Math (Basic Number Theory) GCD, LCM, modulo, primes, divisors c++ code and theory formulas

ChatGPT said:

Here’s a **compact and clear guide** to the **basic number theory** topics in **C++**, including:

1. GCD (Greatest Common Divisor)
2. LCM (Least Common Multiple)
3. Modulo Arithmetic
4. Prime Numbers (sieve and primality check)
5. Divisors of a Number

✓ 1. GCD (Greatest Common Divisor)

Formula:
If `a % b == r`, then `gcd(a, b) = gcd(b, r)`
Eventually: `gcd(a, b) = gcd(b, a % b)`

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

□ **Property**:

- `gcd(a, 0) = a`
- `gcd(a, b) = gcd(b, a % b)`
- `gcd(a, b)` **divides both `a` and `b`**

✓ 2. LCM (Least Common Multiple)

Formula:
`lcm(a, b) = (a * b) / gcd(a, b)`

```
cpp
CopyEdit
int lcm(int a, int b) {
    return (a / gcd(a, b)) * b;  // Avoid overflow
}
```

✓ 3. Modulo Arithmetic

Properties:

- `(a + b) % m = ((a % m) + (b % m)) % m`
- `(a * b) % m = ((a % m) * (b % m)) % m`
- `(a - b) % m = ((a % m) - (b % m) + m) % m`
- `(a^b) % m` **can be computed using modular exponentiation**

```
cpp
CopyEdit
// Modular Exponentiation (a^b % m)
long long mod_pow(long long a, long long b, long long m) {
    long long res = 1;
    a %= m;
    while (b > 0) {
        if (b & 1) res = (res * a) % m;
        a = (a * a) % m;
        b >>= 1;
    }
    return res;
}
```

✅ 4. Prime Numbers

A. Primality Test (Basic)

```
cpp
CopyEdit
bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i*i <= n; ++i)
        if (n % i == 0) return false;
    return true;
}
```

B. Sieve of Eratosthenes (All primes ≤ N)

```
cpp
CopyEdit
vector<bool> sieve(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;
        }
    }
    return is_prime;
}
```

✅ 5. Divisors of a Number

All divisors of `n` up to √n

```
cpp
CopyEdit
```

```
vector<int> getDivisors(int n) {
    vector<int> divs;
    for (int i = 1; i*i <= n; ++i) {
        if (n % i == 0) {
            divs.push_back(i);
            if (i != n / i)
                divs.push_back(n / i);
        }
    }
    sort(divs.begin(), divs.end());
    return divs;
}
```

□ Summary of Formulas:

Concept	Formula
GCD	`gcd(a, b) = gcd(b, a % b)`
LCM	`lcm(a, b) = (a * b) / gcd(a, b)`
Modulo Sum	`(a + b) % m = ((a % m) + (b % m)) % m`
Modulo Multiply	`(a * b) % m = ((a % m) * (b % m)) % m`
Modulo Subtract	`(a - b) % m = ((a % m) - (b % m) + m) % m`
Modulo Power	`a^b % m` via binary exponentiation
Divisors	Up to √n, both `i` and `n/i` if `n % i == 0`

Would you like a **combined C++ demo** of all these in one main program?

You said:

Greedy Algorithms Simple optimal choices (e.g., sorting + decision making

ChatGPT said:

Here’s a **concise guide** to **Greedy Algorithms with simple optimal choices**, especially problems that use **sorting + decision making**. These are classic problems where **local optimal decisions lead to a globally optimal solution**.

✅ What Are Greedy Algorithms?

A **greedy algorithm** builds up a solution piece by piece, always choosing the **most optimal option at the current step** (local optimum), hoping it leads to the **global optimum**.

□ Core Pattern: `Sorting + Decision Making`

This pattern is common in greedy:

1. **Sort the input** (based on some logic)
2. **Make the best choice** at every step
3. Continue until goal is met

🔥 Popular Greedy Problems in C++

1. Activity Selection Problem

Given start and end times of activities, select the **maximum number of non-overlapping activities**.

```
cpp
CopyEdit
bool cmp(pair<int, int> a, pair<int, int> b) {
    return a.second < b.second; // sort by end time
}

int maxActivities(vector<pair<int, int>> &activities) {
    sort(activities.begin(), activities.end(), cmp);
    int count = 1, end = activities[0].second;

    for (int i = 1; i < activities.size(); ++i) {
        if (activities[i].first >= end) {
            count++;
            end = activities[i].second;
        }
    }
    return count;
}
```

