Quicksort, with median-of-three pivoting, requires comparing `a[first]`, `a[center]`, and `a[last]` in a constant number of time units. If the input is on a tape, then all of these operations lose their efficiency because elements on a tape can be accessed only sequentially. Even if the data is on a disk, there is still a practical loss of efficiency because of the delay required to spin the disk and move the disk head.

To see how slow external accesses really are, we could create a random file that is large but not too big to fit in main memory. When we read in the file and sort it using an efficient algorithm, the time to read the input is likely to be significant compared to the time it takes to sort the input, even though sorting is an $O(N \log N)$ operation (or worse for Shellsort) and reading the input is only $O(N)$.

20.6.2 Model for External Sorting

We assume sorts are performed on tape. Only sequential access of the input is allowed.

The wide variety of mass storage devices makes external sorting much more device-dependent than internal sorting does. The algorithms considered here work on tapes, which are probably the most restrictive storage medium. Since access to an element on tape is done by winding the tape to the correct location, tapes can be efficiently accessed only in sequential order (in either direction).

Assume we have at least three tape drives to perform the sorting. We need two drives to do an efficient sort; the third drive simplifies matters. If only one tape drive is present, then we are in trouble: Any algorithm will require $\Omega(N^2)$ tape accesses.

20.6.3 The Simple Algorithm

The basic external sort uses repeated two-way merging. Each sorted group is a *run*. As a result of a pass, the length of the runs doubles and eventually only a single run remains.

The basic external sorting algorithm uses the merge routine from mergesort. Suppose we have four tapes A1, A2, B1, and B2, which are two input and two output tapes. Depending on the point in the algorithm, the A and B tapes are either input tapes or output tapes. Suppose the data is initially on A1. Suppose further that the internal memory can hold (and sort) M records at a time. The natural first step is to read M records at a time from the input tape, sort the records internally, and then write the sorted records alternately to B1 and B2. Each set of sorted records is called a *run*. When this is done, we rewind all the tapes. Suppose we have the same input as our example for Shellsort. The initial configuration is shown in Figure 20.32. If $M = 3$, then after the runs are constructed, the tapes will contain the data as shown in Figure 20.33.

Now B1 and B2 contain a group of runs. We take the first runs from each tape, merge them, and write the result, which is a run twice as long, onto A1. Then we take the next runs from each tape, merge them, and write the result out to A2. We continue this process, alternating output to A1 and A2 until either B1 or B2 is empty. At this point, either both are empty or there is one (possibly short)