COMPETITIVE
PROGRAMMING
CLUB

Membership sign up:

Week 2

Week 1 Contest Review

COMPETITIVE
PROGRAMMING
CLUB

Leaderboard

- Everyone in a team of up to three people will get the credit for every problem the team solves
- Teams of more than three people will receive credit for 3 / [team size] times number of problems solved
    - For example, each member of a team of 4 will get ¾ credit for solved problems
- Remember that official contests have teams of one to three people!

Current Standings (Top 10):

Tied for 1st place with 10 points:

Nhu Di Nguyen
Nathan Weiss
Max McEvoy
Martin Liu
Khoa Nguyen
Alex Chen

In 7th place with 8 points:

Md Abu Quwsar Ohi

Tied for 8th place with 7 points:

Ethan Switzer
Alan Bach
Ahsan Tariq

Knot Knowledge (B)

Goal:

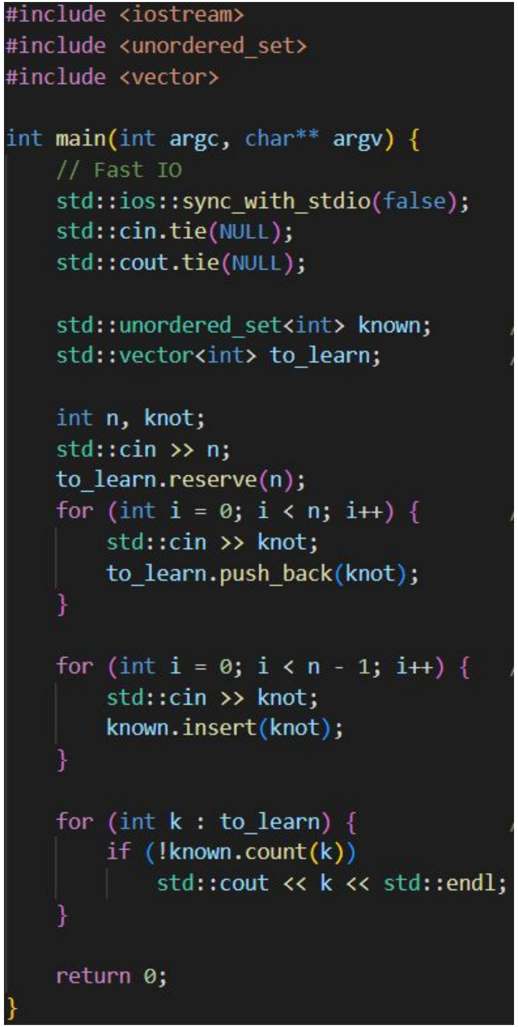
Determine which knot Sonja still needs to learn

Notes:

- Checking for membership of a hash set can be done in ~O(1) time

Solution:

Store the knots needed to learn so that we can easily iterate over them (ie. in an array/vector/ArrayList/list/etc. language dependant). Store the knots Sonja already knows in a hash set (unordered_set/HashSet/set/etc. also language dependant). Iterate over the knots needed to learn and check if the knot is not in the known set. If so we have found the knot to learn.



```
#include <iostream>
#include <unordered_set>
#include <vector>

int main(int argc, char** argv) {
    // Fast IO
    std::ios::sync_with_stdio(false);
    std::cin.tie(NULL);
    std::cout.tie(NULL);

    std::unordered_set<int> known;
    std::vector<int> to_learn;

    int n, knot;
    std::cin >> n;
    to_learn.reserve(n);
    for (int i = 0; i < n; i++) {
        std::cin >> knot;
        to_learn.push_back(knot);
    }

    for (int i = 0; i < n - 1; i++) {
        std::cin >> knot;
        known.insert(knot);
    }

    for (int k : to_learn) {
        if (!known.count(k))
            std::cout << k << std::endl;
    }

    return 0;
}
```

Sum of the Others (D)

Goal:

