Consider the problem of sorting the following file using two way external merge sort. Assume it consists of 8 blocks of 4 records each, and that main memory is only large enough to sort 1 block at a time. We will regard this as initially consisting of 32 runs of size 1 - ignoring any initial order.

Cat
Unicorn
Dog
Bison
Jackal
Elephant
Vixen
Gopher
Fox
Xerus
Hippo
Cow
Hyena
Giraffe
Iguana
Aardvark
Tiger
Kangaroo
Eel
Buffalo
Zebra
Llama
Mouse
Lion
Newt
Parrot
Snake
Osprey
Yak
Penguin
Raccoon
Turtle

Regard as 32 runs of length 1

Split into two scratch files of 4 blocks each, writing alternate blocks to each file.

This requires reading all 8 blocks and writing all 8 blocks

We call this Pass 0.

Cat
Unicorn
Dog
Bison
Fox
Xerus
Hippo
Cow
Tiger
Kangaroo
Eel
Buffalo
Newt
Parrot
Snake
Osprey

Jackal
Elephant
Vixen
Gopher
Hyena
Giraffe
Iguana
Aardvark
Zebra
Llama
Mouse
Lion
Yak
Penguin
Raccoon
Turtle

Regard each scratch file as consisting of 16 runs of length 1.

Now merge successive pairs of runs from each of the two files, writing the results to alternate blocks of two new scratch files.

This requires reading all 8 blocks and writing all 8 blocks.

We call this Pass 1

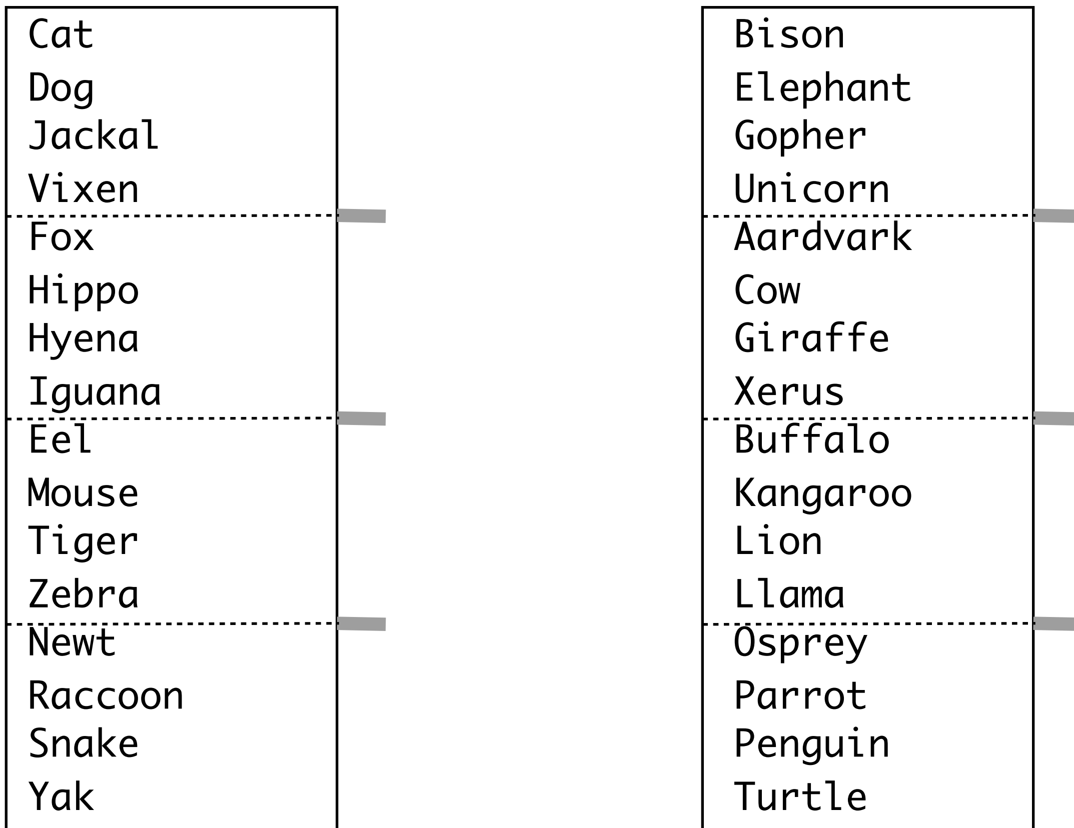
File 1	File 2
Cat	Dog
Jackal	Vixen
Elephant	Bison
Unicorn	Gopher
Fox	Hippo
Hyena	Iguana
Giraffe	Aardvark
Xerus	Cow
Tiger	Eel
Zebra	Mouse
Kangaroo	Buffalo
Llama	Lion
Newt	Raccoon
Yak	Snake
Parrot	Osprey
Penguin	Turtle

Regard each scratch file as consisting of 8 runs of length 2.

