Running time (advanced version)

For each of the following algorithms:

- Compute the output of the function for example arguments.
- Determine the running time of the function.
- Figure out what the code is supposed to compute.
- If indicated, locate the bug. A bug refers to a mistake in the program, which may require several fixes.

Foo

Use n for `v.size()`. There is one bug.

```
int foo(vector<int> const& v) {
    int ans = 0;
    for (int i = 0; i < v.size() - 1; ++i)
        if (v[i] == v[i + 1])
            ++ans;
    return ans;
}
cout << foo({0,1,1,0,1,0,0,1}) << '\n';
cout << foo({3,3,2,2,2,9}) << '\n';
```

Bar

Use n for `v.size()`.

```
int bar(vector<int> const& v) {
    int ans = 0;
    for (int i = 0; i < 1 << v.size(); ++i) {
        int tmp = 0;
        for (int j = 0; j < v.size(); ++j) {
            if((i >> j) & 1) {
                tmp ^= v[j];
            }
        }
        ans += tmp;
    }
    return ans;
}
cout << bar({3,5}) << '\n';
cout << bar({1,2,4}) << '\n';
```

Baz

Use n for `v.size()`. There is one bug.

```
array<int, 4> bazar(vector<int> const&v, int l, int r) {
    if (l == r) return {v[l], v[l], v[l], v[l]};
    int m = (l+r)/2;
    auto L = bazar(v, l, m);
    auto R = bazar(v, m, r);
    return {L[0] + R[0], max(L[1], L[0] + R[1]),
        max(L[2] + R[0], R[2]), max({L[3], R[3], L[2] + R[1]})};
}
int baz(vector<int> const&v) {
    assert(!v.empty());
    return bazar(v, 0, v.size())[3];
}
cout << baz({-8,2,-1,3,-4,2}) << '\n';
cout << baz({-1,-1}) << '\n';
```

Qux

There is one bug. The function is also very inefficient (asymptotically, not just by a constant factor), so optimize it.

```
const int p = 1e9+7;
int qux(int a, unsigned int n) {
    if (n == 0) return 1;
    if (n % 2) return qux(a, n-1) * a % p;
    return qux(a, n/2) * qux(a, n/2);
}
cout << qux(2, 5) << '\n' << qux(3, p) << '\n';
```

Flarp

Use n for `v.size()`. There are two bugs.

```
vector<int> flarp(vector<int> const& v) {
    vector<int> ret(v.size()), s;
    for (int e : v) {
        while(!s.empty() && s.back() >= e) s.pop_back();
        ret.push_back(s.back());
        s.push_back(e);
    }
    return ret;
}
```

Bongo

Use n for `v.size()`. There is one bug.

```
vector<int> bongo(vector<int> const& v) {
    vector<int> bo(v.size(), 1);
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < i; ++i)
            if(a[j] < a[i])
                bo[i] += bo[j];
    return accumulate(bo.begin(), bo.end(), 0);
}
```

Snork

There is one bug.

```
int snorkel(int n, int i) {
    if(i <= 1) return 1;
    if(n % i == 0) return 1 + snorkel(i, i - 1);
    else return snorkel(n, i - 1);
}
int snork(int n){
    return snorkel(n, n-1);
}
cout << snork(25) << '\n' << snork(50) << '\n' << snork(210) << '\n';
```

Garply

Assume that `v.size() == 2^n`.

```
vector<int> garply(vector<int> v, int n) {
    for(int i = 0; i < n; ++i) {
        for(int j = 0; j < v.size(); ++j)
            if(((j >> i) & 1) == 0)
                v[j + (1<<i)] += v[j];
    return v;
}
for(int e:garply({3,1,5,7}), 2) cout << e << "\n";
```