Problem A:
Compress coordinates. The rest can be done with difference arrays.

Problem C:
Scan left to right, Maintain a set of lengths of currently active segments. This can easily be done by adding them at their left endpoint and removing them at their right. Then the answer at an index can be found by taking the minimum element from the set.

Problem D:
For each left index l, calculate the max right index r such sum(l, …. r) <= M. This can be easily done with two pointers or binary search with prefix sums. Then take the maximum sum over all possible left indexes.

Problem F:
In this problem you have to find the max difference between two elements which are not more the d distance apart. We consider each index i as the smaller element, then the larger element is the maximum of the range (i-d, i+d). We can find this maximum with RMQ/sliding windows. Then we take the maximum over all possible i.

Problem G:
Straightforward Range Update, Range Query problem. The only caveat is that the segments are circular. We can circumvent this by dividing a circular segment (l, r) (where l > r) into two segments (l, n) and (1, r).

Problem H:
Discussed in class. We need to keep two variables in each node of the segtree, **sum** and **sqsum,** denoting resp. the sum of the numbers and the sum of squares of the numbers in the corresponding range. Combining these values are trivial. Then evaluating lazy can be done as follows -

```
Node.sum = left.sum + right.sum
Node.sqsum = left.sqsum + right.sqsum + 2*lz[u] * Node.sum + length*lz[u]*lz[u]
```

More Details: https://codeforces.com/blog/entry/47909

Problem K:

The operations described cannot be directly implemented by a segment tree. Instead we have to consider each bit's positions separately. We calculate the contribution for each bit and sum them. To solve for each bit, we need only know the number of 1 bit and 0 bit in a range, and the update is to change an index. It can be easily done with a OrderedSet / Segment Tree. Complexity is O(log^2) per query,

Problem L:

Discussed in class.

Details:https://www.quora.com/Competitive-Programming-How-do-I-solve-Light-OJ-1188-Fast-Queries-using-segment-tree

Problem M:

Discussed in class.

We process the queries in increasing order. To answer a query with third parameter c. we add all indexes with values less than or equal to c in a ordered set, then the answer is the number of integers in the ordered set in the range (a, b). Since we are processing in increasing order, we need only add each index at most once.

Problem P:

https://codeforces.com/blog/entry/20377. See the first problem.