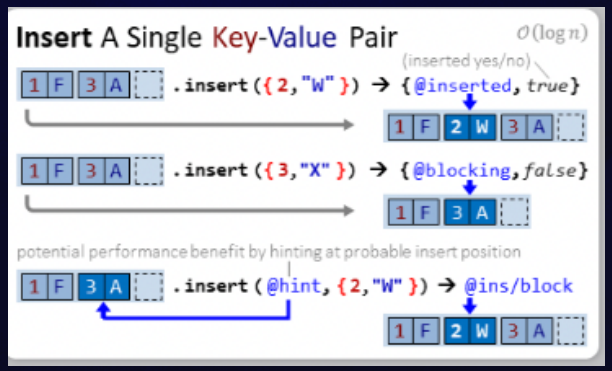
Methods	Description
`push()`	inserts the element into the priority queue
`pop()`	removes the element with the highest priority
`top()`	returns the element with the highest priority
`size()`	returns the number of elements
`empty()`	returns `true` if the priority_queue is empty

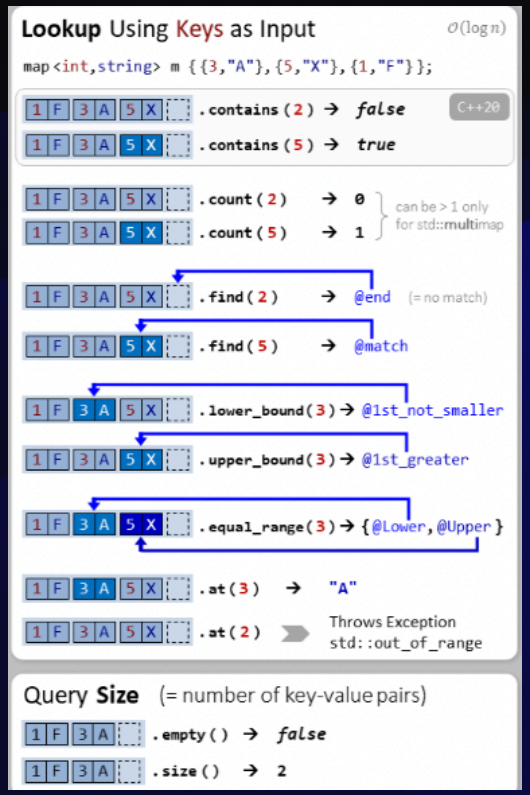


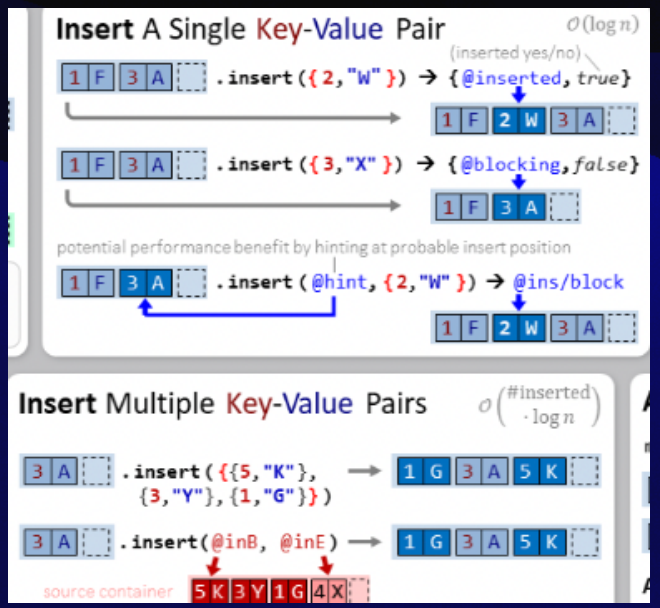
What is a Map?

In C++, maps are associative containers that store data in the form of key value pairs sorted on the basis of keys. No two mapped values can have the same keys. By default, it stores data in ascending order of the keys, but this can be changes as per requirement.







