Sergio E. Garcia Tapia
Algorithms by Sedgewick and Wayne (4th edition) [SW11]
October 12th, 2024

2.1: Elementary Sorts

Exercise 1. Show, in the style of the example trace with Algorithm 2.1, how selection sort sorts the array `E A S Y Q U E S T I O N`.

Solution.

`i`	`min`	0	1	2	3	4	5	6	7	8	9	10	11
		E	A	S	Y	Q	U	E	S	T	I	O	N
0	1	E	A	S	Y	Q	U	E	S	T	I	O	N
1	1	A	E	S	Y	Q	U	E	S	T	I	O	N
2	6	A	E	S	Y	Q	U	E	S	T	I	O	N
3	9	A	E	E	Y	Q	U	S	S	T	I	O	N
4	11	A	E	E	I	Q	U	S	S	T	Y	O	N
5	10	A	E	E	I	N	U	S	S	T	Y	O	Q
6	11	A	E	E	I	N	O	S	S	T	Y	U	Q
7	7	A	E	E	I	N	O	Q	S	T	Y	U	S
8	11	A	E	E	I	N	O	Q	S	T	Y	U	S
9	11	A	E	E	I	N	O	Q	S	S	Y	U	T
10	10	A	E	E	I	N	O	Q	S	S	T	U	Y
11	11	A	E	E	I	N	O	Q	S	S	T	U	Y
		A	E	E	I	N	O	Q	S	S	T	U	Y

Exercise 2. What is the maximum number of exchanges involving any particular item during selection sort? What is the average number of exchanges involving an item?

Solution. Since the algorithms does exactly n exchanges, the maximum number of exchanges is n. If we have an array of n distinct items, then the maximum is attained when we place the largest item in front, followed by the remaining items in ascending order. Then the largest item is exchanged n times.

Let f_i be the number of times the ith element is involved in an exchange. Since there are n items, with exactly n exchanges, and each exchange involves two items, we conclude that $\sum_{i=1}^{n} f_i = 2n$. Therefore, the average number of exchanges is

$$\text{average \# of exchanges} = \frac{\sum_{i=1}^{n} f_i}{n} = \frac{2n}{n} = 2$$

Exercise 3. Give an example of an array of n items that maximizes the number of times the test `a[j] < a[min]` succeeds (and therefore, `min` gets updated) during the operation of selection sort (Algorithm 2.1).

Solution. The best example I can think of is an array that is sorted in reverse. The first iteration there are n successes, then $n-2$, then, then $n-2$, and so on, until we reach the halfway point of the array. At that point, none of the comparisons will succeed. In essence, each exchange places two items in the final destination at once.

Exercise 4. Show, in the style of the example trace with Algorithm 2.2, how insertion sort sorts the array `E A S Y Q U E S T I O N`.

Solution.

i	j	0	1	2	3	4	5	6	7	8	9	10	11
		E	A	S	Y	Q	U	E	S	T	I	O	N
1	0	A	E	S	Y	Q	U	E	S	T	I	O	N
2	2	A	E	S	Y	Q	U	E	S	T	I	O	N
3	3	A	E	S	Y	Q	U	E	S	T	I	O	N
4	2	A	E	Q	S	Y	U	E	S	T	I	O	N
5	4	A	E	Q	S	U	Y	E	S	T	I	O	N
6	2	A	E	E	Q	S	U	Y	S	T	I	O	N
7	5	A	E	E	Q	S	S	U	Y	T	I	O	N
8	6	A	E	E	Q	S	S	T	U	Y	I	O	N
9	3	A	E	E	I	Q	S	S	T	U	Y	O	N
10	4	A	E	E	I	O	Q	S	S	T	U	Y	N
11	4	A	E	E	I	N	O	Q	S	S	T	U	Y
		E	A	S	Y	Q	U	E	S	T	I	O	N

Exercise 5. For each of the two conditions in the inner `for` loop in insertion sort (Algorithm 2.2), describe an array of n items where that condition is always false when the loop terminates.

Solution. The condition `less(a[j], a[j-1])` is always `false` when the loop terminates for an array whose first element is the smallest element. For the condition that `j > 0`, the loop always terminates when it is `false` for an array that is in reverse order.

Exercise 6. Which method runs faster for an array with all keys identical, selection sort or insertion sort?

Solution. Selection sort is quadratic regardless of the input. Insertion sort is linear for such an array. Therefore, insertion sort is faster.

Exercise 7. Which method runs faster for an array in reverse order, selection sort or insertion sort?

Solution. For such an array, the number of inversions are maximized, so it is the worst-case for insertion sort. As a result, it leads to $\sim n^2/2$ exchanges. Meanwhile, selection sort always requires exactly n exchanges. The number of compares is similar. In this case, selection sort is better.

