Module 3 of 12

Bit manipulation

Set, clear, toggle and test bits, XOR's cancelling pairs, popcount, and every subset of n ≤ 20 items as a mask from 0 to $2^n - 1$.

An integer is a row of bits, and a row of bits is a set: bit i on means element i is in. That one view gives you O(1) set operations — set, clear, toggle, test — and a way to walk every subset of up to twenty items by counting from 0 to $2^n - 1$.

The module covers the five one-liners every contest solution uses, the builtins (`__builtin_popcountll` , `__builtin_ctzll`), XOR's habit of cancelling equal pairs, and the counting-per-bit trick that turns an $O(n^2)$ sum over pairs into $O(30 \cdot n)$. The exercises find the one unpaired value among thousands of 18-digit numbers, count the subsets that hit a target, and the checkpoint adds up the Hamming distance of every pair.

CONTENTS

1	Bit tricks for contests	15 min
✓	Checkpoint — Bit manipulation	45 min

Exercises in this module compile on Clang 18 · C++20. Reference solutions and the answer key follow each lesson.

LESSON 1 · 15 MIN

Bit tricks for contests

An integer is a row of bits, and a row of bits is a **set**: bit i on means element i is in. Every operation on that set is one instruction.

The five one-liners

```
x |  (1 << i)        // add element i       (set bit i)
x & ~(1 << i)        // remove element i    (clear bit i)
x ^  (1 << i)        // flip element i      (toggle bit i)
(x >> i) & 1         // is i in the set?    (test bit i)
x & (x - 1)          // drop the lowest set bit
```

- Bit 0 is the rightmost. Bit i is worth 2^i.
- Past bit 30, write `1LL << i`: `1 << 40` is undefined behaviour on an `int`.
- Builtins: `__builtin_popcountll(x)` counts set bits, `__builtin_ctzll(x)` counts trailing zeros (not for x = 0).
- `x & -x` keeps only the lowest set bit.

A set of small numbers in one integer: set, toggle, test, clear



1. An int is a row of bits, bit 0 on the right. As a set, bit i is on when element i is in. Start with the empty set, x = 0.
2. x |= 1 << 1 sets bit 1: OR with a mask holding only that bit turns it on and leaves the rest alone. x = 2.
3. x |= 1 << 3 sets bit 3, worth 8: 2 + 8 = 10. The mask now holds the set {1, 3}.
4. x |= 1 << 5 adds element 5. x = 42 = 0b00101010, the set {1, 3, 5} in one integer.
5. x ^= 1 << 1 toggles bit 1: it was on, so it goes off. 42 becomes 40. Toggling again would bring it back.
6. Testing reads, never writes: (x >> 3) & 1 = 1, so 3 is in the set; (x >> 1) & 1 = 0, so 1 no longer is.
7. x &= x − 1 clears the lowest set bit: 40 & 39 = 32. Repeat it until x is 0 and you have counted the set bits — popcount(40) = 2.

XOR cancels pairs

- `x ^ x = 0`, `x ^ 0 = x`, and the order does not matter.
- So XOR of a list where every value appears twice except one **is** that one — O(n) time, O(1) memory.
- Per-bit thinking: a pair differs in bit b when one has a 1 there and the other a 0. With c ones, that is c · (n − c) pairs — summing over bits replaces an O(n²) loop over pairs.

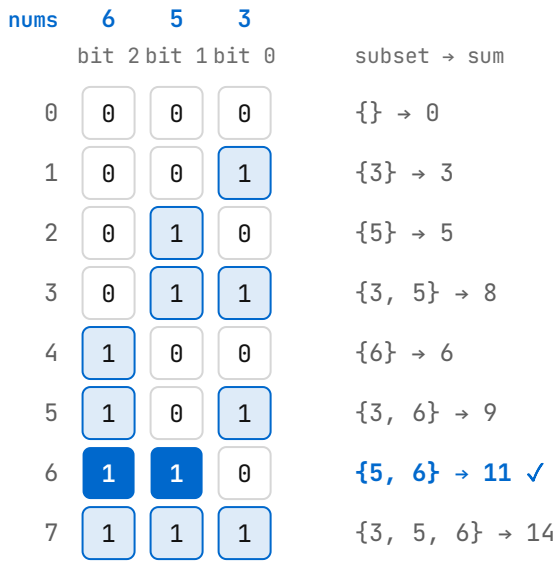
Every subset is a mask

n ≤ 20 in the constraints is a hint: $2^{20} \approx 10^6$ subsets.

```
for (int mask = 0; mask < (1 << n); ++mask) {
    ll s = 0;
    for (int i = 0; i < n; ++i)
        if ((mask >> i) & 1) s += a[i];
    // use the subset's sum s
}
```

- Counting from 0 to $2^n - 1$ visits every pattern of n bits once: every subset, the empty one included.
- Cost $O(2^n \cdot n) \approx 2 \times 10^7$ for n = 20.

