A. www.cs.purdue.edu ~734 documents
    cs.illinois.edu ~1150 documents

Estimating source size: Sample k docs from each, say there are m_1 (out of k) docs in purdue website which contain the phrase 'information retrieval' and there are m_2 (out of k) docs in illinois website which contain phrase 'information retrieval'

Size(purdue source) = \( \frac{k}{m_1} \times 734 \)
Size(illinois source) = \( \frac{k}{m_2} \times 1150 \)

Say k = 10, m_1 = 1, m_2 = 2,
S1 = 7340, S2 = 5750

Scoring based on CORI formula: \( b + (1 - b) \times T \times I \) (b = 0.4)
(Refer to page 37 on 'Federated Search' additional reading for full explanation)
Assuming an average document length of w words

T(purdue_source) = 734 / (734 + 50 + (150 * 1.122)) = 0.771
T(illinois_source) = 1150 / (1150 + 50 + (150 * 0.879)) = 0.863

I = \( \log(2.5 / 2) / \log(3) \) = 0.203

Score(purdue) = 0.4 +0.6 * 0.771 * 0.203 = 0.493
Score(illinois) = 0.4 + 0.6 * 0.863 * 0.203 = 0.505

Illinois website is a better source than purdue for this query.

Actual numbers might vary depending on several variables and assumptions. This one possible solution.

You may also score the source using language model:
P(purdue_source) = S1/(S1+S2) = 7340 / (7340+5750) = 0.5607
P(illinois_source) = S2/(S1+S2) = 5750 / (7340+5750) = 0.4393

Assume Lambda = 0.5

P(Q | purdue_source) = lambda*(m_1/k) + lambda*((m_1+m_2/(2k)) = 0.5*(1/10) + 0.5*(3/20) = 0.125
P(purdue_source | Q) = P(Q | purdue_source) * P(purdue_source) = 0.125 * 0.5607 = 0.070

P(Q | illinois_source) = lambda*(m_1/k) + lambda*((m_1+m_2/(2k)) = 0.5*(2/10) + 0.5*(3/20) =0.175
P(illinois_source | Q) = P(Q | illinois_source) * P(illinois_source) = 0.175*0.4393 = 0.077

So Illinois site is the better site according to the Big Document approach because Illinois had a larger probability.

**Estimating source sizes correctly: 5 pts**
**Calculating CORI scores (or probability using language model) for each website: 5 pts**
**Identifying which is a better website and evaluating qualitatively: 5 pts**
(Total capped to 10 pts)
B. You may either assume that $C'$ is given or $C$ is given.

Document scores for top-10 docs (for both sources):
1, 0.5, 0.33, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.1
Or (above scores * 10) - both yield same answer

Assuming $C$ is given, you may calculate $C'$ in the following way:
$C' = (C - C_{\text{min}}) / (C_{\text{max}} - C_{\text{min}})$

$I = 0.203$ (Calculation similar to part A)
$C_{\text{min}} = 0.4 + 0.4 * 0 * 0.203 = 0.4$
$C_{\text{max}} = 0.4 + 0.4 * 1 * 0.203 = 0.4812$

It is more reasonable to assume $C'$ is given in this case. (2 pts)
If assumed $C$ is given correct calculation of $C'$ (1 pt)

Assuming $C'$ is given, $C'(\text{Umass}) = 0.8$, $C'(\text{OSU}) = 0.3$
$D = (D + 0.4 * D * C') / 1.4$  (Correct formula 3 pts)
For equally weighted doc and collection scores: $D = (D + D * C') / 2$  (Equal weighting 1 pt)

Equally weighted:
Scores for Umass: $0.9 * \text{doc\_score}$
Scores for OSU: $0.65 * \text{doc\_score}$

Rank the documents based on these scores

Comming up with correct doc scores - 3 pts
Calculation CORI scores correctly - in any one of the ways - 6 pts
Merging results based on new scores - 3 pts

(Total capped to 10 pts)
C. (Sample answer)

Site1: flight.google.com
Site2: expedia.com

(1) Source Selection: Both site provides REST interface to send the query on their root webpage. Therefore, we can use the get() request instead of filling out the rendered table and clicking the search button.

(2) Determining what part of a query to put in which field: In both case, “city1” and “city2” with departing and returning dates. There are “number of peoples” and other advanced options like “nonstop” and “refundable flight”, but they are optional.

(3) Mutability: They would not affect the underlying data because they are based on the availability of seats. However, the optimization department at each airline could modify the price based on the demand, so the search activity itself could affect the price of tickets.

(4) Result interpretation: Yes. The result is well standardized according to the electronic business standards, so the results from both sites could be merged and sorted easily based on user's preference.

No selection of two websites: -2 points
Misunderstanding or none of discussion at each part: -2 x (# of parts)

D. Credit to Max Bodoia at Stanford:

Recall that each iteration of standard k-means can be divided into two phases, the first of which computes the sets Si of points closest to mean µi , and the second of which computes new means as the centroids of these sets. These two phases correspond to the Map and Reduce phases of our MapReduce algorithm. The Map phase operates on each point x in the dataset. For a given x, we compute the squared distance between x and each mean and find the mean µi which minimizes this distance. We then emit a key-value pair with this mean's index i as key and the value (x, 1). So our function is

\[
\text{k-meansMap}(x) : \\
\text{emit} \left( \arg\min_i \|x - \mu_i\|^2, (x, 1) \right)
\]

The Reduce phase is just a straightforward pairwise summation over the values for each key. That is, given two value pairs for a particular key, we combine them by adding each corresponding element in the pairs. So our function is:

\[
\text{k-meansReduce}(i, [(x, s), (y, t)]): \\
\text{return} (i, (x + y, s + t))
\]

The MapReduce characterized by these two functions produces a set of k values of the form 

\[
\left( i, \left( \sum_{x \in S_i} x, |S_i| \right) \right)
\]

where Si denotes the set of points closest to mean µi . We can then compute the new means (the centroids of the sets Si) as
\[ \mu_i \leftarrow \frac{1}{|S_i|} \sum_{x \in S_i} x \]

Note that in order for the Map function to compute the distance between a point \( x \) and each of the means, each machine in our distributed cluster must have the current set of means. We must therefore broadcast the new means across the cluster at the end of each iteration. We can write the entire algorithm (henceforth referred to as K-MEANS) as follows:

---

**K-MEANS**

1. Choose \( k \) initial means \( \mu_1, \ldots, \mu_k \) uniformly at random from the set \( X \).
2. Apply the MapReduce given by \( k\text{-meansMap} \) and \( k\text{-meansReduce} \) to \( X \).
3. Compute the new means \( \mu_1, \ldots, \mu_k \) from the results of the MapReduce.
4. Broadcast the new means to each machine on the cluster.
5. Repeat steps 2 through 4 until the means have converged.

---

**Map function:** 5 pts

**Reduce function:** 5 pts

**Explanation:** 5 pts

(Total capped to 10 pts)