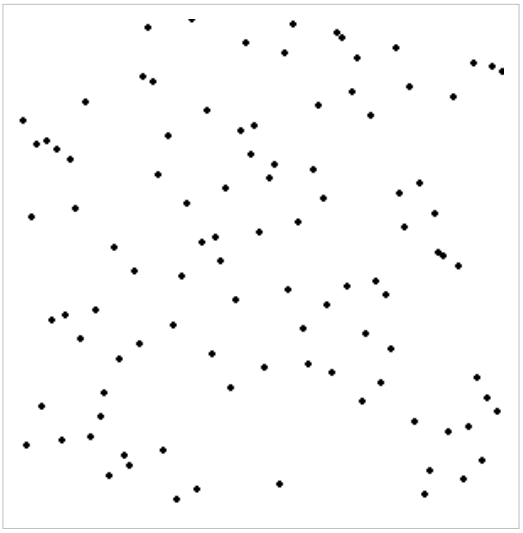
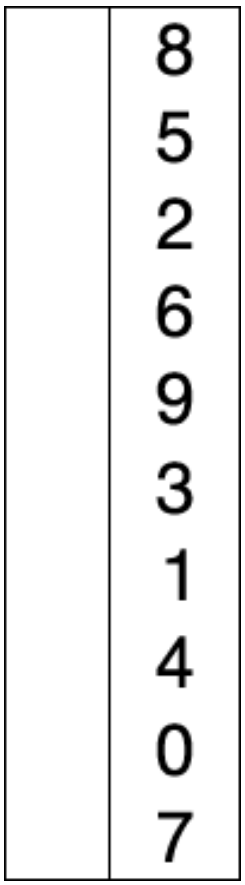
Selection sort



Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n^2)$
Best case performance	$O(n^2)$
Average case performance	$O(n^2)$
Worst case space complexity	$O(n)$ total, $O(1)$ auxiliary

Selection sort is a sorting algorithm, specifically an in-place comparison sort. It has O(n^2) complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.



Selection Sort Animation

Algorithm

The algorithm works as follows:

1. Find the minimum value in the list
2. Swap it with the value in the first position
3. Repeat the steps above for the remainder of the list (starting at the second position and advancing each time)

Effectively, the list is divided into two parts: the sublist of items already sorted, which is built up from left to right and is found at the beginning, and the sublist of items remaining to be sorted, occupying the remainder of the array.

Here is an example of this sort algorithm sorting five elements:

```
64 25 12 22 11

11 25 12 22 64

11 12 25 22 64

11 12 22 25 64

11 12 22 25 64
```

(nothing appears changed on this last line because the last 2 numbers were already in order)

Selection sort can also be used on list structures that make add and remove efficient, such as a linked list. In this case it's more common to *remove* the minimum element from the remainder of the list, and then *insert* it at the end of the values sorted so far. For example:

```
64 25 12 22 11

11 64 25 12 22

11 12 64 25 22

11 12 22 64 25
```

```
11 12 22 25 64
```

```
/* a[0] to a[n-1] is the array to sort */
int iPos;
int iMin;

/* advance the position through the entire array */
/*   (could do iPos < n-1 because single element is also min element) */
for (iPos = 0; iPos < n; iPos++)
{
  /* find the min element in the unsorted a[iPos .. n-1] */

  /* assume the min is the first element */
  iMin = iPos;
  /* test against all other elements */
  for (i = iPos+1; i < n; i++)
    {
      /* if this element is less, then it is the new minimum */
      if (a[i] < a[iMin])
        {
          /* found new minimum; remember its index */
          iMin = i;
        }
    }

  /* iMin is the index of the minimum element. Swap it with the current
 position */
  swap(a, iPos, iMin);
}
```

Mathematical definition

Let L be a non-empty set and $f : L \to L$ such that $f(L) = L'$ where:

1. L' is a permutation of L,
2. $e_i \leq e_{i+1}$ for all $e \in L'$ and $i \in \mathbb{N}$,
3. $f(L) = \begin{cases} L, & \text{if } |L| = 1 \\ \{s\} \cup f(L_s), & \text{otherwise} \end{cases}$,
4. s is the smallest element of L, and
5. L_s is the set of elements of L without one instance of the smallest element of L.

Analysis

Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array. Selecting the lowest element requires scanning all n elements (this takes $n - 1$ comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining $n - 1$ elements and so on, for $(n - 1) + (n - 2) + ... + 2 + 1 = n(n - 1) / 2 \in \Theta(n^2)$ comparisons (see arithmetic progression). Each of these scans requires one swap for $n - 1$ elements (the final element is already in place).

Comparison to other sorting algorithms

Among simple average-case $\Theta(n^2)$ algorithms, selection sort almost always outperforms bubble sort and gnome sort, but is generally outperformed by insertion sort. Insertion sort is very similar in that after the kth iteration, the first k elements in the array are in sorted order. Insertion sort's advantage is that it only scans as many elements as it needs in order to place the $k + 1$st element, while selection sort must scan all remaining elements to find the $k + 1$st element.

Simple calculation shows that insertion sort will therefore usually perform about half as many comparisons as selection sort, although it can perform just as many or far fewer depending on the order the array was in prior to sorting. It can be seen as an advantage for some real-time applications that selection sort will perform identically regardless of the order of the array, while insertion sort's running time can vary considerably. However, this is more often an advantage for insertion sort in that it runs much more efficiently if the array is already sorted or "close to sorted."

While selection sort is preferable to insertion sort in terms of number of writes ($\Theta(n)$ swaps versus $O(n^2)$ swaps), it almost always far exceeds (and never beats) the number of writes that cycle sort makes, as cycle sort is theoretically optimal in the number of writes. This can be important if writes are significantly more expensive than reads, such as with EEPROM or Flash memory, where every write lessens the lifespan of the memory.

