Running time

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.

You can assume that somewhere `cout << boolalpha;` has been called, which makes boolean print as "`true`" or "`true`" (as opposed to "`1`" or "`0`").

Foo

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

```
bool foo(vector<int> const& v, int x) {
    for (int i = 1; i <= v.size(); ++i)
        if (v[i] == x)
            return true;
    return false;
}
cout << foo({3,1,4,1,5,9}, 1) << '\n';
cout << foo({3,1,4,1,5,9}, 2) << '\n';
```

Bar

Use n for `v.size()`. You can assume that `v` is sorted (non-decreasing).

```
int bar(vector<int> const& v, int target) {
    int low = -1;
    int high = v.size();
    while (high - low > 1) {
        int mid = (low + high) / 2;
        if (target > v[mid])
            low = mid;
        else
            high = mid;
    }
    return high;
}
cout << bar({1,3,4,7}, 3) << '\n';
cout << bar({1,3,4,7}, 5) << '\n';
```

Baz

You don't need to figure out what this code does (it solves a task from the first round 2017). Just determine the running time.

```
int baz(int a, int b, int c) {
    if (a == -1 && c != -1)
        return 21;
    if (a == -1)
        return 10;
    if (b == -1 || (c != -1 && a > c))
        return 02;
    return 12;
}
```

Qux

There is one bug.

```
vector<int> qux(int n) {
    vector<int> v;
    for (int i = 3; i<n; ++i) {
        int r = i%2;
        for (int j = 3; r && j*j < i; j += 2)
            r = i%j;
        if (r)
            v.push_back(i);
    }
    return v;
}
for (int q : qux(14)) cout << q << '\n';
```

Flarp

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

```
int flarp(string const& s) {
    int x = 1;
    for (int i=0; i < s.size(); ++i)
      for (int p=0; p < 2; ++p)
        for (int l=i, r=i+p; 0<l && r<s.size() && s[l] == s[r]; --l, ++r)
          x = max(x, r-l+1);
    return x;
}
cout << flarp("stofol") << '\n';
cout << flarp("stoffol") << '\n';
```

Bongo

Use n for `a.size()` and m for `b.size()`. You can assume that a and b are sorted (non-decreasing). There is one bug.

```
vector<int> bongo(vector<int> const& a, vector<int> const& b) {
    vector<int> c;
    for (size_t i = 0, j = 0; i < a.size(); ++i) {
        while (j < b.size() && b[j] < a[i]) {
            c.push_back(b[j]);
            ++j;
        }
        c.push_back(a[i]);
    }
    return c;
}
for (int b : bongo({2,3,6,9}, {1,4,5,8})) cout << b << '\n';
```

Snork

Use n for `a.size()` and m for `b.size()`.

```
  bool snork(vector<int> a, vector<int> b, int k) {
  sort(a.begin(), a.end());
  sort(b.begin(), b.end());
  reverse(b.begin(), b.end());

  auto jt = b.begin(), jt_end=b.end();
  for (int x : a) {
    for (;; ++jt) {
      if (jt == jt_end)
        return false;
      if (*jt + x <= k)
        break;
    }
```

```
14      if (*jt + x == k)
15        return true;
16    }
17    return false;
18  }
19  cout << snork({4, 1, 2}, {3, 5, 1}, 7) << '\n';
20  cout << snork({4, 1, 2}, {3, 5, 1}, 8) << '\n';
```