A "sort" is adaptive if it takes advantage of existing order within the data. Most adaptive sorts are a variation on a more standard sorting algorithm. Below is a description of the protoypical adaptive merge sort, originally called "the adaptive sort".

Adaptive sort

Adaptive sort might be viewed as a kind of variation on mergesort. You will recall that mergesort splits data in half at the middle, sorts each half (recursively) and then merges the two sorted halves. Well, adaptive sort also splits the dataset in two, but then it tries to merge the two halves straight away, and only if the merge fails does it sort each half (recursively), then it merges the two sorted halves just like mergesort.

Does this really help? It sounds like adaptive sort does the same as mergesort, except that it more or less doubles the number of merge operations by optimistically attempting to merge the two halves at the outset. Surely only a very fortuitous ordering of the data would allow such a merge to succeed without some additional sorting. So each halving of the data would simply lead to two merge operations: the optimistic one that probably fails, followed by the one needed to merge the two halves together after they have been sorted. Plain old mergesort just needs the last one. How can this yield any advantage?

The key to adaptive sort is in the way it splits the data. Instead of crudely splitting the dataset in half at the mid-point, adaptive sort treats every other datum as belonging to the same half-set; such that every even-indexed item in the original dataset ends up in one half-set, and all the odd-indexed items go in the other half-set. This is called a *modulo-2 split*.

More precisely, ...

Consider a set X of N data in random order, where $X = \{x_1,x_2,x_3, \ldots ,x_N\}$. Adaptive sort carries out a modulo-2 split to form two subsets, $X_A = \{x_1,x_3,x_5, \ldots ,x_{maxOdd}\}$ and $X_B = \{x_2,x_4,x_6, \ldots ,x_{maxEven}\}$. If subsets X_A and X_B can be successfully merged, then the result is X in sorted order. Otherwise, each subset is sorted using adaptive sort, after which a merge of the two sorted subsets must succeed. How unsorted X is at the start determines how many times such attempted pre-merges fail.

So how do we measure unsortedness?

If X is not sorted, then there are at least two data in X that are out of order with respect to each other. That is, there exists two integers, $0<i<N$ and $i<j<=N$ such that $x_i>x_j$. This situation is called an *inversion* of order *(j-i)*.

For any unordered data there exists some maximum order inversion— an inversion of *order-p*. Such data is said to be *p-sorted*. At most, p = (N-1) when the first and last items in X are out of order with respect to each other. If p = 1, then any out of order items are right next to each other. By convention, p = 0 if the data is sorted.

Note that in the case where data is 1-sorted, the modulo-2 split will be followed by a successful pre-merge. This is because any data that were out of order with respect to each other prior to the modulo-2 split are put into separate subsets by the split. If it were true that two items in a subset were out of order with respect to each other, then it had to be true that there was at least an order-2 inversion in the original dataset and therefore the data was not 1-sorted.

More generally, any time a dataset is modulo-2 split, p is halved. If you think about it, this is because half of all the items between the i-th and j-th ones (i.e. the ones involved in the maximum inversion) go in one list and the other half go in the other list— either x_i and x_j end up in different subsets, thus removing the inversion; or the distance between them halves.

It follows that the maximum number of times a dataset can be split is $\log_2 p$. A split can be executed with N operations, and each merge entails a maximum of N-1 operations, meaning adaptive sort is in $O(N \log_2 p)$, which is of course the same as $O(N \log_2 N)$ in light of the worst case.

Thus it is that adaptive sort works well with data that is partially sorted—reducing computational requirements as a function of the amount of disorder present in the input. However, in practice the savings are small, and typically offset by the doubling of attempted merges. It is, however, a simple and effective method to mitigate thrashing for an external sort (i.e. when data size exceeds the size of main memory).