**Kick Start 2018 - Round G**

# Analysis: Product Triplets

## Product Triplets: Analysis

### Small dataset

The Small dataset can be solved by complete search. We can check for each possible triplet (x, y, z) (with $1 \leq x < y < z \leq \mathbf{N}$) and check whether at least either one of $\mathbf{A}_x$, $\mathbf{A}_y$, or $\mathbf{A}_z$ is the product of the other two. Each time we get a valid triplet, we increment our answer variable by one.

Since $\mathbf{N} \leq 200$ for this dataset, an $O(\mathbf{N}^3)$ time solution will run within the time limit.

### Large dataset

Let us ignore indices i where $\mathbf{A}_i = 0$ for the time being. In other words, let us assume that $\mathbf{A}_i > 0$ for all i.

We can observe that for any three positive integers a, b, and c, if $a \times b = c$, then $a \leq c$ and $b \leq c$. Therefore, if we sort **A** in nondecreasing order, for each triplet (x, y, z) (with $1 \leq x < y < z \leq \mathbf{N}$), we only need to check whether $\mathbf{A}_x \times \mathbf{A}_y = \mathbf{A}_z$. However, doing this naively will still be an $O(\mathbf{N}^3)$ time solution.

We can optimize this solution by computing the number of z satisfying y < z and $\mathbf{A}_x \times \mathbf{A}_y = \mathbf{A}_z$ for each pair (x, y) in constant time. To do this, we can store the number of occurrences of each number in **A** in a hashmap.

If we design our nested loop such that we have y = {**N** .. 1} as our outer loop (and x = {y - 1 .. 1} as our inner loop), then we can update our hashmap for $\mathbf{A}_y$ just before we decrement the value of y. This is to make sure that our hashmap only contains the elements with indices larger than the current value of y. In other words, the algorithm looks something like:

```
sort(A)
ans = 0
occurrences = hashmap()
for y : {N .. 1}
  for x : {y - 1 .. 1}
    ans = ans + occurrences.count(A[x] * A[y])
  occurrences.update(A[y], occurrences.get(A[y]) + 1)
```

We should not forget that our algorithm so far ignores those indices i where $\mathbf{A}_i = 0$. Suppose there are Z such indices (i.e., there are Z zeroes in **A**). We can make C(Z, 3) triplets using three zeroes, and C(Z, 2) × (**N** - Z) triplets using two zeroes and one non-zero.

Therefore, we get our final answer by summing the value of `ans` from our pseudocode, as well as C(Z, 3) and C(Z, 2) × (**N** - Z). This solution runs in $O(\mathbf{N}^2)$ time.