Now merge successive pairs of runs from each of the two files, writing the results to alternate blocks of the two scratch files that became empty after Pass 1.

This requires reading all 8 blocks and writing all 8 blocks.

We call this Pass 2

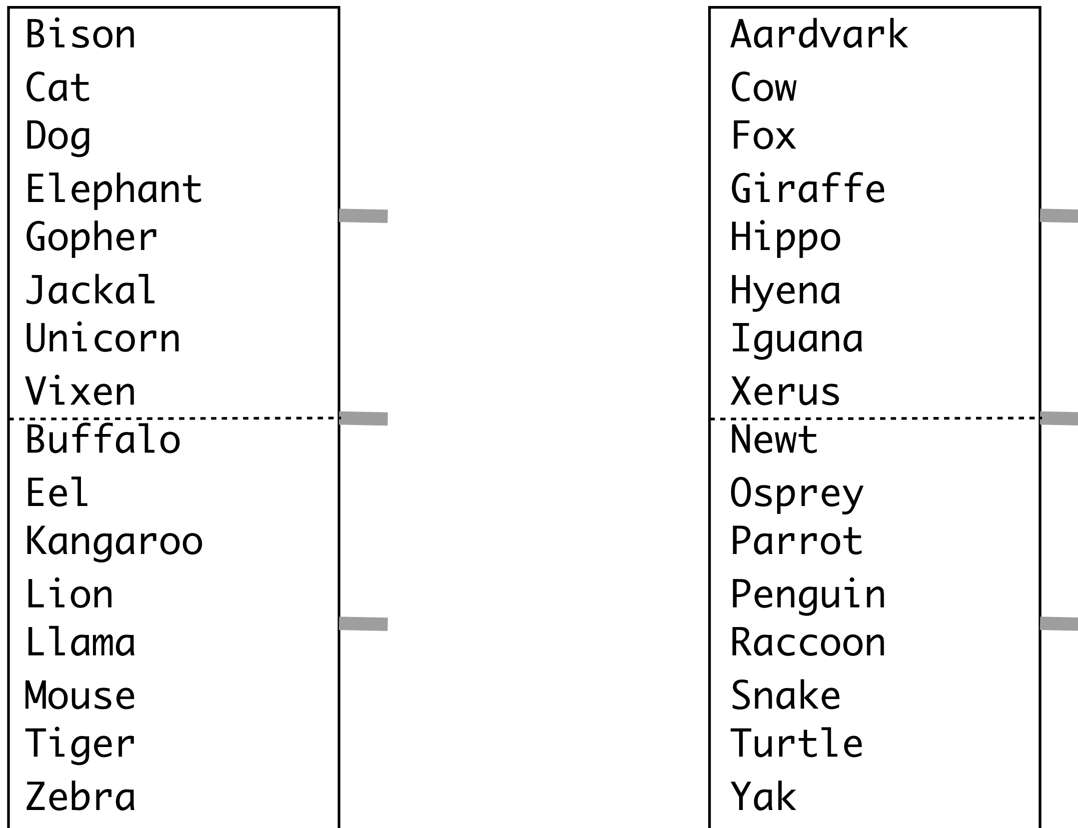


Regard each scratch file as consisting of 4 runs of length 4.

Now merge successive pairs of runs from each of the two files, writing the results to alternate blocks of the two scratch files that became empty after Pass 2.

This requires reading all 8 blocks and writing all 8 blocks.

