**Summary of Sorting**

|  |  |  |
| --- | --- | --- |
|  | Best-case Cost | Worst-case Cost |
| Selection | O(n2) | O(n2) |
| Insertion | O(n) | O(n2) |
| Heap | O(n log n) | O(n log n) |
| Merge | O(n log n) | O(n log n) |
| Quick | O(n log n) | O(n2) |

* Selection sort performs the least swaps, O(n), in the worst case.
* Insertion sort is best if the array is nearly or already sorted.
* Heap sort performs a constant factor more comparisons than Merge sort
* Merge sort requires extra storage proportional in size to the input.
* Quick sort typically (expected case) outperforms Heap and Merge sort because of its simplicity.



split

split

split

split

split

join

join

join

join

join

sorted

unsorted

= unsorted  = sorted  = sort process

**Figure 1: Hypothetical Sort Recursion Tree.**

**Table 1** below summarizes the split/join operations of a few common sort algorithms.

|  |  |  |
| --- | --- | --- |
| **Sort** | **Split operation** | **Join operation** |
| *Insertion* | Return *hi* | Insert *A*[*hi*] into proper location. |
| *Merge* | Return midpoint index. | Merge subarrays. |
| *Quick* | Find and return pivot point index | Do nothing. |
| *Selection* | Swap extremum with *A*[*hi*] and return *hi* | Do nothing. |
| *Bubble* | Bubble up extremum to *A*[*hi*] and return *hi* | Do nothing. |
| *Heap* | Swap extremum (*A*[*lo*]) and *A*[*hi*], reheapify *A*[*lo*, *hi-*1], and return *hi*. | Do nothing. |

From **Table 1**, we can see that selection sort, bubble sort, and heap sort1 are essentially identical

processes, though they have different algorithmic complexities: they all pull out the extremum from the

array and split it off. A trivial no-op join then follows this. Quick sort is similar to the

selection/bubble/heap genera except that it pulls off a set of one or more extrema values.

On the flip side of the coin, we see that insertion sort and merge sort are similar in that their split

operations are trivial while their join operations are more complex. Insertion splits off one element at a

time while merge sort splits the array in half each time. One can think of the join() method in insertion

sort as merging a sorted array with a one-element array (which is obviously sorted).