---

2. Fractional Knapsack

Choose items to **maximize value** within a given weight capacity, but you can take **fractional parts** of items.

```
cpp
CopyEdit
struct Item {
    double value, weight;
};

bool cmp(Item a, Item b) {
    return (a.value / a.weight) > (b.value / b.weight); // value density
}

double fractionalKnapsack(vector<Item> items, double W) {
    sort(items.begin(), items.end(), cmp);
    double total = 0.0;

    for (auto &item : items) {
        if (W >= item.weight) {
            total += item.value;
            W -= item.weight;
        } else {
            total += item.value * (W / item.weight);
            break;
        }
    }
    return total;
}
```

---

3. Minimum Number of Coins

You have coin denominations; find the **minimum number of coins** to make a given amount.

```
cpp
CopyEdit
int minCoins(vector<int> &coins, int amount) {
    sort(coins.rbegin(), coins.rend()); // sort descending
    int count = 0;
    for (int coin : coins) {
        while (amount >= coin) {
            amount -= coin;
            count++;
        }
    }
    return (amount == 0 ? count : -1); // -1 if not possible
}
```

---

4. Job Scheduling (With Deadlines)

Each job has a deadline and profit. Schedule jobs to **maximize total profit** while respecting deadlines (1 job per time slot).

```
cpp
CopyEdit
struct Job {
    int id, deadline, profit;
};

bool cmp(Job a, Job b) {
    return a.profit > b.profit; // greedy: most profit first
}

int jobScheduling(vector<Job> &jobs) {
    sort(jobs.begin(), jobs.end(), cmp);
    int maxTime = 0;
    for (auto j : jobs) maxTime = max(maxTime, j.deadline);

    vector<bool> slot(maxTime + 1, false);
    int totalProfit = 0;

    for (auto job : jobs) {
        for (int t = job.deadline; t >= 1; --t) {
            if (!slot[t]) {
                slot[t] = true;
                totalProfit += job.profit;
                break;
            }
        }
    }
    return totalProfit;
}
```

✓ Summary of Greedy + Sorting Strategy

Problem	Sort By	Decision Rule
Activity Selection	End time ascending	Choose next activity if `start ≥ end`
Fractional Knapsack	Value/weight descending	Take as much as fits
Coin Change (Min Coins)	Coin value descending	Use as many large coins as possible
Job Scheduling (Max Profit)	Profit descending	Schedule latest available slot ≤ deadline

---

Do you want a combined code that includes **menu + test cases** for all the above in one file?

You said:

Brute Force Try all combinations if constraints are small (n ≤ 1000).

ChatGPT said:

Here's a **clear and practical guide** to the **Brute Force** technique, focusing on **trying all combinations** when `n ≤ 1000`. Brute force is often your best bet when:

- Constraints are small (like `n ≤ 20` for subsets or `n ≤ 1000` for pairwise/comparisons).
- No clever optimization is obvious.
- You want to **verify correctness** before optimizing.

✓ What Is Brute Force?

Brute Force = Try **all possible combinations / configurations** and pick the best one.

🔁 Examples of brute force:

- Try every pair ($O(n^2)$)
- Try every triplet ($O(n^3)$)
- Try every subset (2^n) — only for `n ≤ 20`
- Try every permutation (n!) — only for `n ≤ 9~10`

✓ Classic Brute Force Problems with `n ≤ 1000`

---

1. Two Sum (All Pairs)

```
cpp
CopyEdit
bool hasTwoSum(vector<int> &arr, int target) {
    int n = arr.size();
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (arr[i] + arr[j] == target)
                return true;
    return false;
}
```

Time: $O(n^2)$, Safe for `n ≤ 1000`

---

2. Count All Triplets With Sum Zero

```
cpp
CopyEdit
int countTriplets(vector<int> &arr) {
    int n = arr.size(), count = 0;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            for (int k = j+1; k < n; ++k)
                if (arr[i] + arr[j] + arr[k] == 0)
                    count++;
    return count;
}
```

Time: $O(n^3)$, use only if `n ≤ 200~300`

---

3. Find All Pairs (i, j) Where `a[i] < a[j]`

```
cpp
CopyEdit
vector<pair<int, int>> allIncreasingPairs(vector<int> &a) {
    vector<pair<int, int>> res;
    int n = a.size();
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (a[i] < a[j])
                res.push_back({i, j});
    return res;
}
```

