

Competitive Programming Notebook

Daniel Santiago Lopez Daza
dlopezd@unal.edu.co

Tomás David Rodriguez Agudelo
trodrigueza@unal.edu.co

Eduard Joel Ostos Castro
eostos@unal.edu.co

Last updated: July 13, 2024

Contents

2 Template

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define int long long
5 #define pb push_back
6 #define pii pair<int, int>
7 #define vi vector<int>
8 #define vb vector<bool>
9 #define vpi vector<pair<int, int>>
10 #define vvi vector<vector<int>>
11 #define ff first
12 #define ss second
13 #define all(container) container.begin(), container.end()
14
15 void solve()
16 {
17 }
18
19 int32_t main()
20 {
21 #ifndef ONLINE_JUDGE
22     freopen("input.txt", "r", stdin);
23     freopen("out.txt", "w", stdout);
24     freopen("error.txt", "w", stderr);
25 #endif
26     ios::sync_with_stdio(false);
27     cin.tie(nullptr);
28     int t; cin >> t;
29     while(t--) solve();
30 }
```

3 Macros

1. Debugging:

(a) **assert(x)**: If x is false, the program will terminate. ej:

```
1 assert(1 == 2);
2 // Will terminate the program
3
```

2. Container related Macros:

(a) **all(x)**

```
1 #define all(x) x.begin(), x.end()
2
```

(b) **pb**

```
1 #define pb push_back
2
```

3. Types:

(a) **int**

```
1 #define int long long
2
```

(b) **vi, vb and vpi**

```
1 #define vi vector<int>
2 #define vb vector<bool>
3 #define vpi vector<pair<int, int>>
4
```

(c) **pi**

```
1 #define pi pair<int, int>
2
```

(d) **ff and ss**

```
1 #define ff first
2 #define ss second
3
```

4 Container Classes and Initialization

1. Vector:

(a) Class:

```
1 template < class T,
2 class Alloc = allocator<T>> class vector;
3
```

(b) Initialization:

- i. **vector<int> v**: Declares a vector of integers.
- ii. **vector<int> v(n)**: Declares a vector of integers of size n.
- iii. **vector<int> v(n, x)**: Declares a vector of integers of size n, with all elements initialized to x.
- iv. **vector<int> v = {1, 2, 3, 4}**: Declares a vector of integers with the elements 1, 2, 3 and 4.
- v. **vector<int> v {1, 2, 3, 4}**: Declares a vector of integers with the elements 1, 2, 3 and 4.

2. Set:

(a) Class:

```

1  template < class T,
2  class Compare = less<T>, \\ View Interesting Classes Chapter
   for more information
3  class Alloc = allocator<T>> class set;
4

```

(b) Initialization:

- i. `set<int> s`: Declares a set of integers.
- ii. `set<int> s {1, 2, 3, 4}`: Declares a set of integers with the elements 1, 2, 3 and 4.
- iii. `set<int> s = {1, 2, 3, 4}`: Declares a set of integers with the elements 1, 2, 3 and 4.

5 Functions

Containers Functions

1. Vector:

(a) `push_back(x)`: Insert element `x` at the end of the vector. ej:

```

1  vector<int> v;
2  v.push_back(5);
3  // New value of v = {5}
4

```

Note: Amortized complexity of $O(1)$.

(b) `pop_back()`: Remove the last element of the vector. ej:

```

1  vector<int> v ({5, 6, 7});
2  v.pop_back();
3  // New value of v = {5, 6}
4

```

Note: Complexity of $O(1)$.

(c) `size()`: Return the number of elements in the vector. ej:

```

1  vector<int> v ({5, 6, 7});
2  v.size();
3  // Will return 3
4

```

Note: Complexity of $O(1)$.

(d) `at(i)`: Access the element at index `i`. ej:

```

1  vector<int> v ({5, 6, 7});
2  v.at(1);
3  // Will return 6
4

```

Note: Complexity of $O(1)$.

(e) `clear()`: Remove all elements from the vector. ej:

```

1  vector<int> v ({5, 6, 7});
2  v.clear();
3  // New value of v = {}
4

```

Note: Complexity of $O(n)$.

2. Map:

(a) `insert({key, value})`: Insert key-value pair in the map. ej:

```

1  map<int, int> m;
2  m.insert({5, 10});
3  // New value of m = { {5, 10} }
4

```

Note: Complexity of $O(\log_2 n)$.

(b) `erase(key)`: Erase element with the given key from the map. ej:

```

1  map<int, int> m ({ {5, 10}, {6, 20} });
2  m.erase(5);
3  // New value of m = { {6, 20} }
4

```

Note: Complexity of $O(\log_2 n)$.

(c) `find(key)`: Find element with the given key in the map. ej:

```

1  map<int, int> m;
2  m.find(5);
3  // Will return m.end() if key is not in the map
4

```

Note: Complexity of $O(\log_2 n)$.

(d) `lower_bound(key)`: Find the first element that is not less than the given key. ej:

```

1  map<int, int> m ({ {5, 10}, {6, 20} });
2  m.lower_bound(6);
3  // Will return an iterator to the element {6, 20}
4

```

Note: Complexity of $O(\log_2 n)$. **Note:** Uses Binary Search under the hood.

(e) `upper_bound(key)`: Find the first element that is greater than the given key. ej:

```

1  map<int, int> m ({ {5, 10}, {6, 20} });
2  m.upper_bound(6);
3  // Will return an iterator to the end of the map
4

```

Note: Complexity of $O(\log_2 n)$. **Note:** Uses Binary Search under the hood.

3. Multiset:

- (a) `insert(x)`: Insert element `x` in the multiset. ej:

```
1 multiset<int> ms;  
2 ms.insert(5);  
3 // New value of ms = {5}  
4
```

Note: Complexity of $O(\log_2 n)$.

- (b) `erase(x)`: Erase all instances of element `x` from the multiset. ej:

```
1 multiset<int> ms ({5, 6, 7, 5});  
2 ms.erase(5);  
3 // New value of ms = {6, 7}  
4
```

Note: Complexity of $O(\log_2 n)$.

- (c) `find(x)`: Find an instance of element `x` in the multiset. ej:

```
1 multiset<int> ms;  
2 ms.find(5);  
3 // Will return ms.end() if x is not in the multiset  
4
```

Note: Complexity of $O(\log_2 n)$.

- (d) `lower_bound(x)`: Find the first element that is not less than `x`. ej:

```
1 multiset<int> ms ({5, 6, 7});  
2 ms.lower_bound(6);  
3 // Will return an iterator to the element 6  
4
```

Note: Complexity of $O(\log_2 n)$. **Note:** Uses Binary Search under the hood.

- (e) `upper_bound(x)`: Find the first element that is greater than `x`. ej:

```
1 multiset<int> ms ({5, 6, 7});  
2 ms.upper_bound(6);  
3 // Will return an iterator to the element 7  
4
```

Note: Complexity of $O(\log_2 n)$. **Note:** Uses Binary Search under the hood.

4. Unordered Set:

- (a) `insert(x)`: Insert element `x` in the unordered set. ej:

```
1 unordered_set<int> us;  
2 us.insert(5);  
3 // New value of us = {5}  
4
```

Note: Average complexity of $O(1)$, worst-case complexity of $O(n)$.

- (b) `erase(x)`: Erase element `x` from the unordered set. ej:

```
1 unordered_set<int> us ({5, 6, 7});  
2 us.erase(5);  
3 // New value of us = {6, 7}  
4
```

Note: Average complexity of $O(1)$, worst-case complexity of $O(n)$.

- (c) `find(x)`: Find element `x` in the unordered set. ej:

```
1 unordered_set<int> us;  
2 us.find(5);  
3 // Will return us.end() if x is not in the unordered set  
4
```

Note: Average complexity of $O(1)$, worst-case complexity of $O(n)$.

- (d) `bucket_count()`: Return the number of buckets in the unordered set. ej:

```
1 unordered_set<int> us ({5, 6, 7});  
2 us.bucket_count();  
3 // Will return the number of buckets  
4
```

Note: Complexity of $O(1)$.

- (e) `load_factor()`: Return the load factor of the unordered set. ej:

```
1 unordered_set<int> us ({5, 6, 7});  
2 us.load_factor();  
3 // Will return the load factor  
4
```

Note: Complexity of $O(1)$.

5. Unordered Map:

- (a) `insert({key, value})`: Insert key-value pair in the unordered map. ej:

```
1 unordered_map<int, int> um;  
2 um.insert({5, 10});  
3 // New value of um = { {5, 10} }  
4
```

Note: Average complexity of $O(1)$, worst-case complexity of $O(n)$.

- (b) `erase(key)`: Erase element with the given key from the unordered map. ej:

```
1 unordered_map<int, int> um ({ {5, 10}, {6, 20} });  
2 um.erase(5);  
3 // New value of um = { {6, 20} }  
4
```

Note: Average complexity of $O(1)$, worst-case complexity of $O(n)$.

- (c) `find(key)`: Find element with the given key in the unordered map. ej:

```
1 unordered_map<int, int> um;  
2 um.find(5);  
3 // Will return um.end() if key is not in the unordered map  
4
```

Note: Average complexity of $O(1)$, worst-case complexity of $O(n)$.

(d) `bucket_count()`: Return the number of buckets in the unordered map. ej:

```
1 unordered_map<int, int> um ({ {5, 10}, {6, 20} });
2 um.bucket_count();
3 // Will return the number of buckets
4
```

Note: Complexity of $O(1)$.

(e) `load_factor()`: Return the load factor of the unordered map. ej:

```
1 unordered_map<int, int> um ({ {5, 10}, {6, 20} });
2 um.load_factor();
3 // Will return the load factor
4
```

Note: Complexity of $O(1)$.

6. Order Statistics Tree (using tree from `<ext/pb_ds/assoc_container.hpp>`):

(a) `insert(x)`: Insert element x in the order statistics tree. ej:

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 tree<int, null_type, less<int>, rb_tree_tag,
4   tree_order_statistics_node_update> ost;
5 ost.insert(5);
6 // New value of ost = {5}
```

Note: Complexity of $O(\log_2 n)$.

(b) `erase(x)`: Erase element x from the order statistics tree. ej:

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 tree<int, null_type, less<int>, rb_tree_tag,
4   tree_order_statistics_node_update> ost ({5, 6, 7});
5 ost.erase(5);
6 // New value of ost = {6, 7}
```

Note: Complexity of $O(\log_2 n)$.

(c) `find_by_order(k)`: Find the k-th smallest element in the order statistics tree. ej:

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 tree<int, null_type, less<int>, rb_tree_tag,
4   tree_order_statistics_node_update> ost ({5, 6, 7});
5 ost.find_by_order(1);
6 // Will return an iterator to the element 6
```

Note: Complexity of $O(\log_2 n)$.

(d) `order_of_key(x)`: Find the number of elements strictly less than x in the order statistics tree. ej:

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 tree<int, null_type, less<int>, rb_tree_tag,
4   tree_order_statistics_node_update> ost ({5, 6, 7});
5 ost.order_of_key(6);
6 // Will return 1
```

Note: Complexity of $O(\log_2 n)$.

(e) `clear()`: Remove all elements from the order statistics tree. ej:

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 tree<int, null_type, less<int>, rb_tree_tag,
4   tree_order_statistics_node_update> ost ({5, 6, 7});
5 ost.clear();
6 // New value of ost = {}
```

Note: Complexity of $O(n)$.

7. Set:

(a) `insert(x)`: Insert element x in the set. ej:

```
1 set<int> s;
2 s.insert(5);
3 // New value of s = {5}
4
```

Note: Complexity of $O(\log_2 n)$.

(b) `erase(x)`: Erase element x from the set. ej:

```
1 set<int> s ({5, 6, 7});
2 s.erase(5);
3 // New value of s = {6, 7}
4
```

Note: Complexity of $O(\log_2 n)$.

(c) `find(x)`: Find element x in the set. ej:

```
1 set<int> s;
2 s.find(5);
3 // Will return s.end() if x is not in the set
4
```

Note: Complexity of $O(\log_2 n)$.

(d) `lower_bound(x)`: Find the first element that is not less than x. ej:

```
1 set<int> s ({5, 6, 7});
2 s.lower_bound(6);
3 // Will return an iterator to the element 6
4
```

Note: Complexity of $O(\log_2 n)$. **Note:** Uses Binary Search under the hood.

(e) `upper_bound(x)`: Find the first element that is greater than x. ej:

```
1  set<int> s ({5, 6, 7});  
2  s.upper_bound(6);  
3  // Will return an iterator to the element 7  
4
```

Note: Complexity of $O(\log_2 n)$. **Note:** Uses Binary Search under the hood.

6 Greedy

7 Two Pointers

Binary Search

Ternary Search

10 Array and Range Sums

11 Dynamic Programming