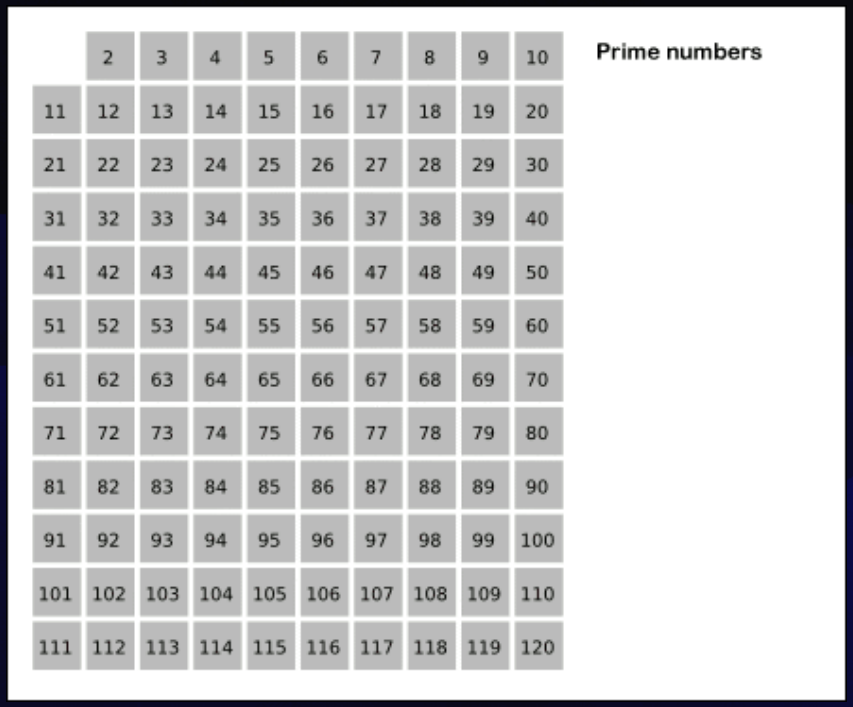


Time Complexities

Container	Insertion	Access	Erase	Find	Persistent Iterators
vector / string	Back: O(1) or O(n) Other: O(n)	O(1)	Back: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	No
deque	Back/Front: O(1) Other: O(n)	O(1)	Back/Front: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	Pointers only
list / forward_list	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	O(n)	Yes
set / map	O(log n)	-	O(log n)	O(log n)	Yes
unordered_set / unordered_map	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	Pointers only
priority_queue	O(log n)	O(1)	O(log n)	-	-

IMPORTANT MATH ALGORITHMS

Sieve of Eratosthenes - Introduction



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Sieve of Eratosthenes is an algorithm for finding all the prime numbers in a segment [1; n] using O(nlog log n) operations. The algorithm is very simple: at the beginning we write down all numbers between 2 and n. We mark all proper multiples of 2 (since 2 is the smallest prime number) as composite. A proper multiple of a number x, is a number greater than x and divisible by x. Then we find the next number that hasn't been marked as composite, in this case it is 3. Which means 3 is prime, and we mark all proper multiples of 3 as composite. The next unmarked number is 5, which is the next prime number, and we mark all proper multiples of it. And we continue this procedure until we have processed all numbers in the row.

Sieve of Eratosthenes - Implementation

```python
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

def primes_naive(limit):
    primes = []
    for num in range(2, limit + 1):
        if is_prime(num):
            primes.append(num)
    return primes
```

Naive Method to Check Primes

✅ Pros: Easy to understand. Good for checking if a single number is prime.

