SoC 2023: Competitive Programming
Week-1: Basics and C++ STL

Mentor: Virendra Kabra

Summer 2023

Contents

1 Resources

- Competitive Programming: Need to solve small problems in a limited time. Most of these are based on standard concepts that we are going to cover. For acceptance, solutions must adhere to a specified time-limit and memory-limit.
- Online Platforms for contests and practice: Codeforces, Codechef, AtCoder, SPOJ, CSES, etc. We would mainly do problems from Codeforces and CSES Problem Set.
  - Can filter questions on specific topics or rating levels on Codeforces
  - CSES has topic-wise sets
- References:
  - CP: Competitive Programmer's Handbook (CSES) by Antti Laaksonen, cp-algorithms, Competitive Programming by Steven and Felix Halim, blogs on Codeforces and other websites
  - Theoretical DSA: CLRS, Kleinberg-Tardos
- Most programmers use C++ for smaller execution time and a convenient template library. Some use Python, Java, etc.

2 Basics

2.1 Template

- Starting Template
- Get VSCode to identify the path to `bits/stdc++.h`: YT Link
- VSCode user snippets
- `ios_base::sync_with_stdio(false); cin.tie(0);`
  disables synchronization between streams and unties `cin` from `cout`. Buffering `cout` has a side-effect of speed improvement.
- `"\n"` vs `endl`
  `endl` flushes the output. Use `"\n"` for better execution speed. Interactive Questions require flushing of output buffer. Use `cout.flush()` and the like.
- Random numbers
- File read and write for debugging and testing
- `define` macros and `typedef`s
- `long long`, `unsigned long long`

2.2 Time Complexity

- Big-O Notation: On a high level, keep the dominant (faster growing) term in a function to get the "order" with which it grows. Constant factors are ignored. Examples:
  - $3n^2 + 2n + 5 = O(n^2)$. Grows like n^2
  - $n + \log n = O(n)$, as n grows faster than $\log n$
  - $f(n) = 10^{100}n$ is $O(n)$ and $g(n) = n^2$ is $O(n^2)$. However, for all practical purposes, g is faster
- Go through this CF blog post or Chapter-2 of the Handbook. It has examples on how to compute time-complexity in terms of the inputs.
- Constraints would be mentioned in the problem: with C++, roughly $10^6 - 10^7$ operations/second. So if $n \leq 10^5$ and time limit ≤ 1 second, then $O(n^2)$ won't work (TLE). If $n \leq 20$, then brute force (2^n) might work.

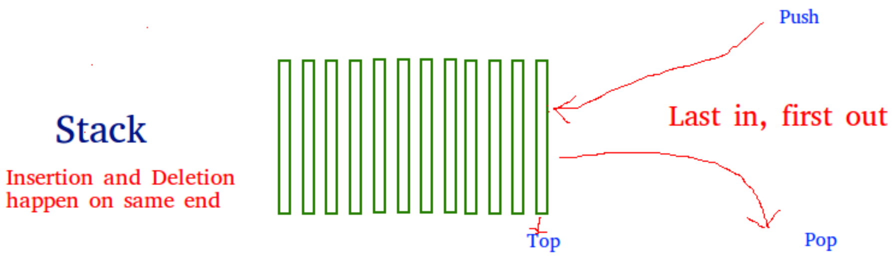
2.3 Operators

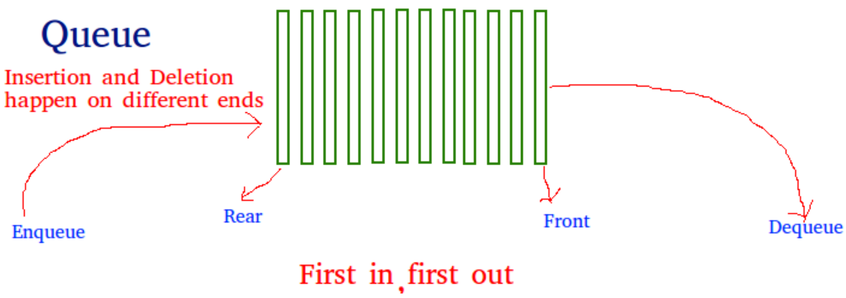
- Commonly used *bitwise* operators `|, &, ~, <<, >>, ^`
- GFG Reference
- XOR
  - Exclusive OR: `0^0=0, 1^1=0, 0^1=1, 1^0=1`
  - `p^0 = p, p^(1...1) = ~p, p^~p = 1...1`. Note that `1...1` is -1 in 2's complement notation
  - XOR is associative and commutative