---

4. Subset Sum ($n \le 20$)

Try all subsets to check if a sum can be formed.

```
cpp
CopyEdit
bool subsetSum(vector<int> &a, int target) {
    int n = a.size();
    for (int mask = 0; mask < (1 << n); ++mask) {
        int sum = 0;
        for (int i = 0; i < n; ++i)
            if (mask & (1 << i))
                sum += a[i];
        if (sum == target)
            return true;
    }
    return false;
}
```

Time: $O(2^n)$, feasible for `n ≤ 20`

---

5. Permutations ($n \le 8$) using `next_permutation`

```
cpp
CopyEdit
void allPermutations(vector<int> a) {
    sort(a.begin(), a.end());
    do {
        for (int x : a) cout << x << " ";
        cout << "\n";
    } while (next_permutation(a.begin(), a.end()));
}
```

□ Tips for Brute Force in Contests

Constraint	Safe Algorithm	Time Complexity
`n ≤ 100`	$O(n^3)$	Triplet loops
`n ≤ 1000`	$O(n^2)$	Pairwise checks
`n ≤ 20`	$O(2^n)$	Subset masks
`n ≤ 8~10`	$O(n!)$	Permutations

🚀 Summary

Brute force is:

- Easy to write
- Great for small inputs
- Useful for validating optimized approaches

Would you like a **template-driven menu-based C++ app** that implements all these brute force examples with user input and results?

You said:

Two Pointers For string and array subrange problems

ChatGPT said:

Here’s a **complete guide to the Two Pointers technique**, especially for solving **string and array subrange problems** — an essential trick for optimizing from **O(n²)** to **O(n)** or **O(n log n)** in many cases.

✓ What is the Two Pointers Technique?

Use **two indices** (`i` and `j`) to represent a **window or subrange** and **slide them efficiently** to solve a problem.

- Common when the array/string is **sorted or monotonic**
- Often used to maintain **a sliding window**
- Replaces nested loops

☐ Where to Use Two Pointers

- **Subarray with given sum**
- **Longest substring/sequence**
- **Remove duplicates**
- **Palindrome check**
- **Merging two sorted arrays**

🔥 Classic Two Pointer Problems

---

1. Find if Any Two Numbers Sum to Target (Sorted Array)

```
cpp
CopyEdit
bool hasTwoSum(vector<int> &arr, int target) {
    int left = 0, right = arr.size() - 1;
    while (left < right) {
        int sum = arr[left] + arr[right];
        if (sum == target) return true;
        else if (sum < target) left++;
        else right--;
    }
    return false;
}
```

- Time: O(n) — single pass
- Condition: array must be **sorted**

---

2. Longest Substring Without Repeating Characters

```
cpp
CopyEdit
int longestUniqueSubstring(string s) {
    unordered_set<char> seen;
    int i = 0, j = 0, n = s.length(), maxLen = 0;

    while (j < n) {
        if (seen.count(s[j]) == 0) {
            seen.insert(s[j++]);
            maxLen = max(maxLen, j - i);
        } else {
            seen.erase(s[i++]);
        }
    }
    return maxLen;
}
```

- Time: O(n)
- Good for strings with only lowercase/uppercase letters

---

3. Count Subarrays with Sum ≤ K (Positive Integers)

```
cpp
CopyEdit
int countSubarrays(vector<int> &a, int k) {
    int i = 0, sum = 0, res = 0;
    for (int j = 0; j < a.size(); ++j) {
        sum += a[j];
        while (sum > k)
            sum -= a[i++];
        res += (j - i + 1);
    }
    return res;
}
```

- Time: O(n)
- Returns total number of valid subarrays

---

4. Remove Duplicates from Sorted Array (in-place)

cpp
CopyEdit

```
int removeDuplicates(vector<int> &nums) {
    int i = 0;
    for (int j = 1; j < nums.size(); ++j) {
        if (nums[j] != nums[i])
            nums[++i] = nums[j];
    }
    return i + 1; // size of unique part
}
```

---

5. Check if String is Palindrome Using Two Pointers

cpp
CopyEdit

```
bool isPalindrome(string &s) {
    int i = 0, j = s.length() - 1;
    while (i < j) {
        if (s[i] != s[j]) return false;
        i++, j--;
    }
    return true;
}
```