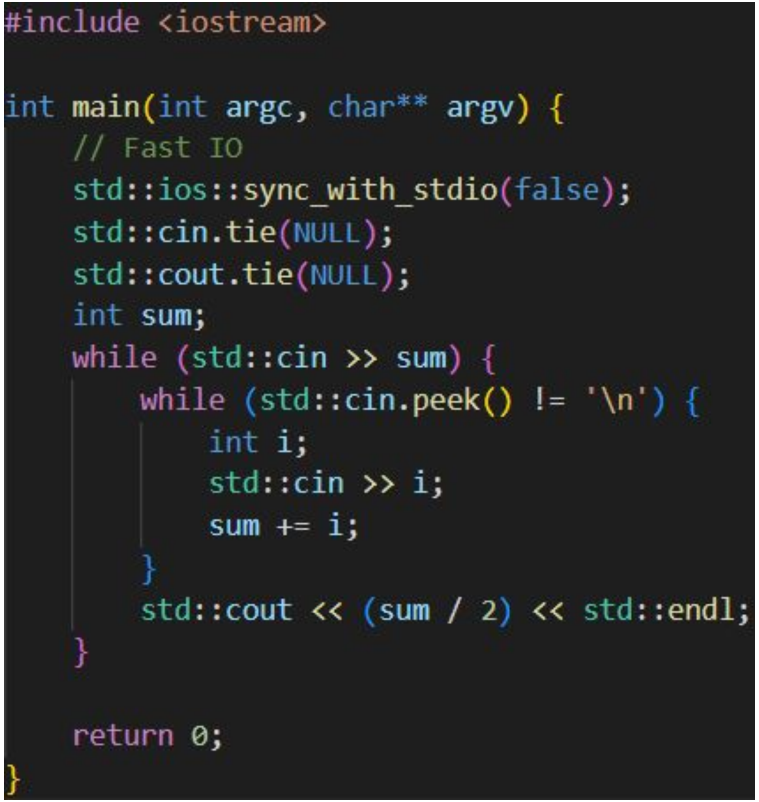
Determine which integer is the sum of the others

Notes:

- Given the integers $[x_1, x_2, \ldots , x_n]$ $x_1 + x_2 + \ldots + x_n - x_a = x_a$ for some a such that $1 \leq a \leq n$
- $x_1 + x_2 + \ldots + x_n = x_a + x_a = 2x_a$

Solution:

Take the sum of all the numbers and divide by two



```
#include <iostream>

int main(int argc, char** argv) {
    // Fast IO
    std::ios::sync_with_stdio(false);
    std::cin.tie(NULL);
    std::cout.tie(NULL);
    int sum;
    while (std::cin >> sum) {
        while (std::cin.peek() != '\n') {
            int i;
            std::cin >> i;
            sum += i;
        }
        std::cout << (sum / 2) << std::endl;
    }

    return 0;
}
```

Sun and Moon (E)

Goal:

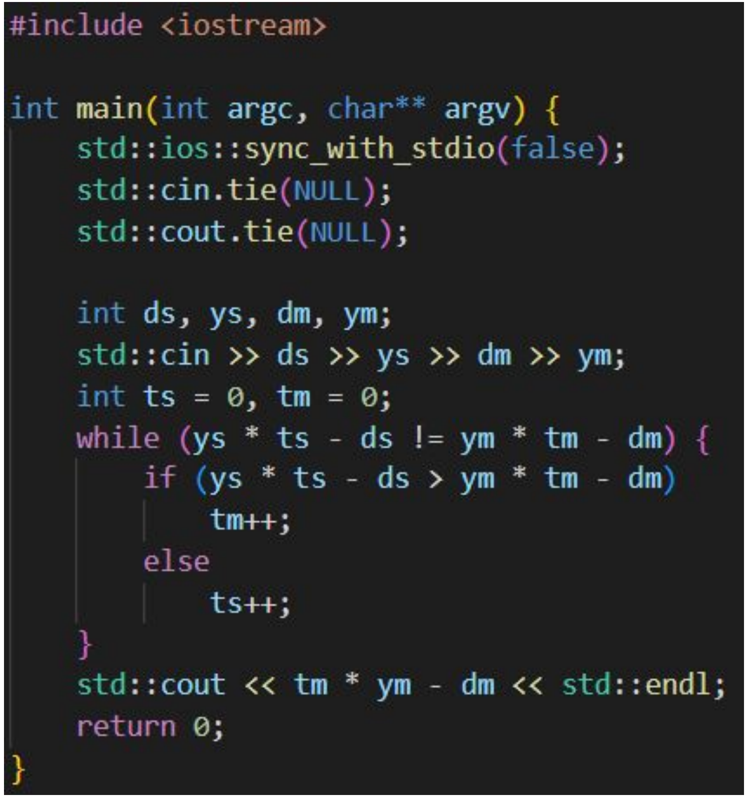
Determine how many year it will be until the next eclipse.

