In computer science, a **selection algorithm** is an algorithm for finding the th smallest number in a list or array; such a number is called the th *order statistic*. This includes the cases of finding the *minimum*, *maximum*, and *median elements*. There are (worst-case linear time) selection algorithms, and sublinear performance is possible for structured data; in the extreme, for an array of sorted data. Selection is a subproblem of more complex problems like the *nearest neighbor* and *shortest path*problems. Many selection algorithms are derived by generalizing a sorting algorithm, and conversely some sorting algorithms can be derived as repeated application of selection.

The simplest case of a selection algorithm is finding the minimum (or maximum) element by iterating through the list, keeping track of the running minimum – the minimum so far – (or maximum) and can be seen as related to the selection sort. Conversely, the *hardest case* of a selection algorithm is finding the median, and this necessarily takes  storage. In fact, a specialized median-selection algorithm can be used to build a general selection algorithm, as in median of medians. The *best-known* selection algorithm is *quickselect*, which is related to quicksort; like quicksort, it has (asymptotically) optimal average performance, but *poor worst-case performance*, though it can be modified to give optimal worst-case performance as well.

By sorting the list or array then selecting the desired element, selection can be reduced to sorting. This method is inefficient for selecting a single element, but is efficient when many selections need to be made from an array, in which case only one initial, expensive sort is needed, followed by many cheap selection operations – for an array, though selection is in a list, even if sorted, due to lack of random access. In general, sorting requires time, where  is the length of the list, although a lower bound is possible with non-comparative sorting algorithms like radix sort and counting sort.

Rather than sorting the whole list or array, one can instead use partial sorting to select the k smallest or  largest elements. The th smallest (resp., th largest element) is then the largest (resp., smallest element) of the partially sorted list – this then takes to access in an array and to access in a list. This is more efficient than full sorting, but less efficient than simply selecting, and takes time, due to the sorting of the  elements. Partial sorting algorithms can often be derived from (total) sorting algorithms. As with total sorting, partial sorting means that further selections (below the th element) can be done in time for an array and time for a list.

Further, if the partial sorting also partitions the original data into "sorted" and "unsorted", as with an in-place sort, the partial sort can be extended to a larger partial sort by only sorting the incremental portion, and if this is done, further selections above the th element can also be done relatively cheaply.