📌 Two Pointers Template (Sliding Window)

cpp
CopyEdit

```
int i = 0, j = 0;
while (j < n) {
    // Expand window
    // Process s[j]

    while (/* window invalid */) {
        // Shrink from the left
        // Update i
    }

    // Update answer
    j++;
}
```

🔁 Summary Table

Problem	Pattern Used	Time
Two Sum (sorted)	`i = 0, j = n-1`	O(n)
Longest Unique Substring	Sliding Window + Set	O(n)
Count Subarrays with Sum ≤ K	Sliding Window + Shrinking	O(n)
Remove Duplicates (sorted array)	Slow + fast pointer	O(n)
Check Palindrome	Two ends moving inward	O(n)

Would you like a **C++ file with all these problems coded in one menu-driven demo**?

You said:

Prefix Sum / Difference Arrays For range sum or difference queries

ChatGPT said:

Here's a detailed yet beginner-friendly guide to **Prefix Sum and Difference Arrays** — two powerful techniques for **range sum and range update problems** in **arrays** and **matrices**.

✅ 1. Prefix Sum Array

◆ Use Case:

- **Fast range sum queries** in **O(1)** time after **O(n)** preprocessing.
- Convert O(n) range queries to O(1).

◆ How It Works:

Let `arr[]` be your input array.
Create a `prefix[]` array such that:

```
cpp
CopyEdit
prefix[i] = arr[0] + arr[1] + ... + arr[i];
```

Then, the sum of range `[l, r]` is:

```
cpp
CopyEdit
sum = prefix[r] - prefix[l-1]; // if l > 0
sum = prefix[r];               // if l == 0
```

✅ C++ Code Example:

```
cpp
CopyEdit
vector<int> buildPrefixSum(const vector<int>& arr) {
    int n = arr.size();
    vector<int> prefix(n);
    prefix[0] = arr[0];
    for (int i = 1; i < n; ++i)
        prefix[i] = prefix[i - 1] + arr[i];
    return prefix;
}

int rangeSum(const vector<int>& prefix, int l, int r) {
    if (l == 0) return prefix[r];
    return prefix[r] - prefix[l - 1];
}
```

✓ 2. Difference Array

◆ Use Case:

- Efficiently apply **range updates** in O(1) time.
- Modify ranges without touching every element.

◆ Idea:

Given original array `arr[]`, build `diff[]`:

```
cpp
CopyEdit
diff[0] = arr[0];
diff[i] = arr[i] - arr[i-1]; // for i > 0
```

To add `x` to range [l, r], do:

```
cpp
CopyEdit
diff[l] += x;
diff[r + 1] -= x;
```

After all updates, build final array:

```
cpp
CopyEdit
arr[0] = diff[0];
for i = 1 to n-1:
    arr[i] = arr[i-1] + diff[i];
```

✓ C++ Code Example:

```
cpp
CopyEdit
vector<int> applyRangeUpdates(int n, vector<tuple<int, int, int>> updates)
{
    vector<int> diff(n + 2, 0); // difference array
    for (auto [l, r, val] : updates) {
        diff[l] += val;
        diff[r + 1] -= val;
    }

    vector<int> result(n);
    result[0] = diff[0];
    for (int i = 1; i < n; ++i)
        result[i] = result[i - 1] + diff[i];

    return result;
}
```

✅ 3. 2D Prefix Sum (For Matrices)

◆ Use Case:

- Answer submatrix sum queries in O(1)

Let `matrix[i][j]` be the original matrix
Build `prefix[i][j]` as:

```
cpp
CopyEdit
prefix[i][j] = matrix[i][j]
               + prefix[i-1][j]
               + prefix[i][j-1]
               - prefix[i-1][j-1];
```

Then, to query sum in rectangle `(r1,c1)` to `(r2,c2)`:

```
cpp
CopyEdit
sum = prefix[r2][c2]
      - prefix[r1-1][c2]
      - prefix[r2][c1-1]
      + prefix[r1-1][c1-1];
```

☐ Summary Table

Problem Type	Technique	Time
Range sum in array	Prefix Sum	O(1)
Range update in array	Difference Array	O(1) per update
2D matrix sum queries	2D Prefix Sum	O(1)

📌 Example Problem Scenarios:

Task	Use
Query sum of salaries from employee 10 to 100	Prefix Sum

Increase scores of students 50 to 70 by 10	Difference Array
Count sum of values in a rectangular grid	2D Prefix Sum

Would you like a **C++ implementation of all these with interactive input**?

