Meetings

Let's denote by $T(n)$ the time needed for n participants to reach a decision. First we observe that $T(n)$ never decreases as n increases. From this follows that when we divide the n participants into i groups, we want to make the largest group as small as possible. Thus, it is optimal to let most groups have size $\lceil n/i \rceil$ and the last one may be smaller. This gives us a simple dynamic programming solution with $O(N)$ memory and $O(N^2)$ time that would be sufficient to get 40 points:

```
a[1] = 0;  // special case: no meeting
for (int i = 2; i <= n; ++i) {
  a[i] = i * p + v; // baseline: no groups
  for (int j = 1; j < i; ++j) {
    // j groups: the size of groups is i/j rounded up
    int k = (i - 1) / j + 1;
    int t = a[k] + a[j];
    if (a[i] > t)
      a[i] = t;
  }
}
```

Next we can observe that the inner loop in the above solution basically finds the value of $T(n)$ as

$$T(n) = \min_{a \cdot b \geq n} \{T(a) + T(b)\}. \tag{1}$$

Since this is symmetric, we only need to consider the pairs (a, b) where $a \leq b$, or $a \leq \sqrt{n}$. Thus we can get a solution with $O(N\sqrt{N})$ running time and earn 70 points just by replacing the line

```
for (int j = 1; j < i; ++j)
```

with the line

```
for (int j = 1; j * j < i; ++j)
```

To model dividing the groups into sub-groups, we can recurrently express $T(a)$ and $T(b)$ in (1), and eventually arrive at the general form

$$T(n) = \min_{n_1 \cdot n_2 \cdot \ldots \cdot n_k \geq n} \{T(n_1) + T(n_2) + \ldots + T(n_k)\}. \tag{2}$$

Without further sub-grouping $T(n_i) = n_i \cdot P + V$ and thus the total working time corresponding to the grouping in (2) can be expressed as

$$T(n, k) = \sum (n_i \cdot P + V) = (\sum n_i) \cdot P + k \cdot V. \tag{3}$$

Obviously $k \cdot V$ in (3) depends only on k, but not on the choice of values of n_i. Thus, to minimize the value of $T(n)$ for a fixed k, we need to choose n_i so that $\sum n_i$ would be minimal.

Since the sum of factors multiplying to a given product is minimal when the factors are equal, we obtain the optimal value for $T(n)$ when n_i are as close to $\sqrt[k]{n}$ as possible. More precisely, we need to use $\lfloor \sqrt[k]{n} \rfloor$ for as many and $\lceil \sqrt[k]{n} \rceil$ for as few n_i as possible so that $\prod n_i$ would still be at least n.

Since each n_i must be at least 2 (there's no benefit in creating sub-groups with just one member), we only need to consider the values of k up to $\log_2 N$. From the preceding, we can easily compute $T(n, k)$ using just $O(k^2)$ multiplications, which gives us a solution with $O(1)$ memory and $O(\log^3 N)$ time that would get the full score:

```
// computes T(n, k) for fixed k
long long solve_k(long long n, int p, int v, int k) {
  long long fact = (long long) pow(n, 1.0 / k);
  // since fact was rounded down above, we now increase
  // some factors by 1 to have them multiply to at least n
  int incr = 0;
  while (power(fact + 1, incr) * power(fact, k - incr) < n)
    ++incr;
  // the answer is \sum(factors)*p + k*v
  return (k * fact + incr) * p + k * v;
}

// computes T(n)
long long(solve(long long n, int p, int v) {
  if (n == 1)
    return 0;
  long long result = solve_k(n, p, v, 1);
  for (int k = 2; 2ll << k <= n; k++) {
    long long r = solve_k(n, p, v, k);
    if (result > r)
      result = r;
  }
  return result;
}
```

Looking at the task text, there might be some doubt whether the group leaders are required to hold a single meeting or may also form sub-groups among themselves. In other words, are we allowed to use $T(a) + T(b)$ as the right-hand side of (1) or should we be restricted to $T(a) + b \cdot P + V$ instead? It turns out this does not matter, as any process involving the b group leaders forming b' sub-groups could also be modeled as forming b' groups first and these splitting into a total of b sub-groups.

Task idea by Konstantin Tretyakov, spoiler by Ahto Truu, with hints from Thomas Fersch and Ludwig Schmidt.