Notes:

- The years the sun will be in the right position are parameterized by $S(t) = y_s \cdot t - d_s , t \in Z$
- The years the moon will be in the right position are parameterized by $M(t) = y_m \cdot t - d_m , t \in Z$

Solution:

Pick two values for t (t_s and t_m) and “march” them until a solution is found ie. $S(t_s) = M(t_m)$ then calculate the year



```cpp
#include <iostream>

int main(int argc, char** argv) {
    std::ios::sync_with_stdio(false);
    std::cin.tie(NULL);
    std::cout.tie(NULL);

    int ds, ys, dm, ym;
    std::cin >> ds >> ys >> dm >> ym;
    int ts = 0, tm = 0;
    while (ys * ts - ds != ym * tm - dm) {
        if (ys * ts - ds > ym * tm - dm)
            tm++;
        else
            ts++;
    }
    std::cout << tm * ym - dm << std::endl;
    return 0;
}
```

Integer Division (H)

Goal:

Count how many pairs of indices (i, j) there are such that $a_i / d = a_j / d$, $i \neq j$

Notes:

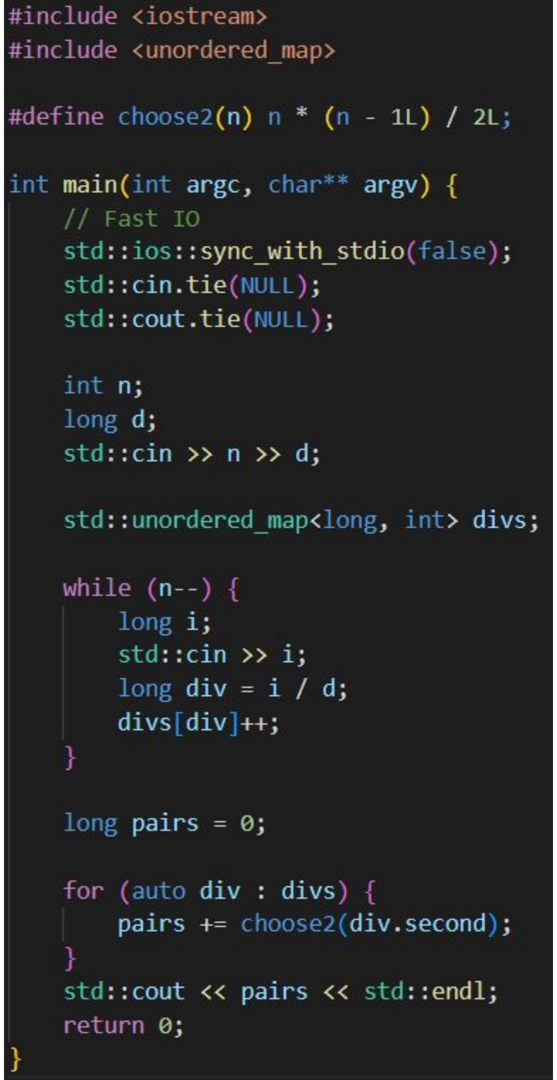
- Let n be the number of indices, then there are n choose 2 pairs of indices
- Hash maps have near constant lookup and update time

Solution:

For each integer calculate the quotient and increment that entry in the hash map. Then iterate over the entries and take the sum of the counts choose 2.