You said:

Arrays and Strings Core data types used in almost every problem.

ChatGPT said:

Absolutely! **Arrays and Strings** are the **core data types** in competitive programming and interviews — they appear in **almost every problem**, either directly or indirectly.

Below is a detailed **cheat sheet** covering:

1. Array Basics & Patterns
2. String Basics & Operations
3. Common Interview-Level Problems
4. Key STL Functions in C++

✓ 1. Arrays – Core Concepts

📌 Declaration

```
cpp
CopyEdit
int a[100];                   // Static array
vector<int> v;               // Dynamic array
vector<int> v(n, 0);         // Size n, all elements 0
```

📌 Input/Output

```
cpp
CopyEdit
vector<int> v(n);
for (int i = 0; i < n; ++i) cin >> v[i];
for (int x : v) cout << x << " ";
```

🔥 Common Array Patterns

Pattern	Description	Example Problem
Prefix Sum	Precompute sum for range queries	Range Sum Query
Two Pointers	Solve subrange efficiently	Subarray with given sum

Sliding Window	Fixed/variable window problems	Max sum subarray of size k
Hashing/Frequency Count	Store counts in map/vector	Majority Element, Anagram check
Sorting + Greedy	Min/max combinations	Min Arrows to Burst Balloons
Binary Search	Use on sorted arrays	Search Insert Position

✅ Example Problems (Arrays)

a. Maximum Subarray (Kadane's Algorithm)

```
cpp
CopyEdit
int maxSubArray(vector<int>& nums) {
    int maxSum = nums[0], curr = nums[0];
    for (int i = 1; i < nums.size(); ++i) {
        curr = max(nums[i], curr + nums[i]);
        maxSum = max(maxSum, curr);
    }
    return maxSum;
}
```

b. Move Zeros to End

```
cpp
CopyEdit
void moveZeros(vector<int>& nums) {
    int i = 0;
    for (int j = 0; j < nums.size(); ++j)
        if (nums[j] != 0) swap(nums[i++], nums[j]);
}
```

✅ 2. Strings – Core Concepts

📌 Declaration

```
cpp
CopyEdit
string s = "hello";
cin >> s;                // Input without spaces
getline(cin, s);         // Input with spaces
```

🔥 Common String Patterns

Pattern	Description	Example Problem
Two Pointers	Palindromes, substrings	Longest Unique Substring
Hashing	Frequency maps	Anagram, Duplicate chars
Sliding Window	Optimal substrings	Longest substring without repeat

Z-Algorithm / KMP	Pattern matching	Find string in string
Prefix Sum on Chars	Count specific characters fast	Number of a’s in prefix

---

✅ Example Problems (Strings)

a. Check Palindrome

```
cpp
CopyEdit
bool isPalindrome(string s) {
    int i = 0, j = s.size() - 1;
    while (i < j)
        if (s[i++] != s[j--]) return false;
    return true;
}
```

b. Reverse Words in String

```
cpp
CopyEdit
string reverseWords(string s) {
    stringstream ss(s);
    string word, res = "";
    while (ss >> word)
        res = word + " " + res;
    res.pop_back();
    return res;
}
```

✅ 3. STL Functions to Master (Arrays & Strings)

📌 For Arrays / Vectors

```
cpp
CopyEdit
sort(v.begin(), v.end());
reverse(v.begin(), v.end());
accumulate(v.begin(), v.end(), 0); // sum
*max_element(v.begin(), v.end());
count(v.begin(), v.end(), x);
```

📌 For Strings

```
cpp
CopyEdit
sort(s.begin(), s.end());
reverse(s.begin(), s.end());
s.substr(i, len);      // substring
to_string(x);          // int to string
stoi(s);               // string to int
```

📚 Summary: Must-Know Concepts

Area	Must Know
Arrays	Two pointers, Prefix sum, Kadane’s, Sliding window
Strings	Palindrome, Anagram, Frequency, Substrings, Reversal
STL	sort, reverse, max_element, substr, find, count

Would you like a **C++ template project** that includes:

- Menu-driven interface
- All these array & string patterns implemented with examples?

Let me know and I’ll generate it for you.

You said:

Sorting + Binary Search (Basic)

ChatGPT said:

Here’s a **complete guide to using Sorting + Binary Search (Basic level)** — a powerful and commonly used combination in competitive programming and interviews.