3 C++ STL Data Structures

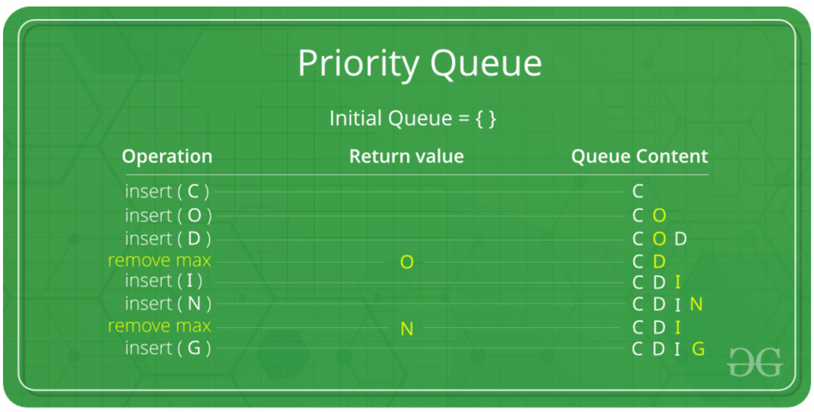
Some commonly used data structures with their methods.

1. `vector`
   - Dynamic length array. Memory for elements is contiguously allocated on heap
   - GFG Reference for methods and respective time-complexities. Constant-time access, linear-time insertion and deletion. Insertion at the end is *amortized* constant-time.
   - File
2. `string`
   - Dynamic `char` array. Many convenient STL functions
   - Methods
   - File
3. `stack`, `queue`, `deque`, `priority_queue` `list`. Images from GFG
   - Stack: Last-In-First-Out
   - Queue: First-In-First-Out





   - Double Ended Queue: Allows insertions and deletions from either end
   - Priority Queue: Top of queue is greatest/least (or custom comparison criteria)



- List: Similar to vector, but with non-contiguous memory allocation. This allows constant-time insertions and deletions, but slower access.
- Methods and time-complexities: stack, queue, priority queue, deque, list
- File

4. pair, map, set

- Pair: Two elements, first and second. For example, {int, vector<int>}
- Set: Collection of elements. Similar to vector, but faster lookups, deletes. For example, {-5,0,3}
  - Ordered: Elements are ordered by some comparator. For example, default is ascending order for an ordered set of ints.
  - Unordered: Not ordered, allowing faster inserts, lookups, deletions. However, time-complexity constants might be higher.
  - STL: set and unordered_set
- Map: Key-value pairs. For example, map<int, unsigned int> with entries {{-5, 5}, {1, 0}, {20, 10}}.
  Similar to sets, we have ordered and unordered maps: map and unordered_map. Ordering is done by keys.

```
                  | map               | unordered_map
---------------------------------------------------------
Ordering         | increasing order   | no ordering
                 | of keys(by default) |

Implementation   | Self balancing BST  | Hash Table
                 | like Red-Black Tree |

search time      | log(n)              | O(1) -> Average
                 |                     | O(n) -> Worst Case

Insertion time   | log(n) + Rebalance  | Same as search

Deletion time    | log(n) + Rebalance  | Same as search
```

- A popular question: Given an array of integers and a target sum, is there a pair that add up to the target sum?
  $O(n^2), O(n \log n), O(n)$
- File

5. Bitset. File

4 Todos

- Visit links in this document
- Start out with Chapters 1 and 4 of the CP handbook
- CSES Introductory Problems