

Sqrt Techniques

Raagav Bala, Abhiraj Mallangi, Jason Zhang



The background is a solid teal color. In the four corners, there are decorative geometric shapes made of overlapping hexagons. The top-left and bottom-left corners feature a cluster of three hexagons in shades of light blue and teal. The top-right and bottom-right corners feature a cluster of three hexagons in shades of light blue and teal, with some hexagons having white outlines.

Sqrt Decomposition

Square Root Decomposition

The idea is to divide a structure of size n into approximately \sqrt{n} blocks of approximately size \sqrt{n} while still maintaining some information about the overall structure.

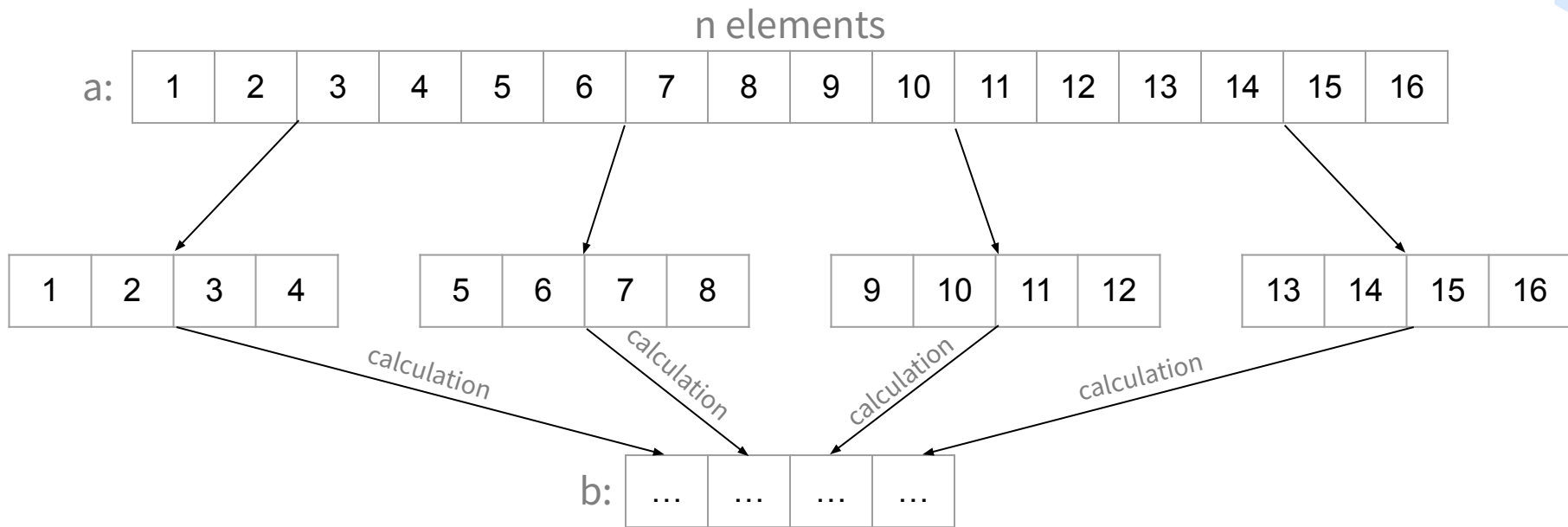
This usually allows us to answer queries very quickly since we will only have to work with \sqrt{n} items of data.

It can be shown that \sqrt{n} is best possible division/decomposition of data such that each block is representative of its elements (for a range query problem).

Proof of Claim

Proof. Suppose we divided into a certain amount of blocks based on a function $f(n)$. If we were given a problem that involved ranges of the form $0 \leq l \leq r < n$ then we would have to check $f(n)$ blocks with $\frac{n}{f(n)}$ additional singular elements. This is equivalent to $O(f(n) + \frac{n}{f(n)}) = O(\max(f(n), \frac{n}{f(n)}))$. If $f(n) > \sqrt{n}$, then obviously it is worse. If $f(n) < \sqrt{n}$, then $\frac{n}{f(n)} > \frac{n}{\sqrt{n}} = \sqrt{n}$. So \sqrt{n} is the optimal choice for f . \square

Visual Representation



We can see how the original array (a) was split into \sqrt{n} blocks each with \sqrt{n} data. We then run some sort of calculation (min, max, sum, etc.) so that we can represent each block with a singular value, which goes into a new array b.

Finding Minimum

For demonstration purposes, here is a problem where sqrt decomposition is efficient.

Part 1: You are given an array $a[0..n-1]$ and q queries. In each query, you are given l, r ($0 \leq l < r < n$) and you must find the minimum of the subarray $a[l..r]$ efficiently.

After you think you have the solution for the above, consider this extra challenge (part 2): after each query, exactly one index i ($0 \leq i < n$) will have its value changed (the change persists).

Solution

We will simply use the idea of sqrt decomposition and represent each block with the minimum of its elements. Upon receiving l and r , we can quickly identify which blocks are included within the range. It is very likely that there will be blocks that do not have every element included, so we will also have to add all of the elements of the “partial blocks” in. However, there can be at most \sqrt{n} blocks and \sqrt{n} partial block elements, so it is still $O(\sqrt{n})$. A visual example will be provided on the next slide.

As for updating values, we simply need to identify which block is being updated and recalculate the minimum (if necessary). This can be done in $O(\sqrt{n})$ worst case.

