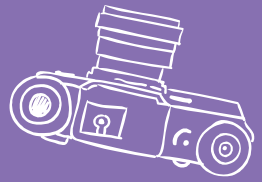
MCSS Presents

Competitive Programming Workshop #1



Square Root Decomposition

Disclaimer

- This is an informal workshop
- You should not use (or need to use) the material here in your courses (*especially* regarding time complexity)

Let's do a fun problem!

Suppose you are given a list L of length n as input.

Given two indices l, r how do you find the sum of everything in between?

In other words, how do you find $L[l] + L[l+1] + \ldots + L[r]$?

Let's make it harder

Suppose you are given a list L of length n as input.

Then you are given m queries that each give two indices and ask you for the sum of everything in between.

How can we answer all queries?

Prefix Sum Array

Idea: for each index i, keep track of $L[0] + L[1] + \ldots + L[i-1]$ in another list (call it *psum*).

Now the sum of everything between l, r is simply $psum[r+1]-psum[l]$

(Be careful with defining $psum[0]$ & $psum[n]$!)

Competitive Programming

That’s pretty efficient...

In competitive programming we care about:

- Time constraint (given in terms of seconds)
- Memory constraint (given in terms of megabytes)

And input is always constrained. For example, the previous problem might have a constraint like:

- 1 <= n <= 10^5
- 1 <= m <= 1000
- -10^9 <= L[i] <= 10^9
- 0 <= l,r < n

Time Complexity

So we should get good at telling whether our solution works for the given constraints, before starting to write up the code!

Rule of thumb: Assume your program can do ~3 * 10^8 operations in a second

WARNING: This is extremely informal, not always accurate, and you should not use this in courses

How to calculate number of operations?

1) We don't care about the *exact* number of operations (an upper bound, shown using big O notation, is good enough)
2) We don't care about constants that much.
    a) Doing $2 * n$ operations is about as fast as doing n or $n/2$ operations for really big values of n
    b) Doing a few calculations here and there (outside loops) will not affect time complexity

You might need to care about these constants if your solution is really close to passing the time constraints.

What is the time complexity of the prefix array solution?

We can fill out *psum* using a single for loop over the list! aka O(n) operations.

```
psum[0] = 0
for i from 1 to n
    psum[i] = psum[i-1] + L[i]
```

(This is called pseudo-code. It's not always convenient to write out whole details of your program while adhering to syntax.)

Then each query can be answered with O(1) since it's just calculating the difference of two *psum*s.

So in total, O(n + n + m) to read the input, calculate psum, and read/answer queries, which is equal to O(n + m).

Memory?

Every competitive programmer should know how to calculate the amount of memory their code needs, but maybe that’s something for another workshop :)

A harder version?

Suppose you are given a list L of length n as input.

Then you are given m queries. Each query can be one of these two types:

1) Gives two indices and asks you for the sum of everything in between
2) Tells you to change the value of $L[i]$ for some i

How can we answer all queries?

Why does our previous method fail?

:(

Updating a prefix sum array is costly enough that we might as well take the naive approach to solve this problem. Both take O(mn) at the end of the day.

Enter Decomposition

Suppose you decomposed your array in the following way:

L[0],..., L[k-1], L[k],..., L[2k-1] , ... , L[tk],...,L[n-1]

Creating buckets of size k.

How many buckets in total?
What is the size of the last bucket?

Buckets are OP

We can keep track of the sum of elements in each bucket!

What is O of doing this before any queries?

Query type (2)

Suppose we are told to change the value of *L[i]* for some *i*.

What do we need to do to maintain the correct bucket sums?

How many operations does this take?

Query type (1)

Suppose we correctly know the sums of elements in each bucket.

Given i,j how can we utilize these sums to our advantage to compute sum of every element between i,j?

What is the O of this?

So in total...

So...

This depends on k! What is the best value of k we can choose?

...AND THAT'S WHY WE CALL IT...

Square Root Decomposition

Let's code it

We'll use Python today.

Most of the competitive programmers I know use C++.

I recommend you learn it. :)

(Spoiler: it makes learning C easier, which is a language you'll be using a lot in upper year CS.)

Another problem

https://codeforces.com/problemset/problem/13/E

Codeforces is a competitive programming website, with tons of practicing problems and contests. You can look at some of the easier problems to get familiar with the format.

Other resources

Competitive programming handbook: https://cses.fi/book/book.pdf

GeeksforGeeks: https://www.geeksforgeeks.org/

MCSS Discord: https://discord.gg/wfJUZyXjfM

Feel free to reach out to me by email/discord if you have any questions about today's workshop!