Sqrt Techniques

Abhiraj, Jason, Raagav

18 May 2025

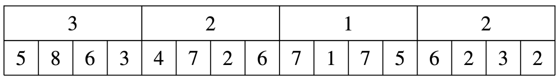
§1 Introduction

Sqrt techniques appear in USACO Gold and the 1900+ rating range on Codeforces, and can be far easier to implement and think about than alternatives.

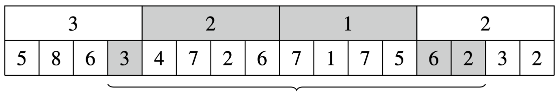
§2 Sqrt Decomposition

Sqrt decomposition is the process of separating a structure of size n into $\sqrt{n}$ blocks of size $\sqrt{n}$, maintaining something about each block.

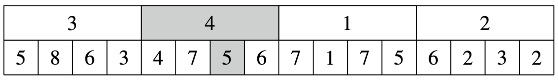
Suppose we want to be able to find the min value in a range and modify array values. Then we maintain for each block the min value inside it:



To find the minimum value in a range, we divide it into three parts such that the range consists of single values and the blocks between them, and then take the minimum of these. Because the number of single elements is $O(\sqrt{n})$ and the number of blocks is also $O(\sqrt{n})$, queries are answered in $O(\sqrt{n})$.

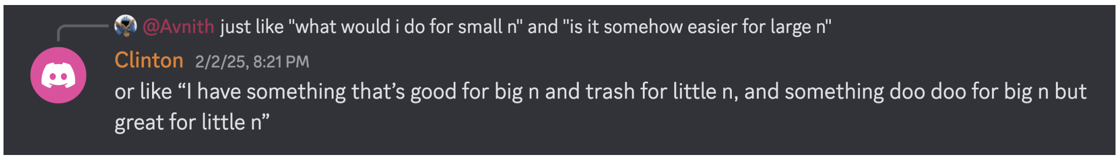


When an array value changes, we update the corresponding block by going through the values inside the block and finding the new minimum. This is done in $O(\sqrt{n})$ time.



You can extend this to answer max, sum, GCD, and even k-th order statistics.

§3 Subalgorithms & Sqrt Trick


@Avnith just like "what would i do for small n" and "is it somehow easier for large n"

Clinton 2/2/25, 8:21 PM

or like “I have something that’s good for big n and trash for little n, and something doo doo for big n but great for little n”

To illustrate what the above means, we will use Codeforces Round 920, F. Sum of Progression:

You are given an array A of n numbers. There are also q queries of the form (s, d, k). For each query, compute

$$1 \cdot a_s + 2 \cdot a_{s+d} + 3 \cdot a_{s+2d} + \cdots + k \cdot a_{s+(k-1)d} \cdot$$

In other words, for each query you must sum k elements of the array, starting at index s, stepping by d, and multiplying each term by its position in that subsequence.

The static array sum queries should remind us of prefix sums. Notice that, for each step size d, we can answer all queries with that d in $O(1)$ after constructing prefix sums arrays in $O(n)$ (think about remainder classes).

Because prefix sums are not sufficient, consider the naive solution of just summing the queried elements. For any d, this will take at most $\frac{n}{d}$ operations. Obviously, the naive solution is good for big d, but bad for small d. So, for small $d \leq \sqrt{n}$, we can use prefix sums, and for big d, we can use the naive solution. This gives us a time complexity of $O(n\sqrt{n} + Q\sqrt{n})$ because we do our precomputations in $O(n\sqrt{n})$ and answer queries in $O(1)$ or $O(\sqrt{n})$.

This combination of two algorithms is called a subalgorithm.

§4 Integer Partitions

If

$$A_1 + A_2 + \cdots + A_{k-1} + A_k = n$$

there are at most $O(\sqrt{n})$ distinct values (think about the sum of consecutive numbers). To demonstrate this technique, we will solve a classical DP problem.

You are given a list of integer weights $[w_1, w_2, \ldots, w_k]$ such that $w_1 + w_2 + \cdots + w_k = n$. Find all possible weight sums that can be created.

We can solve this problem easily in $O(nk)$ by iterating through w, and storing a $DP[i] =$ whether i is a possible sum. To be more efficient, we can process equal w_i together, which introduces $O(\sqrt{n})$. For each group, we can edit DP in $O(n)$ in several simple ways (think about remainder classes), resulting in a time complexity of $O(n\sqrt{n})$.

§5 Problems

1. USACO 2024 December Contest, Gold Problem 1. Cowdependence
2. USACO 2025 US Open Contest, Gold Problem 2. Election Queries
3. Codeforces Round 136 (Div. 1) B. Little Elephant and Array

4. Codeforces Round 199 (Div. 2) E. Xenia and Tree
5. Codeforces Round 233 (Div. 1) D. Instant Messanger
6. Codeforces Round 254 (Div. 1) C. DZY Loves Colors
7. Codeforces Round 340 (Div. 2) E. XOR and Favorite Number