Comparison of array sorting methods

C = number of comparisons
M = number of data moves
Data moves are costlier than comparisons

Insertion sort
When the data are in order,
$C_{min} = \sum_2^n 1 = n - 1$,
$M_{min} = \sum_2^n 2 = 2(n - 1)$.
When the data are in reverse order,
$C_{max} = \sum_2^n i = ½ (n - 1) (n + 2) = ½ (n^2 + n) - 1$,
$M_{max} = \sum_2^n (i + 2) = ½ (n^2 + 5n - 6)$.
On an average,
$C_{ave} = \sum_2^n i/2 = ¼ (n^2 + n - 2)$,
$M_{ave} = \sum_2^n (i/2 + 2) = ¼ (n^2 + 9n - 10)$.
Thus, insertion sort exhibits a natural behavior. Also, insertion sort is stable and in-place.

Binary insertion sort
Exercise

Selection sort
Evidently, the number of comparisons is independent of the order of data. In this sense, this method may be said to behave less naturally than insertion sort.
$C = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1$
$= \sum_{i=1}^{n-1} [n - (i + 1) + 1]$
$= \sum_{i=1}^{n-1} (n - I) = ½ (n^2 - n)$.
Also,
$M_{min} = 3 (n - 1)$ in the case of ordered data.
$M_{max} = 3 (n - 1) + [(n - 1) + (n - 3) + (n - 5) + \ldots]$, in case of reverse ordered data
$= 3 (n - 1) + \text{trunc}(n^2/4)$.

M_{ave} is difficult to determine.

Selection sort is not stable. The problem comes from the algorithm. When there are two occurrences of the same datum, the last is selected as the one with least key; this is exchanged with the first of the current list of a[i]…a[n]. The other occurrence is selected next and is swapped with the next first, which is behind the previous first, and thus they are interchanged.

Selection sort is in-place.

Bubble sort
$C = (n - 1)(n - 1) = n^2 - 2n + 1$
$M_{min} = 0$
$M_{max} = (n - 1) + (n-3) + \ldots = \text{trunc}(n^2/4)$

$M_{ave} = n^2/4$
Bubble sort is in-place and stable.

Flagged bubble sort
Exercise

Quick sort

If we are lucky then each time a datum is correctly positioned, the subarray to its left will be of the same size as that to its right. This would leave us with the sorting of two subarrays each of size n/2 roughly. If this happens through all the passes then the total number of comparisons is approximately
$n + 2*n/2 + 4*n/4 + 8*n/8 + \ldots n*n/n.$
Assuming, $n = 2^m$ such that $m = \log_2 n$ this is same as
$n + n + n + \ldots$ m times
$= n* \log_2 n.$
However, if the array is already sorted in ascending/descending order then it divides into
$n + (n - 1) + (n - 2) + \ldots + 2$ or its reverse i.e., $O(n^2)$.

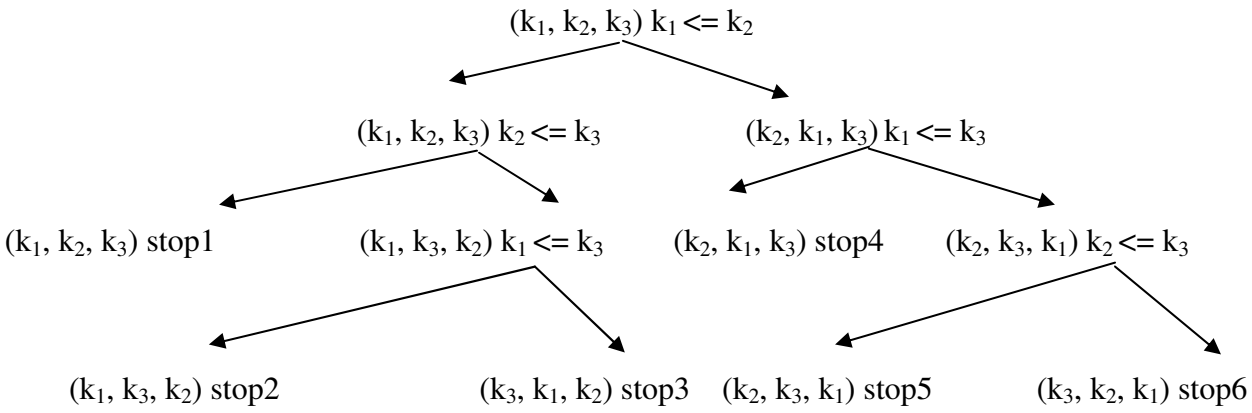
Heap sort

Suppose $2^{k-1} <= n < 2^k$ so that the tree has k levels and the number of nodes on level i is 2^{i-1}. In the first *for* loop create-heap is called once for each node that has a child. Hence, the time required for this loop is the sum, over each level, of the number of nodes on a level times the maximum distance the can move i.e., $\sum_1^k 2^{i-1}$ as a node at level may move down to level k. This is equal to
$2^0 * (k - 1) + 2 * (k-2) + \ldots + 2^{k-2} * 1 + 2^{k-1} * 0$
$= 2^0 * (k-1) + 2 * (k-2) + \ldots + 2^{k-2} * 1$
$= 1 * 2^{k-2} + 2 * 2^{k-3} + \ldots + (k-2) * 2 + (k-1) * 2^0$
$= \sum_1^{k-1} i*2^{k-1-i} = \sum_1^{k-1} i*2^{k-i-1}$
$= 2^{k-1} \sum_1^{k-1} i/2^i <= n\sum_1^{k-1} i/2^i < 2n = O(n).$

In the next *for* loop n-1 applications of create-heap are made with maximum depth $k = \log_2(n+1)$ as only the root is out of place and moves at most k levels down. Hence, the computing time for this loop is $O(n*\log_2 n)$. Consequently, the total computing time is
$O(n) + O(n*\log_2 n) = O(n*\log_2 n)$ as $n*\log_2 n >= n$ for $n >= 2$.
Also, the only additional space required is for an interchange in the second loop.

Best possible time for sorting by comparisons and interchanges

Decision tree (of three data k_1, k_2, k_3)



3! = 6 permutations are possible with 3 data and hence the 6 outputs. A decision tree that sorts n data may give rise to n! permutations and, therefore, has n! leaves. But a tree with height k can have at most 2^{k-1} leaves. Height of the decision tree, therefore, must be at least $1 + \log_2 n!$. (For n = 4, $1 + \log_2 n! = 5$.)

Now $n! = n(n-1)(n-2)\ldots2.1 \Rightarrow (n/2)^{n/2}$
Or $\log_2 n! \Rightarrow n/2 \log_2 n/2 = O(n \log_2 n)$

Thus, height of the decision tree is at least $O(n \log_2 n)$. Therefore, #comparisons/exchanges is at least $O(n \log_2 n)$. In other words, the least possible time for sorting by comparisons/exchanges is $O(n \log_2 n)$.