Exercise 8. Suppose that we use insertion sort on a randomly ordered array where items have only one of three values. Is the running time linear, quadratic, or somewhere in between?

Solution. If all three items have equal frequency, and the array is sorted in reverse, then the number of inversions is quadratic in the array size. A randomly ordered array would have about half as many inversions, and hence would still be quadratic in performance.

Exercise 9. Show, in the style of the example trace with Algorithm 2.3, how shellsort sorts the array `E A S Y S H E L L S O R T Q U E S T I O N`.

Solution. No, this is a very long exercise.

Exercise 10. Why not use selection sort for h-sorting in shellsort?

Solution. Selection sort does not take advantage of the order that shellsort has attained through taking care of inversions at the intermediate stages.

Exercise 11. Implement a version of shellsort that keeps the increment sequence in an array rather than computing it.

Solution. A decided a `Stack` was more natural so I did that instead of an array. See the `com.segarciat.algs4.ch2.sec1.ex11.Shell` class.

Exercise 12. Instrument shellsort to print the number of compares divided by the array length for each increment. Write a test client that tests the hypothesis that this number is a small constant, by sorting arrays of random `Double` values, using array lengths that are increasing powers o f 10, starting at 100.

Solution. See the class `com.segarciat.algs4.ch2.sec1.ex12.ShellCompares`.

Exercise 13. *Deck sort.* Explain how you would put a deck of cards in order by suit (in the order spades, hearts, clubs, diamonds) and by rank within each suit, with the restriction that the cards must be laid out face down in a row, and the only allowed operations are to check the values of two cards and to exchange two cards (keeping them face down).

Solution. I would employ the insertion sort algorithm, which is:

1. Start at the second card, flipping that card and the one immediately next to it to the left.
2. If the suits are different and out of order, exchange them. Move left one pair and continue until this condition is false. Otherwise check the next condition.
3. Otherwise, if the suits are the same and are out of order, exchange them. Move left one pair and flip two cards again, until this condition is false. Otherwise, check the next condition.
4. Otherwise, leave them as is and move on to to the next pair to the right of them. Now move to the third position in the row and start over.
5. ...
6. Move to the i th position in the row and start over.

Exercise 14. *Deque sort.* Explain how you would sort a deck of cards, with the restriction that the only allowed operations are to look at the values of the top two cards, to exchange the top two cards, and to move the top card to the bottom of the deck.

Solution.

1. Inspect the top to wards.
   (a) If they are of different suits and the second card is lower (for example, the top card is hearts and the second is spades), swap them.
   (b) Otherwise, if they are of the same suit and the top card is of higher rank, swap them.
   (c) Otherwise, send the card top card to the bottom of the deck.
2. Repeat until sorted.

Exercise 15. *Expensive exchange.* A clerk at a shipping company is charged with the task of rearranging a number of large crates in order of the time they are to be shipped out. Thus, the cost of compares is very log (just look at the two labels) relative to the cost of exchanges (move the crates). The warehouse is nearly full — there is extra space sufficient to hold any one of he crates, but not two. What sorting method should the clerk use?

Solution. Selection sort. It uses a minimal number of exchanges.

Exercise 17. *Animation.* Add code to `Insertion`, `Selection`, and `Shell` to make them draw the array contents as vertical bars, like the visual traces in this section, redrawing the bars after each pass, to produce an animated effect, ending in a "sorted" picture where the bars appear in the order of height. *Hint*: Use a client like the one in the text that generates random `Double` values, insert calls to `show()` as appropriate in the sort code, and implement a `show()` method that clears the canvas and draws the bars.

Solution. See the classes in the `com.segarciat.algs4.ch2.sec1.ex17` package. The programs `Selection`, `Insertion`, and `Shell` use common functionality from the `SortUtil` class. The assumption is that they are working with random `double` values between 0 and 1.

Exercise 24. *Insertion sort with sentinel.* Develop an implementation of insertion sort that eliminates the `j>0` test in the inner loop by first putting the smallest item into position. Use `SortCompare` to evaluate the effectiveness of doing so. *Note*: It is often possible to avoid an index-out-of-bounds test i this way — the element that enables the test to be eliminated is known as a *sentinel.*

Solution. See the `com.segarciat.algs4.ch2.sec1.ex24.InsertionSentinel` class.

Exercise 25. *Insertion sort without exchanges.* Develop an implementation of insertion sort that moves larger elements to the right one position with one array access per entry, rather than using `exchange()`. Use `SortCompare` to evaluate the effectiveness of doing so.

Solution. See the `com.segarciat.algs4.ch2.sec1.ex25.InsertionHalfExchanges` class.

References

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011. ISBN: 9780321573513.