Every subset is one mask from 0 to $2^n - 1$



1. Bit i of a mask says whether nums[i] is chosen. mask 0 has no bits on: the empty subset, sum 0. The target is 11.
2. mask 1 = 001 in binary picks {3}, sum 3.
3. mask 2 = 010 in binary picks {5}, sum 5.
4. mask 3 = 011 in binary picks {3, 5}, sum 8.
5. mask 4 = 100 in binary picks {6}, sum 6.
6. mask 5 = 101 in binary picks {3, 6}, sum 9.
7. mask 6 = 110 in binary picks 5 and 6: 5 + 6 = 11, the target. Count it.
8. mask 7 = 111 in binary picks {3, 5, 6}, sum 14, over the target.
9. Counting 0 to $2^n - 1 = 7$ visits every subset exactly once: 1 of 8 hit 11. Cost $O(2^n \cdot n)$, fine for $n \le 20$ — about 2×10^7 steps.

More practice

- Single Number — XOR the list.
- Number of 1 Bits — popcount by hand.
- Counting Bits — popcount of i from i >> 1.
- Subsets — masks from 0 to $2^n - 1$.
- Total Hamming Distance — count per bit.

EXERCISE 1

The odd one out

Every value in the list appears exactly twice — except one, which appears once. Print it.

Constraints: `n` is odd, `1 ≤ n < 2·10⁵`, `1 ≤ aᵢ ≤ 10¹⁸`. Use O(1) extra memory.

Input: `n`, then the `n` values. **Output:** the value that appears once.

```
5
4 1 2 1 2
```

prints

```
4
```

INPUT 1

```
5
4 1 2 1 2
```

OUTPUT 1

```
4
```

REFERENCE SOLUTION

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    ll acc = 0;
    for (int i = 0; i < n; ++i) {
        ll x;
        cin >> x;
        acc ^= x;
    }
    cout << acc << '\n';
    return 0;
}
```

EXERCISE 2

Subsets that hit the target

Count the subsets of the `n` numbers whose sum is exactly `T`. The empty subset has sum 0; equal numbers in different positions are different elements.

Constraints: `1 ≤ n ≤ 20`, `1 ≤ a_i ≤ 10^9`, `0 ≤ T ≤ 2·10^10`.

Input: `n T`, then the `n` numbers. **Output:** the number of subsets.

```
3 11
3 5 6
```

prints

```
1
```

(only {5, 6}.)

INPUT 1

```
3 11
3 5 6
```

OUTPUT 1

```
1
```

REFERENCE SOLUTION

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    ll T;
    cin >> n >> T;
    vector<ll> a(n);
    for (auto& x : a) cin >> x;
    ll count = 0;
    for (int mask = 0; mask < (1 << n); ++mask) {
        ll s = 0;
        for (int i = 0; i < n; ++i)
            if ((mask >> i) & 1) s += a[i];
        if (s == T) ++count;
    }
    cout << count << '\n';
    return 0;
}
```

QUIZ

1. Which expression clears bit `i` of `x` and leaves the others alone?

A. `x | (1 << i)`

B. `x & ~(1 << i)`

C. `x ^ (1 << i)`

D. `x >> i`

2. What is `x & (x - 1)` for x = 12 (binary 1100)?

A. 4

B. 8

C. 11

D. 0

3. `long long x = 1 << 40;` — what goes wrong?

A. Nothing

B. `1` is an int, so `1 << 40` shifts past its 32 bits — undefined behaviour — before the result is converted. Write `1LL << 40`

C. It is a compile error

D. It gives $2^{40} - 1$

4. XOR of all of [6, 9, 6, 4, 9]?

A. 4

B. 0

C. 6

D. 34

ANSWER KEY · BIT TRICKS FOR CONTESTS

1. B `~(1 << i)` is all ones except bit i, so AND keeps every other bit and forces bit i to 0. OR sets it, XOR toggles it (which would turn an off bit on).

2. B x − 1 = 1011. AND with 1100 gives 1000 = 8: the lowest set bit is cleared. Repeating it until x is 0 counts the set bits; x & (x − 1) == 0 tests for a power of two (x > 0).

3. B The literal's type decides the shift's type. Any mask beyond bit 30 needs `1LL << i` (or `1ULL`), and `__builtin_popcountll` for 64-bit values.

4. A The two 6s cancel and the two 9s cancel, whatever their positions; 0 ^ 4 = 4.

CHECKPOINT · 45 MIN

Checkpoint — Bit manipulation

The bit manipulation checkpoint. It is timed: **45 minutes** from when you start it.

What it covers. Setting, clearing, toggling and testing bits; `x & (x - 1)` and `x & -x`; `1LL << i` past bit 30; XOR's cancelling pairs; popcount; and walking every subset as a mask.

How it works. Eleven questions and one program. Clear 70% of the questions and get the program accepted. Its hidden cases hold thousands of numbers, so comparing every pair is far slower than counting one bit position at a time.

Before you start, make sure you can say from memory:

- the expression for each of set, clear, toggle and test;
- which elements mask 13 selects;
- how many pairs differ in a bit where c of the n numbers have a 1.

EXERCISE

Total Hamming distance

The Hamming distance of two numbers is the number of bit positions where they differ — `popcount(x ^ y)`. Print the sum of the Hamming distances over all pairs `i < j`.