✅ Why Combine Sorting + Binary Search?

- Sorting helps to bring structure/order to data
- Binary search lets us search in **O(log n)** on sorted data
- Together, they solve many problems efficiently in **O(n log n)** time

---

🔥 Classic Patterns

Problem Type	Strategy	Time Complexity
Find if an element exists	Sort + Binary Search	O(log n)
Count elements ≤/≥ x	Sort + `lower_bound` / `upper_bound`	O(log n)
Min/max value satisfying condition	Binary Search on Answer	O(log answer × f(n))
Pairs/triplets with sum x	Sort + Two Pointers / Binary Search	O(n log n) / $O(n^2)$

✅ 1. Binary Search Template (Classic)

```
cpp
CopyEdit
int binarySearch(vector<int>& arr, int target) {
    int low = 0, high = arr.size() - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) return mid;
        else if (arr[mid] < target) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
```

✅ 2. Find if Element Exists in Array

```
cpp
CopyEdit
bool exists(vector<int>& arr, int x) {
    sort(arr.begin(), arr.end());  // sort first
    return binary_search(arr.begin(), arr.end(), x);
}
```

✅ 3. Count Elements ≤ X

```
cpp
CopyEdit
int countLessEqual(vector<int>& arr, int x) {
    sort(arr.begin(), arr.end());
    return upper_bound(arr.begin(), arr.end(), x) - arr.begin();
}
```

✅ 4. First and Last Occurrence of Element

```
cpp
CopyEdit
int firstOccurrence(vector<int>& arr, int x) {
    int low = 0, high = arr.size() - 1, ans = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == x) {
            ans = mid;
            high = mid - 1;  // go left
        } else if (arr[mid] < x) low = mid + 1;
        else high = mid - 1;
    }
    return ans;
}
```

✅ 5. Search Insert Position

```
cpp
CopyEdit
int searchInsert(vector<int>& arr, int x) {
    int low = 0, high = arr.size() - 1, ans = arr.size();
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] >= x) {
            ans = mid;
            high = mid - 1;
        } else low = mid + 1;
    }
    return ans;
}
```

✅ 6. Binary Search on Answer (Minimum Valid Value)

E.g., Find the minimum `x` such that `f(x) == true`

```
cpp
CopyEdit
int binarySearchAnswer(int low, int high) {
    int ans = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (check(mid)) {
            ans = mid;
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return ans;
}
```

□ STL Functions for Binary Search

```
cpp
CopyEdit
binary_search(arr.begin(), arr.end(), x);     // true/false
lower_bound(arr.begin(), arr.end(), x);       // ≥ x
upper_bound(arr.begin(), arr.end(), x);       // > x
```

📚 Summary Table

Task	Code Snippet
Find if x exists	`binary_search()`
Count elements ≤ x	`upper_bound() - begin()`
Find first ≥ x	`lower_bound()`
Insert position for x	`lower_bound()` index
Custom condition check	Binary Search on Answer

Would you like a **combined C++ code template** implementing all the above with examples and input/output?

You said:

Maps/Sets (Hashing) Counting elements, uniqueness, frequency.

ChatGPT said:

Here's a summary with theory + C++ code examples for **Maps/Sets (Hashing)** used in counting, uniqueness, and frequency problems:

✅ Theory: Maps and Sets in C++

- **Map** stores key-value pairs. Example: `map<int, int> mp;`
    - Automatically sorted by key (Red-Black Tree).
    - Useful for frequency counting, mapping values.
- **Unordered Map**: `unordered_map<int, int> ump;`
    - Faster (Average O(1)), but no ordering.
- **Set** stores unique elements.
    - Automatically sorted.
- **Unordered Set**: `unordered_set<int> us;`
    - Stores unique elements with faster operations (average O(1)).

