Performance analysis

The solution is based on quick sort algorithm with average complexity is n*log(n), IO operations are performed using MappedByteBuffer, which means that everything related to caching is the responsibility of the OS. There is a considerable overhead related to MappedByteBuffer that causes a huge measurement errors for relatively small amounts of data. The algorithm itself is pretty simple, profiling shows that threads spend about 80% of their lifetime (I suppose that a lot more, maybe this is just a profiler measurement error) being blocked in IO operations, which means that the overall performance is practically restricted by the hard drive speed. Together with the thread management overhead that could explain low efficiency of multithreading for less than 400 MB files.

Stand:

- core i5 2400k
- 16 GB RAM
- Windows 7 x64
- Windows 7 data transfer rate score 5,9

10 million integers

Threads	Seconds
64	1,559
32	1,161
16	1,153
8	1,461
4	1,330
2	1,358
1	2,122

100 million integers

Threads	Seconds
64	13,543
32	14,304
16	14,119
8	15,062
4	13,790

2	16,851
1	24,795

250 million integers

Threads	Seconds
64	28,802
32	30,441
16	29,846
8	58,861
4	68,399
2	133,488
1	177,973

Performance improvements

BufferAggregator implementation is quite memory-consuming (see todos in the source code for details), its performance can be improved by reducing objects creation (benchmarking is required). The main problem is that the application highly depends on OS files mapping mechanism, in case of big files it does not work so well (at least on my machine), performance greatly degrades with file size increasing:

64 threads

Integers	Seconds
500 million	~ 60
1 billion	~480
2 billion	n/a

Possible solution could be split file on parts 1 or 2 GB large, sort them in course and merge using additional disk space equal to the original file size. The implementation is quite simple, using the existing mechanism of buffers aggregation (or only simple IO operations without file mapping), it would take about 1 m.d. Merging means additional IO operations, that will increase n*log(n) complexity, but it also will reduce the OS caching system load and have a positive impact on large (>2 GB) files sorting time (needs benchmarking).