MO’s Algorithm (Query Square Root Decomposition) | Set 1 (Introduction)

Let us consider the following problem to understand MO’s Algorithm.

We are given an array and a set of query ranges, we are required to find the sum of every query range.

Example:

Input: arr[] = {1, 1, 2, 1, 3, 4, 5, 2, 8};

query[] = [0, 4], [1, 3] [2, 4]

Output: Sum of arr[] elements in range [0, 4] is 8

Sum of arr[] elements in range [1, 3] is 4

Sum of arr[] elements in range [2, 4] is 6

The idea of **MO’s algorithm** is to pre-process all queries so that result of one query can be used in next query. Below are steps.

Let **a[0…n-1]** be input array and **q[0..m-1]** be array of queries.

1. Sort all queries in a way that queries with L values from **0** to **√n – 1** are put together, then all queries from **√n** to **2\*√n – 1**, and so on. All queries within a block are sorted in increasing order of R values.
2. Process all queries one by one in a way that every query uses sum computed in the previous query.
   * Let ‘sum’ be sum of previous query.
   * Remove extra elements of previous query. For example if previous query is [0, 8] and current query is [3, 9], then we subtract a[0],a[1] and a[2] from sum
   * Add new elements of current query. In the same example as above, we add a[9] to sum.

The great thing about this algorithm is, in step 2, index variable for R change at most **O(n \* √n)** times throughout the run and same for L changes its value at most **O(m \* √n)** times (See below, after the code, for details). All these bounds are possible only because the queries are sorted first in blocks of **√n** size.

The preprocessing part takes O(m Log m) time.

Processing all queries takes **O(n \* √n)** + **O(m \* √n)** = **O((m+n) \* √n)**time.