Sorting Algorithms (I)

2023

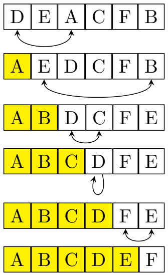
Problem Set and Solutions

The Centre for Education in Mathematics and Computing
Faculty of Mathematics, University of Waterloo
www.cemc.uwaterloo.ca

Selection Sort: Problem Set

1. Sort the following list of letters in alphabetical order using selection sort: **D E A C F B**

Solution



Selection Sort: Problem Set

2. If the unsorted list contains n elements, how many swaps does the algorithm make?

Solution

Every element (except the last) is swapped (possibly with itself). Therefore, the algorithm makes $n-1$ swaps.

Selection Sort: Problem Set

3. How many comparisons does the algorithm make? That is, how many times does it need to compare two elements?

Solution

In the first pass it has to find the minimum of n elements. This takes $n-1$ comparisons. In the second pass it has to find the minimum of $n-1$ elements which takes $n-2$ comparisons. In the final pass it has to find the minimum of 2 elements which takes 1 comparison. Therefore:

$$(n-1)+(n-2)+(n-3)+\ldots+1=\frac{n(n-1)}{2}$$

Selection Sort: Problem Set

4. How does the algorithm perform on data that is already sorted? How does it perform on data that is sorted in reverse?

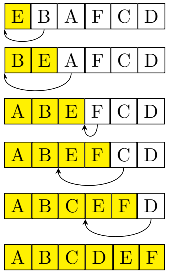
Solution

It performs exactly the same all the time, regardless of the data's composition. Being already (or almost) sorted provides no advantage.

Insertion Sort: Problem Set

1. Sort the following list of letters in alphabetical order using insertion sort: **E B A F C D**

Solution



Insertion Sort: Problem Set

2. If the unsorted list contains n elements, how many elements need to be shifted?

Solution

Every element (except the first) is shifted (possibly in place). Therefore, $n-1$ elements need to be shifted.

Insertion Sort: Problem Set

3. How many comparisons does the algorithm make? That is, how many times does it need to compare two elements?

Solution

The first element shifted is compared with 1 other element. The second element shifted is compared with *at least* 1 and *at most* 2 elements. The $n-1$(th) element shifted is compared with *at least* 1 and *at most* $n-1$ elements. Therefore:

At least $1+1+1+\ldots = n-1$ and
At most $1+2+3+\ldots+(n-1) = \dfrac{n(n-1)}{2}$

Insertion Sort: Problem Set

4. How does the algorithm perform on data that is already sorted? How does it perform on data that is sorted in reverse?

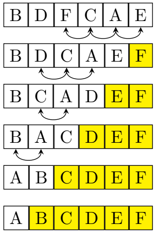
Solution

On data that is already sorted the algorithm will make the minimum number of comparisons. On data that is sorted in reverse the algorithm will make the maximum number of comparisons. Being already (or almost) sorted provides a significant advantage.

Bubble Sort: Problem Set

1. Sort the following list of letters in alphabetical order using bubble sort: **B D F C A E**

Solution



Bubble Sort: Problem Set

2. If the unsorted list contains n elements, how many passes does the algorithm make?

Solution

Every element (except the smallest) needs to be bubbled. Therefore, the algorithm makes $n-1$ passes.

Bubble Sort: Problem Set

3. How many comparisons does the algorithm make?

Solution

The first pass makes $n-1$ comparisons. The second pass makes $n-2$ comparisons. The final pass makes 1 comparison. Therefore:

$$(n-1)+(n-2)+(n-3)+\ldots+1=\frac{n(n-1)}{2}$$

Bubble Sort: Problem Set

4. How many swaps does the algorithm make?

Solution

If the data is already sorted, then no comparison will result in a swap. If the data is sorted in reverse, then every comparison will result in a swap. Therefore:

Between 0 and $\dfrac{n(n-1)}{2}$

Bubble Sort: Problem Set

5. How can the algorithm be improved so that already sorted or nearly sorted data has more of an advantage?

Solution

One possibility is to count the number of swaps made within a pass. If a pass results in 0 swaps, halt the algorithm early.

Challenge: Problem Set

1. If your data consists of only integers, **Radix Sort** is another viable sorting algorithm. Study the example below and explain how the Radix Sort algorithm works.

170	045	075	090	802	024	002	066

170	090	802	002	024	045	075	066

802	002	024	045	066	170	075	090

002	024	045	066	075	090	170	802

Hint: Consider the individual digits of each element.

Challenge: Problem Set

Solution

Radix Sort sorts by digits. During the first pass the elements are sorted by least significant digit (the 1s). During the second pass the elements are sorted by the 10s digit. During the third pass the elements are sorted by the 100s digit. The final pass will sort the elements by most significant digit, and perhaps surprisingly, will result in the entire set of data being sorted in ascending order.

For this algorithm to work correctly, it is very important that in the event of a tie, elements maintain their relative positioning.

Challenge: Problem Set

2. Here is an algorithm to sort four animals from lightest to heaviest:

    - Randomly pick two animals and compare their weights.
    - Compare the weights of the other two animals.
    - Compare the weights of the heaviest animal from each pair. (The “heavies”)
    - Compare the weights of the lightest animal from each pair. (The “lights”)
    - Arrange the animals as follows: lightest of the “lights”, heaviest of the “lights”, lightest of the “heavies”, and heaviest of the “heavies”.

    Unfortunately this algorithm will sometimes fail. When? What is the probability that this algorithm will succeed?

Challenge: Problem Set

Solution

Suppose you start by comparing the two heaviest animals. One of them will end up getting grouped with the “lights” when it is actually the lightest of the “heavies”. Similarly, suppose you start by comparing the two lightest animals. One of them will end up getting grouped with the “heavies” when it is actually the heaviest of the “lights”.

So, out of the 6 different ways to pick the first pair of animals to compare, 2 ways will fail. This means that $\frac{4}{6}$ or $\frac{2}{3}$ of the time the algorithm will succeed.