We call this Pass 3

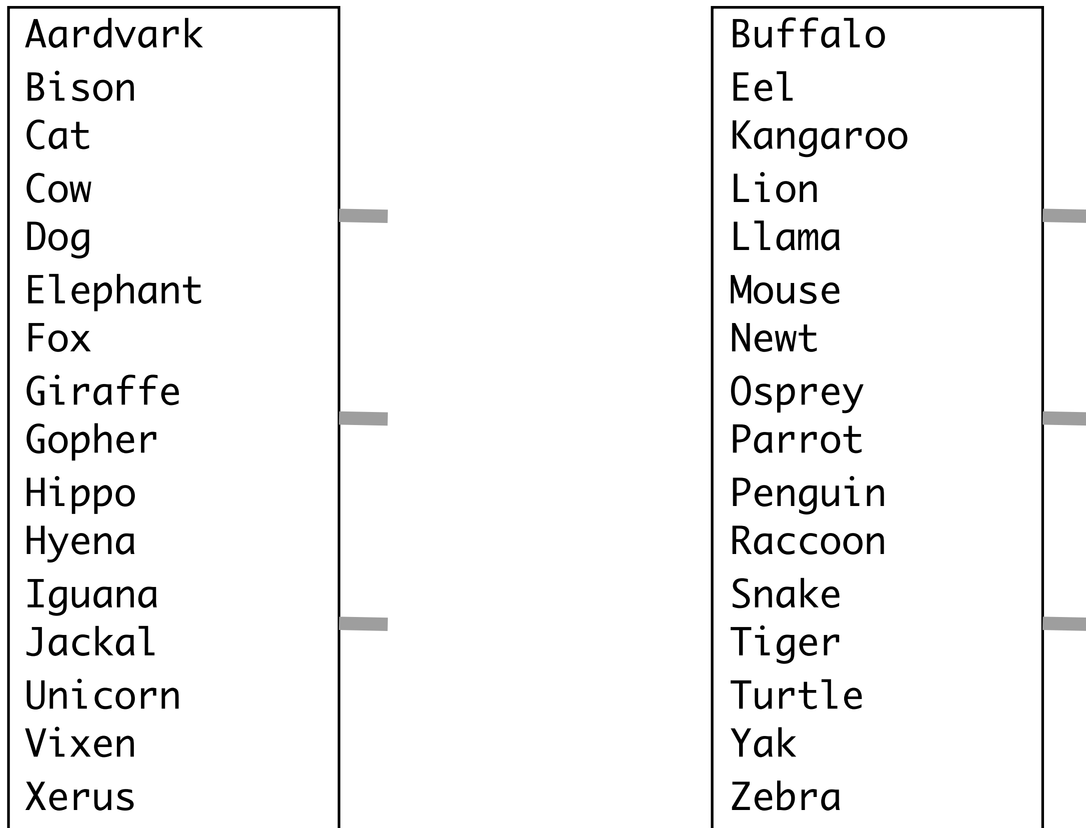


Regard each subfile as consisting of 2 runs of length 8.

Now merge successive pairs of runs from each of the two files, writing the results to alternate blocks of the two scratch files that became empty after Pass 3.

This requires reading all 8 blocks and writing all 8 blocks.

We call this Pass 4



Regard each subfile as consisting of 1 run of length 16.

Merge these to produce the final sorted file

This requires reading all 8 blocks and writing all 8 blocks.

We call this Pass 5

Aardvark
Bison
Buffalo
Cat
Cow
Dog
Eel
Elephant
Fox
Giraffe
Gopher
Hippo
Hyena
Iguana
Jackal
Kangaroo
Lion
Llama
Mouse
Newt
Osprey
Parrot
Penguin
Raccoon
Snake
Tiger
Turtle
Unicorn
Vixen
Xerus
Yak
Zebra

Total effort is one initial distribution pass plus five merge passes, or 6 passes in all. On each pass we read all 8 blocks and wrote all 8 blocks - or 48 reads and 48 writes in all.

In general, if there are n records grouped b per block (here n = 32 and b = 4), then the total number of reads is (1+log n) passes * (n/b) and the same number of writes.

Possible improvements:

1) Take advantage of existing natural order in the file to reduce the initial number of runs, and hence the number of passes needed to merge down to 1 run. This is called NATURAL MERGE

   In a totally random file, the expected average run length is 2 - which results in reducing the number of merge passes by 1.

2) During the initial distribution pass, use an internal sort to sort each block. (The additional time cost of this is small when compared to the time cost for the disk accesses.) [For our example data, this would result in 8 runs of length 4 initially, reducing the number of merge passes by 2. Total reads would be 8 * (1+3) = 32 instead of 48.]

   In a more typical case, this can result in a much greater reduction in the number of passes. For example, if we could sort 1024 items internally, we could reduce the number of merge passes needed by 10, since $\log_2 1024 = 10$.

3) Instead of doing a 2-way merge, do a 3 or higher way merge. The internal code needed to do this is more complex, but this can significantly reduce the number of passes needed since the number of passes is $\log_{\text{merge-order}} n$ - e.g. a 3-way merge needs only 63% as many merge passes as a 2-way merge for the same number of items. [For the example we just did, after one merge pass we would have 11 runs of length 2-3; after two passes 4 runs of length 5-9 ; after three passes 2 runs; and the fourth pass would produce the final result.]

   (Greater improvements are possible by using even higher merge orders, though possible improvement achievable this way is limited by the amount of additional memory needed for buffers and disk space needed for scratch files.)

1) An even greater improvement is possible by using a technique known as POLYPHASE merging.

Suppose we used internal sorting to create 8 initial runs of length 4. But instead of dividing these initial runs evenly between the two scratch files, we put 5 in one scratch file and 3 in the other.

We could then proceed in a series of phases as follows:

Phase	Run Distribution	Blocks Read
0 - Initial distribution	5, 3 blocks of length 1	8
1 - Merge 3 runs	3 @ 2 blocks	6
	2 left over @ 1 block	
2 - Merge 2 runs	2 @ 3	6
	1 left over @ 2	
3 - Merge 1 run	1 @ 5	5
	1 left over @ 3	
4 - Final merge	1 run of length 8 blocks	8
Total		33 blocks

(This strategy can achieve greater savings when used with higher merge orders.)