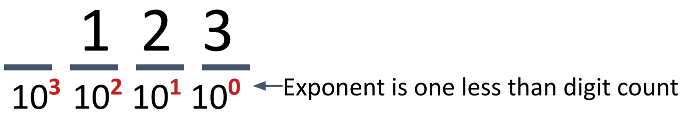
Implementation notes:

1. Be careful when working with large numbers. Use long integers (64-bit integers) whenever applicable. Some languages will automatically cast down to 32-bit integers if you use 32-bit integer literals



```cpp
#include <iostream>
#include <unordered_map>

#define choose2(n) n * (n - 1L) / 2L;

int main(int argc, char** argv) {
    // Fast IO
    std::ios::sync_with_stdio(false);
    std::cin.tie(NULL);
    std::cout.tie(NULL);

    int n;
    long d;
    std::cin >> n >> d;

    std::unordered_map<long, int> divs;

    while (n--) {
        long i;
        std::cin >> i;
        long div = i / d;
        divs[div]++;
    }

    long pairs = 0;

    for (auto div : divs) {
        pairs += choose2(div.second);
    }
    std::cout << pairs << std::endl;
    return 0;
}
```

How Many Digits? (J)



Exponent is one less than digit count

Goal:

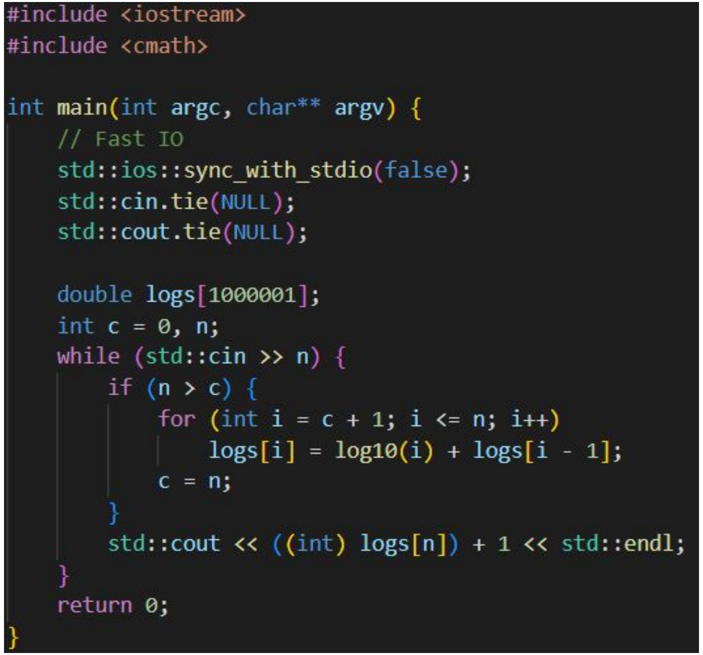
Find the number of digits in the base 10 representation of n!

Notes:

- The number digits in the base b representation of n is floor($\log_b$(n)) + 1
- log(a · b) = log(a) + log(b)
- n! = n · (n - 1)! for n > 0
- log(n!) = log(n · (n - 1)!) = log(n) + log((n - 1)!)

Solution(s):

1) Precompute all values of $\log_{10}$(n!) for 0 ≤ n ≤ 1,000,000 using recurrence above.
2) For each query compute all values of $\log_{10}$(a!) for c ≤ a ≤ n where c is the last value computed then set c := n. This ensures we only calculate values we need without repeat calculations.



```
#include <iostream>
#include <cmath>

int main(int argc, char** argv) {
    // Fast IO
    std::ios::sync_with_stdio(false);
    std::cin.tie(NULL);
    std::cout.tie(NULL);

    double logs[1000001];
    int c = 0, n;
    while (std::cin >> n) {
        if (n > c) {
            for (int i = c + 1; i <= n; i++)
                logs[i] = log10(i) + logs[i - 1];
            c = n;
        }
        std::cout << ((int) logs[n]) + 1 << std::endl;
    }
    return 0;
}
```