Constraints: $1 \le n \le 2 \cdot 10^5$, $0 \le a_i < 2^{30}$.

Input: `n`, then the `n` numbers. **Output:** the total.

```
3
4 14 2
```

prints

```
6
```

(4 and 14 differ in 2 bits, 4 and 2 in 2, 14 and 2 in 2.)

Every pair is $O(n^2) = 2 \cdot 10^{10}$ at the limits — think per bit instead.

INPUT 1

```
3
4 14 2
```

OUTPUT 1

```
6
```

REFERENCE SOLUTION

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    vector<ll> a(n);
    for (auto& x : a) cin >> x;
    ll total = 0;
    for (int b = 0; b < 30; ++b) {
        ll ones = 0;
        for (ll x : a) ones += (x >> b) & 1;
        total += ones * (n - ones);
    }
    cout << total << '\n';
    return 0;
}
```

CHECKPOINT QUESTIONS

1. Which expression tests whether bit `i` of `x` is set?

A. `x | (1 << i)`

B. `(x >> i) & 1`

C. `x ^ i`

D. `x & i`

2. Which expression toggles bit `i`?

A. `x ^ (1 << i)`

B. `x | (1 << i)`

C. `x & (1 << i)`

D. `~x`

3. `x & -x` for x = 40 (binary 101000) is…

A. 8

B. 32

C. 40

D. 1

4. How do you test that a positive `x` is a power of two?

A. `x % 2 == 0`

B. `(x & (x - 1)) == 0`

C. `x >> 1 == 0`

D. `__builtin_popcount(x) > 1`

5. `__builtin_popcountll(x)` returns…

A. The number of set bits in a 64-bit x

B. The position of the highest bit

C. x rounded to a power of two

D. The number of trailing zeros

6. n = 20. How many masks does `for (int m = 0; m < (1 << n); ++m)` visit?

A. 20

B. 400

C. 1,048,576

D. 2,097,152

7. Mask 13 (binary 1101) over a = [a0, a1, a2, a3] selects…

A. {a0, a2, a3}

B. {a1, a2, a3}

C. {a0, a1, a3}

D. {a0, a1, a2}

8. Why does XOR-ing every value find the single unpaired one?

A. Because XOR sorts the values

B. XOR is commutative and associative with x ^ x = 0 and x ^ 0 = x, so every pair cancels wherever it is, leaving the single value

C. Because the values are small

D. It only works if the pairs are adjacent

9. `int` x = 5; what is `x << 1` and `x >> 1`?

A. 10 and 2

B. 6 and 4

C. 10 and 3

D. 2 and 10

10. The sum over all pairs of popcount(a_i ^ a_j), n = $2 \cdot 10^5$ values below 2^{30}, is computed fastest by…

A. Every pair, $O(n^2)$

B. Per bit: with c ones at a bit, $c \cdot (n - c)$ pairs differ there — $O(30 \cdot n)$

C. Sorting the values

D. A prefix array of popcounts

11. Iterating the submasks of `mask` with `for (int s = mask; s; s = (s - 1) & mask)` visits…

A. Every number below mask

B. Every non-empty subset of mask's bits, each once, largest first

C. Only single bits

D. Every mask with the same popcount

ANSWER KEY · CHECKPOINT — BIT MANIPULATION

1. B Shift bit i down to position 0 and mask it off: 1 when set, 0 otherwise. `x & i` tests the bits of the *number* i, not bit i.

2. A XOR with a one-bit mask flips that bit and leaves the rest. `~x` flips every bit.

3. A x & −x isolates the lowest set bit: 40 = 32 + 8, so 8. Fenwick trees step by exactly this amount.

4. B A power of two has a single set bit; clearing the lowest set bit leaves 0. Even numbers like 6 fail the test, as they should.

5. A popcount = population count, the number of 1 bits. `__builtin_ctzll` counts trailing zeros, `__builtin_clzll` leading zeros (both undefined for x = 0). C++20 also has `std::popcount` in `<bit>` for unsigned types.

6. C 2^{20} = 1,048,576 — one per subset, the empty one included. With an O(n) inner loop that is about $2 \cdot 10^7$ steps.

7. A 13 = 8 + 4 + 1: bits 0, 2 and 3 are set, so a0, a2 and a3. Bit 0 is the rightmost digit.

8. B Regroup the terms so each pair sits together: each pair is 0, and 0 ^ single = single. No order is needed.

9. A Shifting left by one doubles (101 → 1010 = 10); shifting right by one halves rounding down (101 → 10 = 2).

10. B Each bit contributes independently; counting ones per bit replaces $2 \cdot 10^{10}$ pair checks with $6 \cdot 10^6$ steps.

11. B Subtracting 1 and AND-ing with mask steps to the next smaller number made only of mask's bits. Add the empty submask 0 separately if you need it. Over all masks of n bits this totals 3^n steps.

CodeKairo · Competitive Programming study plan · Module 3: Bit manipulation. Solve the exercises online to track your progress and earn XP.