Plagiarism

An obvious solution for this task is to explicitly check all pairs of files and count the ones whose sizes are close enough to warrant more detailed comparison. A bit of care is needed to avoid counting twice the pairs where the sizes are exactly equal. A possible implementation that runs in $O(N^2)$ time and would be good enough to score 50 points:

```
long long k = 0;
for (int i = 0; i < n; ++i)
  for (int j = 0; j < i; ++j)
    if (10 * min(f[i], f[j]) >= 9 * max(f[i] , f[j]))
      ++k;
```

Also note that the replacement of floating-point arithmetic with integer multiplications to prevent rounding errors might introduce overflows if the file sizes would be allowed to be closer to `MAXINT`.

If we would first sort the files by size, we could for each potential larger file compute the smallest possible size of the smaller file and use binary search to efficiently count the number of sufficiently large preceding files in the sorted list for an $O(N \log N)$ solution that would get the full score:

```
long long k = 0;
for (int i = 0; i < n; ++i) {
  // smallest possible size of smaller file: 0.9*f[i], rounded up
  int g = (9 * f[i] - 1) / 10 + 1;
  int *p = lower_bound(f, f + i, g);
  // the distance from *p to f[i]
  k += f + i - p;
}
```

With the files sorted, we could count the answer even in $O(N)$ time using linear search. If we would start the search for the smaller file matching f_i from the position of the match found for f_{i-1} on previous iteration, these incremental searches would accumulate just one scan over the sorted list:

```
long long k = 0;
for (int i = 0, j = 0; i < n; ++i) {
  while (10 * f[j] < 9 * f[i])
    ++j;
  k += i - j;
}
```

Of course, since the sorting already takes $O(N \log N)$ time, the overall complexity function of the third solution is really not better than the second one.

Task idea by Konstantin Tretyakov, spoiler by Ahto Truu.