📘 Common Use Cases

Use Case	Data Structure
Count frequency	`map` / `unordered_map`
Check uniqueness	`set` / `unordered_set`
Grouping by key	`map`
Element exists or not	`set` / `map.count()`

🔢 Example 1: Count Frequencies of Elements

```
cpp
CopyEdit
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

int main() {
    vector<int> arr = {2, 3, 2, 4, 3, 2};
    unordered_map<int, int> freq;
    for (int x : arr) freq[x]++;

    for (auto [val, count] : freq)
        cout << val << " appears " << count << " times\n";

    return 0;
}
```

✅ Example 2: Check Duplicates Using Set

```
cpp
CopyEdit
#include <iostream>
#include <unordered_set>
#include <vector>
using namespace std;

int main() {
    vector<int> arr = {1, 2, 3, 2};
    unordered_set<int> seen;

    for (int x : arr) {
        if (seen.count(x)) {
            cout << "Duplicate found: " << x << endl;
            break;
        }
        seen.insert(x);
    }

    return 0;
}
```

📘 Example 3: Count Unique Words in a String

```
cpp
CopyEdit
#include <iostream>
#include <set>
#include <sstream>
using namespace std;

int main() {
    string text = "the quick brown fox jumps over the lazy dog";
    set<string> uniqueWords;
    stringstream ss(text);
    string word;

    while (ss >> word)
        uniqueWords.insert(word);

    cout << "Unique words: " << uniqueWords.size() << endl;
    return 0;
}
```

□ Time Complexity

Operation	`map` / `set`	`unordered_map` / `unordered_set`
Insert/Delete/Access	O(log N)	Average O(1), Worst O(N)
Search/Count	O(log N)	Average O(1), Worst O(N)

Would you like 10 practice problems on maps/sets with solutions too?

You said:

Basic Recursion / Backtracking Subset generation, small dfs-style problems.

ChatGPT said:

Here's a full explanation + C++ code examples for **Basic Recursion / Backtracking**, especially for **subset generation** and **DFS-style small problems**.

✅ Theory: Recursion and Backtracking

🔁 Recursion

- Function calls itself with smaller subproblems.
- Used for problems where a task can be divided into similar subtasks.

🔙 Backtracking

- Try all options, go back if a choice doesn't work (undo steps).
- Used for:
  - Generating all **subsets**
  - **Permutations**
  - **Combinations**
  - Solving **mazes**, **Sudoku**, **N-Queens**, etc.

📒 Subset Generation (Power Set)

All subsets of a given set:
`[1, 2] → [], [1], [2], [1, 2]`

◆ Code (Backtracking style)

cpp
CopyEdit

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

void generateSubsets(int i, vector<int>& nums, vector<int>& subset) {
    if (i == nums.size()) {
        cout << "[ ";
        for (int val : subset) cout << val << " ";
        cout << "]\n";
        return;
    }

    // Exclude nums[i]
    generateSubsets(i + 1, nums, subset);

    // Include nums[i]
    subset.push_back(nums[i]);
    generateSubsets(i + 1, nums, subset);
    subset.pop_back();  // backtrack
}

int main() {
    vector<int> nums = {1, 2, 3};
    vector<int> subset;
    generateSubsets(0, nums, subset);
    return 0;
}
```

🔁 Factorial using Recursion

```
cpp
CopyEdit
int factorial(int n) {
    if (n == 0 || n == 1) return 1;
    return n * factorial(n - 1);
}
```

🔍 DFS-style Recursion Example – Maze Path Count

```
cpp
CopyEdit
int countPaths(int i, int j, int n, int m) {
    if (i == n - 1 && j == m - 1) return 1;
    if (i >= n || j >= m) return 0;

    return countPaths(i + 1, j, n, m) + countPaths(i, j + 1, n, m);
}
```

♻️ Backtracking: Print All Permutations

```
cpp
CopyEdit
void backtrack(vector<int>& nums, vector<bool>& used, vector<int>& curr) {
    if (curr.size() == nums.size()) {
        for (int val : curr) cout << val << " ";
        cout << endl;
        return;
    }

    for (int i = 0; i < nums.size(); ++i) {
        if (used[i]) continue;
        used[i] = true;
        curr.push_back(nums[i]);

        backtrack(nums, used, curr);

        used[i] = false;
        curr.pop_back();  // backtrack
    }
}

int main() {
    vector<int> nums = {1, 2, 3};
    vector<bool> used(nums.size(), false);
    vector<int> curr;
    backtrack(nums, used, curr);
    return 0;
}
```

□ Time Complexities

Problem	Time Complexity
Subset generation	O(2^n)
Permutations	O(n!)
Basic DFS	Depends on branching

✓ Common Applications

- Generating all combinations (e.g., k-combinations from n elements)
- Solving puzzles (Sudoku, N-Queens)
- Exploring paths in a matrix or graph
- Knapsack (recursive versions)

Would you like **10 practice problems** for recursion/backtracking as well?