❌ Cons: Slow for large values of limit. Time complexity: O(n√n).

```cpp
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

Sieve of Eratosthenes

✅ Pros: Very fast for generating many primes. Time complexity: O(n log log n). Great for tasks like cryptography, number theory, or competitive programming.

❌ Cons: Uses a bit more memory (O(n) space). Not suitable for checking very large individual primes.

Sieve of Eratosthenes - Optimised

```
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

Obviously, to find all the prime numbers until n, it will be enough just to perform the sifting only by the prime numbers, which do not exceed the root of n.

Example: Sieve up to 30

You only need to loop p up to:

sqrt 30 a = 5.47 => loop p = 2 ,3,5

At p = 2 , mark 4, 6, 8, 10, ..., 30

At p = 3 , mark 9, 12, 15, 30

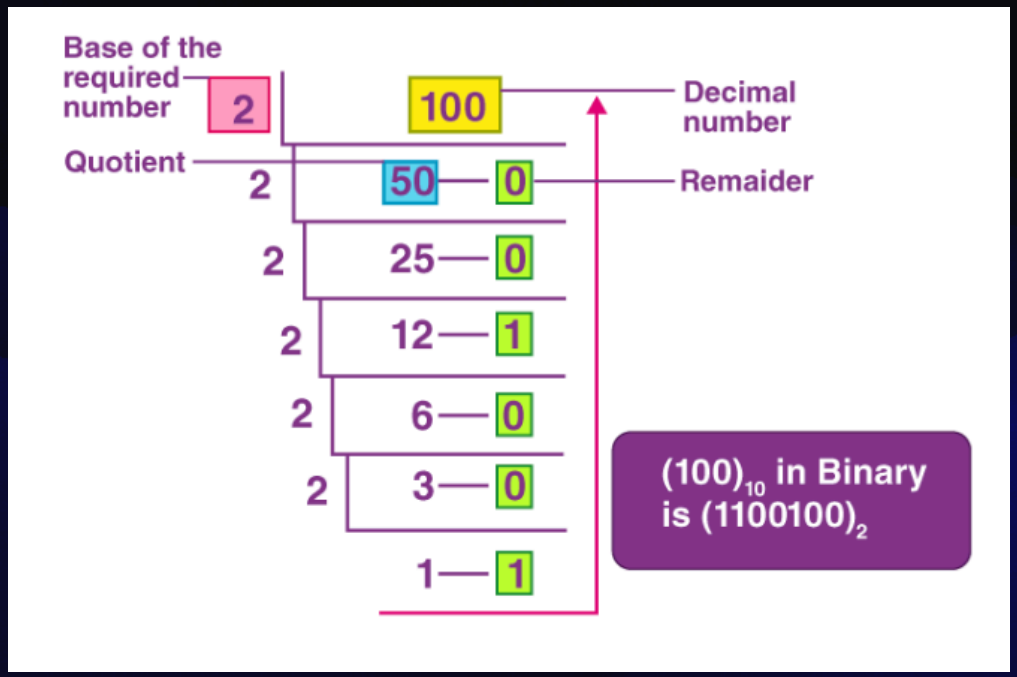
At p = 5 , mark 25, 30

No need to use p = 7 , p = 11 , etc., because all their multiples will either:

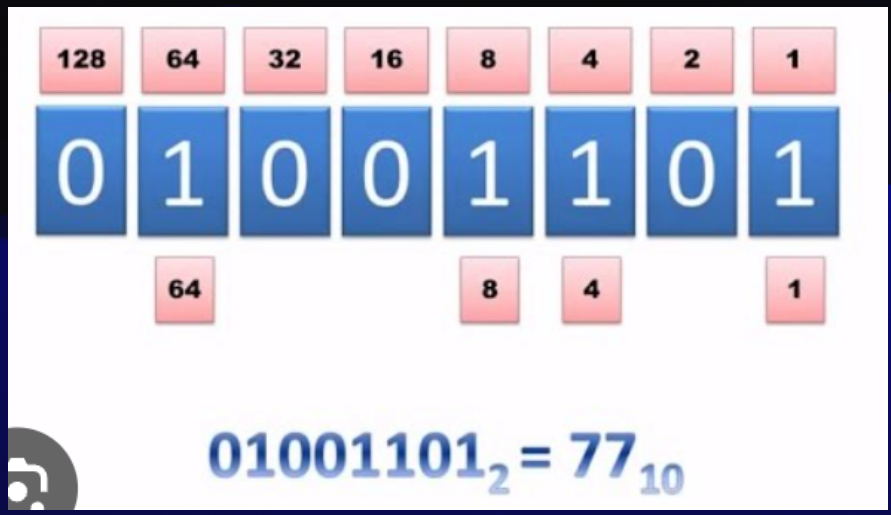
Be bigger than 30

Or already marked by smaller primes

Bit Manipulation - Basics



Decimal to Binary Conversion



Binary to Decimal Conversion

Bit Manipulation - Basics

Name	Symbol	Usage	What it does
Bitwise And	&	a&b	Returns 1 only if both the bits are 1
Bitwise Or	|	a|b	Returns 1 if one of the bits is 1
Bitwise Not	~	~a	Returns the complement of a bit
Bitwise Xor	^	a^b	Returns 0 if both the bits are same else 1
Bitwise Left shift	<<	a<<n	Shifts a towards left by n digits
Bitwise Right shift	>>	a>>n	Shifts a towards right by n digits

1. Bitwise And - The Bitwise and returns 1 only if both the bits are 1.

Number a = 25	0	0	0	1	1	0	0	1
Number b = 14	0	0	0	0	1	1	1	0
a & b	0	0	0	0	1	0	0	0

2. Bitwise Or - The Bitwise or returns 1 if either of the bits is 1.

Number a = 25	0	0	0	1	1	0	0	1
Number b = 14	0	0	0	0	1	1	1	0
a | b	0	0	0	1	1	1	1	1

4. Bitwise Xor - The Bitwise xor returns 1 only if one of the bits is zero.

Number a = 25	0	0	0	1	1	0	0	1
Number b = 14	0	0	0	0	1	1	1	0
a ^ b	0	0	0	1	0	1	1	1

5. Bitwise Left Shift

Number a = 25	0	0	0	1	1	0	0	1
a << 3	1	1	0	0	1	0	0	0

6. Bitwise Right shift

Number a = 25	0	0	0	1	1	0	0	1
a >> 3	0	0	0	0	0	0	1	1

Bit Manipulation - Swap two number using XOR

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 2, b = 3;
    cout << "a = " << a << " b = " << b << endl;

    a = a ^ b;
    b = a ^ b;
    a = a ^ b;

    cout << "a = " << a << " b = " << b << endl;
    return 0;
}
```

The idea is to use the properties of XOR to swap the two variables.

- **a = a ^ b:** Store the Bitwise XOR of **a** and **b** in **a**. Now, **a** holds the result of **(a ^ b)**.
- **b = a ^ b:** Bitwise XOR the new value of **a** with **b** to get the original value of **a**. This gives us, b = (a ^ b) ^ b = a.
- **a = a ^ b:** Bitwise XOR the new value of **a** with the new value of **b** (which is the original **a**) to get the original value of **b**. This gives us, a = (a ^ b) ^ a = b.

Finally, **a** and **b** hold the swapped values.

Bit Manipulation - Techniques

Technique	Code
Test k^{th} bit is set	`num & (1 << k) != 0`
Set k^{th} bit	`num |= (1 >> k)`
Turn off k^{th} bit	`num &= ~(1 << k)`
Toggle the k^{th} bit	`num ^= (1 << k)`
Multiply by 2^k	`num << k`
Divide by 2^k	`num >> k`
Check if a number is a power of 2	`(num & num - 1) == 0` or `(num & (-num)) == num`
Swapping two variables	`num1 ^= num2; num2 ^= num1; num1 ^= num2`


THANKS FOR LISTENING
ANY QUESTIONS?