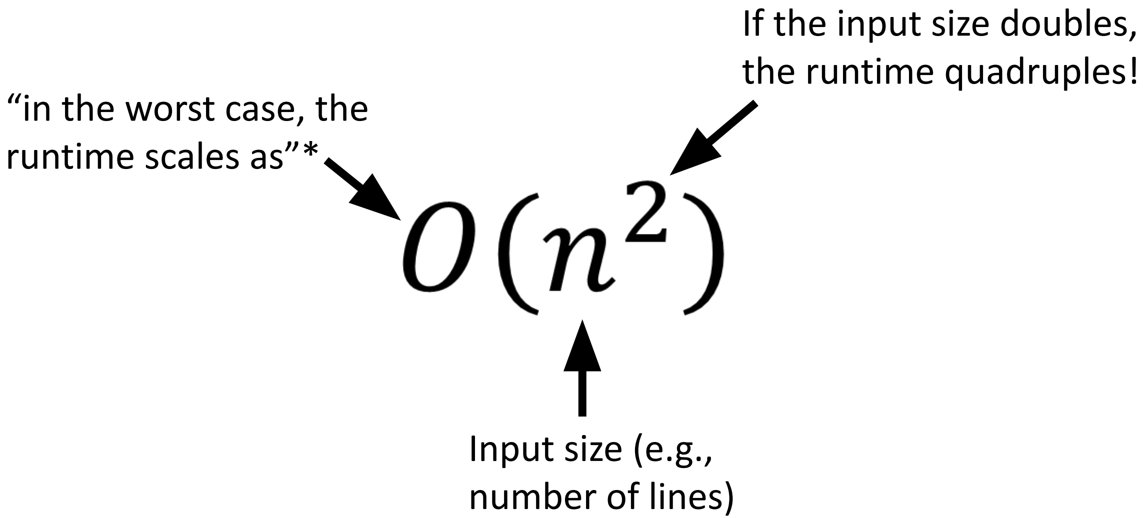
Writing efficient code and predicting runtime

COMPETITIVE
PROGRAMMING
CLUB

Membership sign up:

Time Complexity - “Will this scale?”

- Before coding an algorithm, consider how it will scale with the input size



“in the worst case, the runtime scales as”*

$$O(n^2)$$

If the input size doubles, the runtime quadruples!

Input size (e.g., number of lines)

* Not the formal definition! See https://en.wikipedia.org/wiki/Big_O_notation for technical details