Visual Solution

The grey are the affected blocks within the range $[l,r]$.

3				2				1				2			
5	8	6	3	4	7	2	6	7	1	7	5	6	2	3	2

[l,r]

Implementation

A full implementation is too long to include here. Please see [this link](https://tinyurl.com/5xbzmuwm) (<https://tinyurl.com/5xbzmuwm>) to find an implementation for the first part of the problem.

If you want to implement this yourself, we recommend using $\lfloor \sqrt{n} \rfloor + 1$ for both the block size and amount of blocks. There may be an extra block that doesn't do anything, but it doesn't matter.

Linked Code (if unable to view link)

This is the same code linked on the implementation slide without comments.

Try to see the full code for a better explanation

```
1  n = int(input())
2  arr = [int(i) for i in input().split(" ")]
3  blocks = int((n ** 0.5)) + 1
4  sqrt_arr = [0 for i in range(blocks)]
5  ind = -1
6  for i in range(n):
7      if i % blocks == 0:
8          ind += 1
9          sqrt_arr[ind] = arr[i]
10         sqrt_arr[ind] = min(arr[i], sqrt_arr[ind])
11  q = int(input())
12  for _ in range(q):
13      l, r = [int(i) for i in input().split(" ")]
14      out = arr[l]
15      while l <= r:
16          if (l % blocks == 0 and l + blocks <= r):
17              out = min(out, sqrt_arr[int(l / blocks)])
18              l += blocks
19          else:
20              out = min(out, arr[l])
21              l += 1
22  print(out)
```

Part 2 Solution

If we receive queries of the form

ind val

then to maintain the properties of the array we use the following code. Better implementation may be found on online resources (see the end slide).

```
ind, val = [int(i) for i in input().split(" ")]
if sqrt_arr[int(ind / blocks)] == arr[ind]:
    arr[ind] = val
    if arr[ind] <= sqrt_arr[int(ind / blocks)]:
        sqrt_arr[int(ind / blocks)] = val
    else:
        for i in range(int(ind / blocks) * blocks, min(int(ind / blocks) * (blocks) + blocks, n)):
            sqrt_arr[int(i / blocks)] = min(sqrt_arr[int(i / blocks)], arr[i])
else: # the value updated is not the min
    arr[ind] = val
    if arr[ind] < sqrt_arr[int(ind / blocks)]:
        sqrt_arr[int(ind / blocks)] = val
```

Part 1 Pseudo-Code

In case you cannot read Python,
here is a pseudo-code
implementation.


Data: See problem statement (part 1).

Result: Correct answers to queries.

```
 $n \leftarrow \text{input};$ 
 $\text{arr} \leftarrow \text{input};$ 
 $\text{blocks} \leftarrow \lfloor \sqrt{n} \rfloor + 1;$ 
 $\text{sqrtArr} \leftarrow \text{empty array of length blocks};$ 
 $\text{ind} \leftarrow -1;$ 
for  $i \leftarrow 0$  to  $n - 1$  do
    if  $i \bmod \text{blocks} = 0$  then
         $\text{ind} \leftarrow \text{ind} + 1;$ 
         $\text{sqrtArr}[\text{ind}] \leftarrow \text{arr}[i];$ 
    end
     $\text{sqrtArr}[\text{ind}] \leftarrow \min(\text{sqrtArr}[\text{ind}], \text{arr}[i]);$ 
end
 $q \leftarrow \text{input};$ 
for  $t \leftarrow 0$  to  $q - 1$  do
     $l, r \leftarrow \text{input};$ 
     $\text{out} \leftarrow \text{arr}[l];$ 
    while  $l \leq r$  do
        if  $l \bmod \text{blocks} = 0$  and  $l + \text{blocks} \leq r$  then
             $\text{out} \leftarrow \min(\text{out}, \text{sqrtArr}[\lfloor \frac{l}{\text{blocks}} \rfloor]);$ 
             $l \leftarrow l + \text{blocks};$ 
        else
             $\text{out} \leftarrow \min(\text{out}, \text{arr}[l]);$ 
             $l \leftarrow l + 1;$ 
        end
    end
end
```

Subalgorithms

Clinton's Definition of Subalgorithms

 @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 "

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

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

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.

Naive Solution?

Share!



Prefix Sums?

Share!3



Integer Partitions

Intuition

If $A_1 + A_2 + \dots + A_k = n$, then there are at most $O(\sqrt{n})$ distinct values among all the A_i values.

The proof of this is that to maximize the number of distinct values, we would need each A_i to be distinct from each other (so k values). We would also want our A_i values to be as small as possible (if every number is very large, then we can split some of those numbers). Therefore, let us use $1+2+3+\dots+k=k(k+1)/2$. This sum is less than or equal to n , and through some basic algebra we conclude that k is of $O(\sqrt{n})$.

Weights Problem

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



Solution

We can solve this problem easily in $O(nk)$ by iterating through w , and using dp with $DP[i]$ = whether i is a possible sum or not. To be more efficient, we can process equal w_i together, which introduces a $O(\sqrt{n})$. For each group, we can edit DP in $O(n)$ in several simple ways (think about remainder classes), resulting in a time complexity of $O(n\sqrt{n})$.

Additional Problems and Practice

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 Messenger
6. Codeforces Round 254 (Div. 1) C. DZY Loves Colors
7. Codeforces Round 340 (Div. 2) E. XOR and Favorite Number