Finally, selection sort is greatly outperformed on larger arrays by $\Theta(n \log n)$ divide-and-conquer algorithms such as mergesort. However, insertion sort or selection sort are both typically faster for small arrays (i.e. fewer than 10-20 elements). A useful optimization in practice for the recursive algorithms is to switch to insertion sort or selection sort for "small enough" sublists.

Variants

Heapsort greatly improves the basic algorithm by using an implicit heap data structure to speed up finding and removing the lowest datum. If implemented correctly, the heap will allow finding the next lowest element in $\Theta(\log n)$ time instead of $\Theta(n)$ for the inner loop in normal selection sort, reducing the total running time to $\Theta(n \log n)$.

A bidirectional variant of selection sort, called **cocktail sort**, is an algorithm which finds both the minimum and maximum values in the list in every pass. This reduces the number of scans of the list by a factor of 2, eliminating some loop overhead but not actually decreasing the number of comparisons or swaps. Note, however, that cocktail sort more often refers to a bidirectional variant of bubble sort.

Selection sort can be implemented as a stable sort. If, rather than swapping in step 2, the minimum value is inserted into the first position (that is, all intervening items moved down), the algorithm is stable. However, this modification either requires a data structure that supports efficient insertions or deletions, such as a linked list, or it leads to performing $\Theta(n^2)$ writes.

In the **bingo sort** variant, items are ordered by repeatedly looking through the remaining items to find the greatest value and moving all items with that value to their final location. Like counting sort, this is an efficient variant if there are many duplicate values. Indeed, selection sort does one pass through the remaining items for each item moved. Bingo sort does two passes for each value (not item): one pass to find the next biggest value, and one pass to move every item with that value to its final location. Thus if on average there are more than two items with each value, bingo sort may be faster.[1]

References

[1] *This article incorporates public domain material from the NIST* document "Bingo sort" (http://www.nist.gov/dads/HTML/bingosort.html) by Paul E. Black (Dictionary of Algorithms and Data Structures).

- Donald Knuth. *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0. Pages 138–141 of Section 5.2.3: Sorting by Selection.
- Anany Levitin. *Introduction to the Design & Analysis of Algorithms*, 2nd Edition. ISBN 0-321-35828-7. Section 3.1: Selection Sort, pp 98-100.
- Robert Sedgewick. *Algorithms in C++, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching: Fundamentals, Data Structures, Sorting, Searching Pts. 1-4*, Second Edition. Addison-Wesley Longman, 1998. ISBN 0-201-35088-2. Pages 273–274

External links

- Animated Sorting Algorithms: Selection Sort (http://www.sorting-algorithms.com/selection-sort) – graphical demonstration and discussion of selection sort
- Applet and source code (http://www.miaowang.de/studium/tutorials/applets/selectionsort_en.html)
- Selection Sort in C++ (http://24bytes.com/selection-sort.html)
- Selection Sort Demonstration (http://web.engr.oregonstate.edu/~minoura/cs162/javaProgs/sort/SelectSort.html)
- Selection sort illustrated explanation. Java and C++ implementations. (http://www.algolist.net/Algorithms/Sorting/Selection_sort)

Article Sources and Contributors

Selection sort *Source*: http://en.wikipedia.org/w/index.php?oldid=406515759 *Contributors*: Abu adam, Ahy1, Alexius08, Ario, Arthur Rubin, Arun APEC, Aseld, AxelBoldt, Balabiot, Beetstra, BiT, Bpapa2, BrokenSegue, CJLL Wright, CardinalDan, Ceros, Closedmouth, Clx321, Conversion script, DaGizza, Daehrednud, Dave683, David Eppstein, Dcoetzee, DevastatorIIC, Dissident, Dmaclach, Doradus, El C, Eric119, Fangyuan1st, Flatline, Fumitol, Gaspercat, GeorgeBills, Giftlite, Gilliam, Glrx, Grick, HJ Mitchell, HairyFotr, Harrisonmetz, Head, Hede2000, Hyad, Hyegolfer, Ian Ashley, Ilyathemuromets, J.delanoy, JLaTondre, JMRodrigues, Jassyt, Javierito92, Jbonneau, Jesin, Joestape89, Joey-das-WBF, Johnl1479, Jonhall, Josh Kehn, Juliancolton, Kaly J., Klutzy, Knuckles, Knutux, Kusunose, LeaveSleaves, Lee Daniel Crocker, Lights, Looxix, Mahanga, Mantipula, Marco Polo, Mariosal, Mike.pr, Mullacy, Nsrao2k, Nuno Tavares, Olathe, Oli Filth, Opraveen, Oskar Sigvardsson, Oğuz Ergin, Paul Ebermann, Peristarkawan, Pfalstad, PhilipMW, Pichpich, Pingveno, Pred, Qst, Rhanekom, Robert Merkel, Romain Thouvenin, Ruud Koot, SPTWriter, Shreeniwasiyer, Silly rabbit, Simguru, SiobhanHansa, Sligocki, Smckenna999, Smjg, Starkana, Starylon, Swift, TakuyaMurata, Tbhotch, Teacup, Timwi, Userabc, VTBassMatt, VernoWhitney, Willking1979, Ww, Ycl6, Yelod, Yksyksyks, Zdeneks, ZeroOne, Zvar, Михајло Анђелковић, Олександр Кравчук, يانام, 294 anonymous edits

Image Sources, Licenses and Contributors

Image:Selection sort animation.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:Selection_sort_animation.gif *License*: Public Domain *Contributors*: at en.wikipedia.org

Image:Selection-Sort-Animation.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:Selection-Sort-Animation.gif *License*: GNU Free Documentation License *Contributors*: German, LipeFontoura, Nillerdk

License