Lists vs Sets

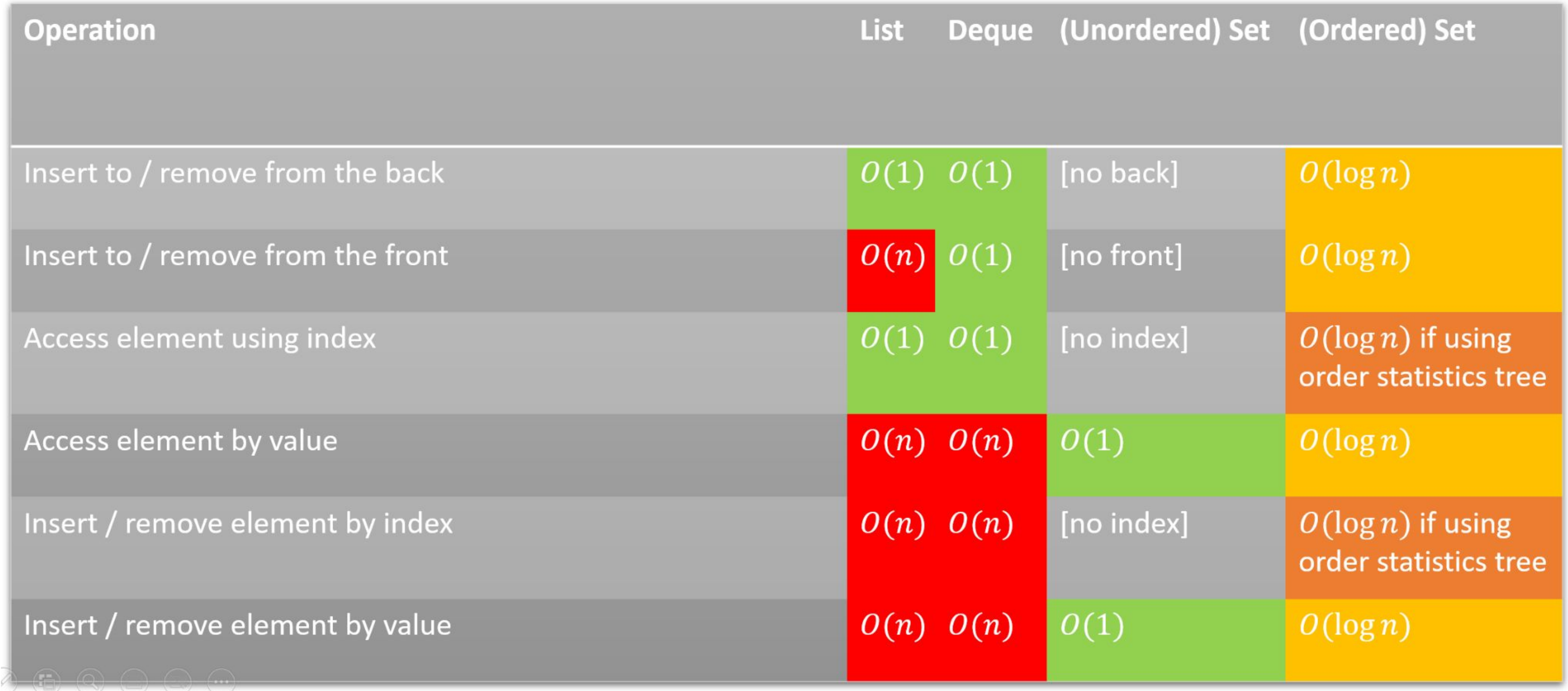
(Resizable) Lists

- `[]` in Python, `vector` in C++
- Keep a sequence of objects in a fixed order
- Inserting to and removing from the end take constant time O(1)
  - "Push" and "pull" from the top when interpreting it as a stack
  - Use a deque (double-ended queue) for fast insertion to and removal from the front
  - Otherwise, O(n) insertions and removals!
- Can also make multidimensional, e.g. `[[0]*N for i in range(M)]` in Python or `vector<vector<int>>` in C++

Sets

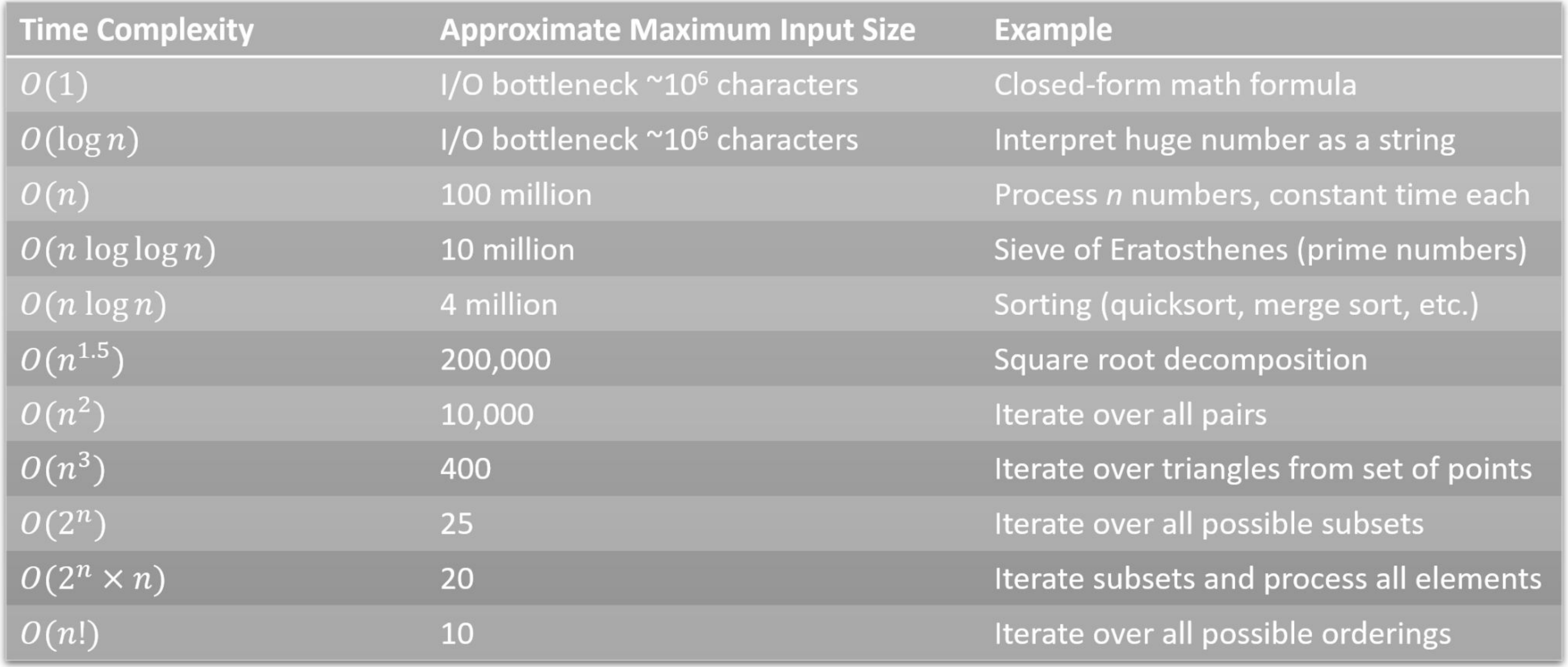
- `set()` in Python, `unordered_set` in C++
- Keep a collection of unique objects in no particular order
- Inserting and removing any element takes constant time O(1)
- Use C++ `set` to access the previous and next element in O(1), at the cost of O(logN) lookups, insertions, and removals
- Can also map keys to values, e.g. dictionaries in Python `{"one": 1, "two": 2}` and `[unordered_]map` in C++

