SOI C++ Cheat-Sheet

C++ Template (with <soi> header)

```cpp
#include <soi>  // the soi header -- sets up everything

signed main() {
  ...
}
```

Interactive Tasks

```cpp
signed main() {
  interactive_task();
}
```

Redirecting IO

```cpp
signed main() {
  redirect_input("sample01.in");   // read from sample01.in
  redirect_output("sample01.out"); // write to sample01.out
}
```

C++ Template (without <soi>)

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

#define int int64_t      // make int to long long

signed main() {          // main must return int, and
                         // signed is an alias for int
  // Disable synchronization between cin/cout and scanf/printf
  ios_base::sync_with_stdio(false);

  // Disable automatic flush of cout when reading from cin
  // => REMOVE THIS FOR INTERACTIVE TASKS!
  cin.tie(0);
}
```

Redirecting IO

```cpp
signed main() {
  freopen("sample01.in", "r", stdin);
  freopen("sample01.out", "w", stdout);
}
```

Compile and Run

```
# Compiling C++ (program prog.cpp)
$ g++ -Wall -Wextra -std=c++17 -g3 -ggdb3 \
  -fsanitize=address,undefined \
  -D_GLIBCXX_DEBUG prog.cpp -o prog
# -Wall -Wextra:
#    Enable warnings and extra warnings
# -std=c++17:
#    Enable C++17 features
# -g3 -ggdb3: Write debug informations for gdb
# -D_GLIBCXX_DEBUG:
#   Bounds-checking for containers and iterators
# -fsanitize=address,undefined: catch undefined behaviour

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

Container Overview

vector<T>: `v[i]`, `v.push_back(x)`, `v.pop_back()`.
map<K,V>: `m[k]=v`, `m.erase(k)`, `m.count(k)`.
set<T>: `s.insert(x)`, `s.erase(x)`, `s.count(x)`.
multimap<K,V>/multiset<T>: like map/set but allows duplicated keys.
deque<T>: like `vector`, but with `push_front` and `pop_front`
string: like `vector<char>` but works with `cin`.
queue<T>/stack<T>: use deque<T> or `vector<T>` instead.
priority_queue<T>: Has operations (`top()`, `pop()`, `push(x)`), the largest element is at the beginning. To reverse the order:

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

Containers

vector ("better array"):

```cpp
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):

```cpp
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';
for (auto& [key, value] : v) // structured decomposition
  cout << key << ' ' << value << '\n';
```

Algorithms

```cpp
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:

```cpp
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
```

Pass by value/reference

Bad: Pass by values. Makes a copy. For vectors: $\mathcal{O}(n)$:

```cpp
int top(vector<int> v) { return v.back(); }
```

Ugly: Pass by mutable reference ($\mathcal{O}(1)$):

```cpp
int top(vector<int>& v) { return v.back(); }
```

Good: Pass by const reference ($\mathcal{O}(1)$):

```cpp
int top(vector<int> const& v) { return v.back(); }
```

Operator overloading

```cpp
struct Point {
    int x, y, id;
};
bool operator<(Point const& a, Point const& b){
    return a.x < b.x;
}
bool operator==(Point const& a, Point const& b){
    return a.x == b.x;
}
Point operator+(Point const& a, Point const& b){
    return Point{a.x+b.x, a.y+b.y, -1};
}
```

Pair, Tuple, Array

Pairs: Two fields, has comparison operators by default

```cpp
pair<int, char> p{42, 'x'};
cout << p.first << " " << p.second; // 42 x
pair<int, char> q = make_pair(42, 'x');
p.second = 'n';
auto [a, b] = p; // unpack
```

Tuple: More fields than a tuple

```cpp
tuple<int, int, bool, int, char> p;
auto [a, b, c, d, e, f] = p; // unpack
bool x = get<2>(p); // get at index
```

Arrays: use if all types are the same

```cpp
array<int, 2> point{4, 2};
auto [x, y] = p; // unpack like in a tuple
cout << p[0] << ' ' << p[1] << '\n'; // index like an array
```

Debugging with GDB

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

Grader errors

TLE (Time limit exceeded): Estimate if your algorithm is fast enough. Set $\log(x) \approx 10$ (for all x) and plug in limits. 10^7 always passes, 10^8 if you optimize, 10^9 usually not.
RE (Runtime error): Out of bounds? Assertion error? Very deep recursion? Too much memory? Pop empty stack?