require 900,000 fewer operations in order to read all the records! It is not uncommon to have disks with an average access time of 30 milliseconds. If this were the case, for the first file it would take about 8 hours longer to read all the records in the file.

This example illustrates why information on the disks is stored in blocks. Moreover, it is clear that the more records in the block, the fewer accesses to the disk are necessary, which speeds up the operations with the file. It should be noted that the number of records that can fit into a block is called the *blocking factor.* The blocking factor depends on both the size of the record and the physical size of the block. Both of these vary within certain limits. Blocks are limited in size because the computer system must allocate primary storage space to store at least one block. This space is called a *buffer.* As primary storage is essential for computer system operation in general, there is usually a specification of the maximum size (different for various computers) allowed for buffers, and consequently for blocks. Now the procedure of reading a file can be specified.

The block with the maximum possible blocking factor is transferred from disk into the buffer organized in primary storage. Next, only one record is read from that buffer and brought to another (smaller) buffer called a *record buffer,* which is equal in size to the biggest record available in the file. From there the record goes to the A&LU where it is processed according to the program. Naturally, reading record after record from block buffer (i.e., reading in electronic primary storage) is thousands of times faster than reading from the disk. Therefore, it is important to remember that a blocking factor greater than 1 will always improve overall performance and the system throughout by cutting down on the amount of I/O that the system must perform.

So, the greater the blocking factor, the higher the performance. But we have already pointed out that the size of a block is limited. This prompts the following question: If the block's size is limited, then is it possible to reduce the size of the record, thereby increasing the blocking factor? At first thought, the idea seems somewhat strange, because the record contains information needed to form the output. In our file, for example, the record keeps only the document text and the document profile (Figure 8.1). Yet this idea is fruitful, as can be seen in our next example.

It is quite clear that a subject search requires only document profiles, and the document texts are not involved in this process. This being the case, is it not possible to remove the biggest part of the record, that is, the document text (field 1)? This will allow us to substantially increase the blocking factor and, consequently, the speed of the search. Of course, it is possible and this is exactly how it is done in practice. But what about the output, which must contain the texts of the found documents? The answer lies in forming two files (and this is done in practice): one for the subject search (a file consisting of document profiles) and the other to form the output (a file consisting of the original documents). The records in these files are organized as follows. The record for subject

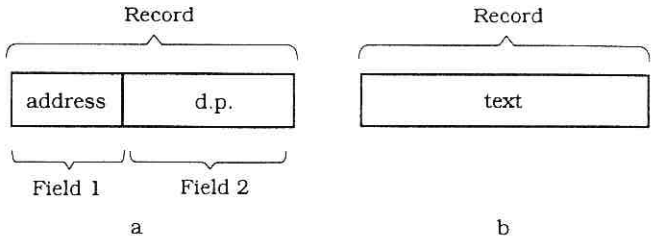

Figure 8.3
Records from newly formed files.

search also contains two fields: field 2, as before, contains a document profile, and field 1, instead of a full document (which is many times larger than the document profile), contains a short address of that document in the second file. Figure 8.3(a) gives an example of such a record.

In the second file, the record contains only the document text, and the whole record essentially consists of one field. An example of such a record is given in Figure 8.3(b). But from where is the address taken and what does it look like in practice? Perhaps the simplest way of explaining this is as follows. When each of the 10,000 documents are entered into the IR system (they are entered in order of their arrival), each document is assigned an ordinal number (from 0000 to 9999, for example). After indexing a document, records are formed (similar to those given in Figure 8.3), and the document's ordinal number is entered as an address into the document profile record (field 1). Then the records formed are put into the files, with the text record being put into that part of the text file that corresponds to the ordinal number. With this kind of file organization existing in the BSR, the process (in general) may look as follows.

There are two stages realized by the search program, which handles two files. The first stage has to do with the file of a document profile, and the second stage concerns the text file. In this case, the program essentially provides for two different searches (one following the other). The first is a subject search in the sequentially organized file of a document profile, and the other is an *address search,* that is, a search for specially selected addresses (a result of the first search) done in the text file providing for random access. It means that in the case of address search there is no need to read all the blocks from the text file sequentially, and only those containing needed records are read. Because it does not require reading the whole file, this approach understandably expedites the search and presupposes the imperative use of file organization, other than sequential organization. The kinds of file organization that can be used for random access is the subject of the next section of this chapter. Now we will give a more detailed description of the search carried out in the BSR.