Lists vs Sets



Operation	List	Deque	(Unordered) Set	(Ordered) Set
Insert to / remove from the back	$O(1)$	$O(1)$	[no back]	$O(\log n)$
Insert to / remove from the front	$O(n)$	$O(1)$	[no front]	$O(\log n)$
Access element using index	$O(1)$	$O(1)$	[no index]	$O(\log n)$ if using order statistics tree
Access element by value	$O(n)$	$O(n)$	$O(1)$	$O(\log n)$
Insert / remove element by index	$O(n)$	$O(n)$	[no index]	$O(\log n)$ if using order statistics tree
Insert / remove element by value	$O(n)$	$O(n)$	$O(1)$	$O(\log n)$

Time Complexity Cheat Sheet



Time Complexity	Approximate Maximum Input Size	Example
$O(1)$	I/O bottleneck ~10^6 characters	Closed-form math formula
$O(\log n)$	I/O bottleneck ~10^6 characters	Interpret huge number as a string
$O(n)$	100 million	Process *n* numbers, constant time each
$O(n \log \log n)$	10 million	Sieve of Eratosthenes (prime numbers)
$O(n \log n)$	4 million	Sorting (quicksort, merge sort, etc.)
$O(n^{1.5})$	200,000	Square root decomposition
$O(n^2)$	10,000	Iterate over all pairs
$O(n^3)$	400	Iterate over triangles from set of points
$O(2^n)$	25	Iterate over all possible subsets
$O(2^n \times n)$	20	Iterate subsets and process all elements
$O(n!)$	10	Iterate over all possible orderings

Table adapted from Table 1.4 Competitive Programming 4 by Steven Halim, Felix Halim, and Suhendry Effendy

Input Output Optimization

- For I/O-heavy problems, reading and writing becomes a bottleneck
- For C++, try adding these lines to the beginning of `main`:

```
ios_base::sync_with_stdio(false);
cin.tie(NULL);
```

- For Python, try adding these lines at the beginning:

```
from sys import stdin, stdout
input = stdin.readline
def print(*args):
  stdout.write(' '.join(map(str,args)) + '\n')
```

- Warning: these can cause issues with interactive problems due to not necessarily flushing the output after every line

Reminder - Competitions

- For those thinking about signing up for the Regional competition, this is a reminder that you must also compete in the North America Qualifier competition.
- The Qualifiers will be held this year on September 30th, and the deadline to register is **today.**
- **Sign up here:** https://forms.gle/hSxdW5EzbP8RFgpv5

Alberta Competitive Programming Discord

- https://discord.gg/MSj9Xq4RQb
- Join a larger community of competitive programmers in Alberta
  - Mostly run by UAlberta students and alumni
- Open to all levels of experience

Today’s Contest

https://open.kattis.com/contests/cbyz8w

(or look up **“CPC Fall 2023 Practice Contest Week 2”** in the Kattis contest list)

Feel free to ask questions until 7pm, and then throughout the week on Discord!

COMPETITIVE
PROGRAMMING
CLUB