Exercise 1

Let's call the proximity score as $\text{PSCORE}(q, d)$, where q is the query, and d is the document. Now I will present 3 alternatives that I can come up with in order to calculate this score. Method 1 is the simplest method (this doesn't mean that it's a naïve method), because it's easy to implement and the algorithm can be quite fast. However, method 1 is less accurate compared to the other two. Method 2 and 3 are almost the same in terms of the techniques that are used and in terms of time complexity, but method 3 is a bit more accurate.

Method 1:

We measure the proximity using the smallest window (w) in the document d that contains all the terms in the query q. The smaller the window, the closer the query terms are to each other. So the higher $1/w$, the higher rank we get. If the document doesn't have all the query terms, then we let w to be some enormous number. So in the most simplistic form, we can let $\text{PSCORE}(q, d) = 1/w$. Of course, we can also use some other variants, like logarithmic scale, for instance.

Advantage of this method is that it is quite straightforward to implement (though we might need lots of optimizations in the algorithm in order to find the smallest window efficiently). Disadvantage is that it is sensitive to extreme cases. Suppose that we have 10 query terms, in which 9 of them stand next to each other at the start of the document, and the last term is at the end of the document. Then the smallest window would be the entire length of the document, which isn't fair. Also, this method doesn't take into account the spread of the query terms in the windows (because we only use the length of the window as the measurement of the proximity).

Method 2:

For each pair of query terms, we find the smallest distance between them in the document. Then we use the average distance for all the pairs as a measurement of proximity. Let $D(q_a, q_b)$ be the shortest distance between the 2 query terms q_a and q_b. Suppose that we have the query $q = q_1 \dots q_n$, then mathematically, the score approximately would be:

$$\text{PSCORE}(q, d) = \text{average}\left(\sum_{i,j}^{n} D(q_i, q_j)\right) \quad \text{for all } j > i \geq 1$$

The advantage of this method is that it takes into account the spread of all the query terms in the document, which means that the proximity score is more accurate compared to the first method. The disadvantage is that it is also very sensitive to the extreme cases. Like before, if the query has 10 terms, in which 9 of them are next to each other, while the last term is far away from the rest, then we still end up with the same problem as in method 1. However, we can expect the effects to be less extreme here (because for method 2, we are taking the average distance, rather than the absolute distance of the window as in method 1).

Method 3:

This is actually a variant of the method 2 above. It's similar in the sense that we are still going to measure the proximity scores for each and every pair, and then use them to compute the average score as the final proximity score for the document. However, the difference is that the proximity for a single pair will not be the shortest distance between the two terms in the document. Rather, we are going to transform the two query terms into 2 vectors of the same dimension (the dimension will be equal to the length of the document), and then compute the cosine similarity (which is also the inner product) between these 2 vectors. This idea is actually taken from an answer here on StackOverflow, which converts query terms into vectors using the "pyramid" technique (see here: http://stackoverflow.com/questions/19034271/computing-the-dot-product-for-calculating-proximity). As it has already been explained very well on StackOverflow, I will not attempt to explain it here.

The advantage of this method is that it's more accurate than method 2, because rather than measuring the distance, we measure the angle between 2 vectors (which is less sensitive to extreme cases). The disadvantage is that it's arguably more complicated than both of the previous methods. Besides, it's not really that fast. The problem is that for a pair of query terms, we have to convert both lists of positional indexes into 2 separate vectors (which will take approximately $O(r)$ time, where r is the length of the document). And then we also have to do this for all the pairs, and there are exactly $n(n-1)/2$ such pairs, where n is the number of query terms. So in total, the time complexity is $O(r \times n^2)$. This, in fact, has the same time complexity as method 2, but here we have to do some extra work to calculate the dot product between the 2 vectors.