Coding-Tricks Cheat-Sheet

All-Include

```
#include <bits/stdc++.h> // includes everything
using namespace std;     // avoid typing std::
```

Container

vector (“better array”):

```
vector<int> v(n); // v={0,0,...,0}, v.size()==n
v.clear();        // v={}
v.push_back(5);   // v={5}
int x = v[0];     // x=5
for (size_t i=0; i<v.size(); ++i) // custom loop
  cout << v[i] << '\n';
for (vector<int>::iterator it=v.begin(); // iterators
     it!=v.end(); ++it)
  cout << *it << '\n';
for (auto it=v.begin(); it!=v.end(); ++it) // auto
  cout << *it << '\n';
for (auto& elem : v) // range based for
  cout << elem << '\n';
```

map (key-value pairs, access by key, always sorted):

```
map<string, int> m;
m["key"] = 5;     // m={"key": 5}
// If the key doesn't exist, it is inserted with default value
int x = m["new"]; // x=0, m={"key": 5, "new": 0}
for (auto& elem : v) // range based for
  cout << elem.first << ' ' << elem.second << '\n';
```

Other containers: set, multiset, deque (like vector, but with push_front), queue and stack, priority_queue.
In map and map, the smallest element is at the beginning. In priority_queue the largest. To reverse it:

```
map<int, int, greater<int> > m; // largest element in front
set<int, greater<int> > s;

// Smallest element at pq.top()
priority_queue<int, vector<int>, greater<int> > pq;
```

Strings

```
string s; // just like vector<char>
cin >> s; // can input and output
s = "cystoflagellate"
string t = s.substr(2, 5); // start at 2, length 5: t=="stofl"
```

Algorithms

```
vector<int> v{5,4,3,2,1};
sort(v.begin(), v.end());       // sort everything
sort(v.begin()+1, v.begin()+4); // sort at indices 1,2,3
// custom compare function: is lhs<rhs?
bool comp(int lhs, int rhs) { return lhs<rhs; }
sort(v.begin(), v.end(), comp); // sort using comp
// define custom less than operator
bool operator<(const mystruct& lhs, const mystruct& rhs) {...}
```

Other useful functions:

```
vector<int> v{1,2,2,5,7,7,7,8};
// v must be sorted
bool b = binary_search(v.begin(), v.end(), 5); // b==true
auto it = lower_bound(v.begin(), v.end(), 5); // points to 5
v.erase(unique(v.begin(), v.end()), v.end()); // v={1,2,5,7,8}
// v can be arbitrary
auto it = find(v.begin(), v.end(), 5); // it points to first 5
reverse(v.begin(), v.end()); // reverse v
```

Terminal

```
# Navigation
$ cd directory  # change directory
$ pwd           # show directory name
$ ls            # list files in directory
$ ls *.in       # list all files that end with .in

# Compiling C++ (program prog.cc)
$ g++ -Wall -Wextra -std=c++11 -g3 -ggdb3 \
  -D_GLIBCXX_DEBUG prog.cc -o prog
# -Wall -Wextra:
#    Enable warnings and extra warnings
# -std=c++11 or -std=c++14:
#    Enable C++11 resp. C++14.
# -g3 -ggdb3: Write debug informations for gdb
# -D_GLIBCXX_DEBUG:
#   Bound-checking for containers and iterators

# testing
$ ./prog <sample01.in # input sample01.in
# input all *.in
$ for f in *.in; do echo "-- $f --"; ./prog <$f; done

# using gdb
$ gdb prog
(gdb) run <sample01.in
(gdb) bt  # show backtrace
(gdb) q   # quit
```

printf-Debugging

```
// During debugging
#define DEB(x) cerr << x << '\n'

// When submitting
#define DEB(x) // cerr << x << '\n'

for (int i=0; i<10; ++i) {
  DEB("i=" << i << " sum=" << sum);
  sum += i;
}
```

Oder compile locally with -DDEBUG (which is not defined on the grader)

```
#ifdef DEBUG
#  define DEB(x) cerr << x << '\n'
#else
#  define DEB(x)
#endif
```

I/O-Optimizations

```
// Turn off synchronization between cin/cout and scanf/printf
ios_base::sync_with_stdio(false);
// Disable automatic flush of cout when reading from cin
cin.tie(0);
```

Strategy

Runtime analysis: 1 second = 10^7 calculation steps.
Problem solving:

1. Look at limits and guess runtime
2. Try solve examples
3. Have an idea
4. Show correctness, think of counterexamples
5. Think how to implement it
6. Code

If wrong answer:

1. Do you have a counterexample? If yes: printf-Debugging
2. Tink of counterexamples and corner cases and test them
3. Why is the approach correct?
4. Read task description again
5. Read code
6. Try other tasks
7. Write brute force solution, automatically generate tests and check against your solution

]cheatsheet.pyg]cheatsheet.out.pyg