CS50 Selection Sort

Overview

Sorted arrays are typically easier to search than unsorted arrays. One algorithm to sort is bubble sort. Intuitively, it seemed that there were lots of swaps involved; but perhaps there is another way? **Selection sort** is another sorting algorithm that minimizes the amount of swaps made (at least compared to bubble sort). Like any optimization, everything comes at a cost. While this algorithm may not have to make as many swaps, it does increase the amount of comparing required to sort a single element.

Key Terms

- selection sort
- array
- pseudocode

Implementation

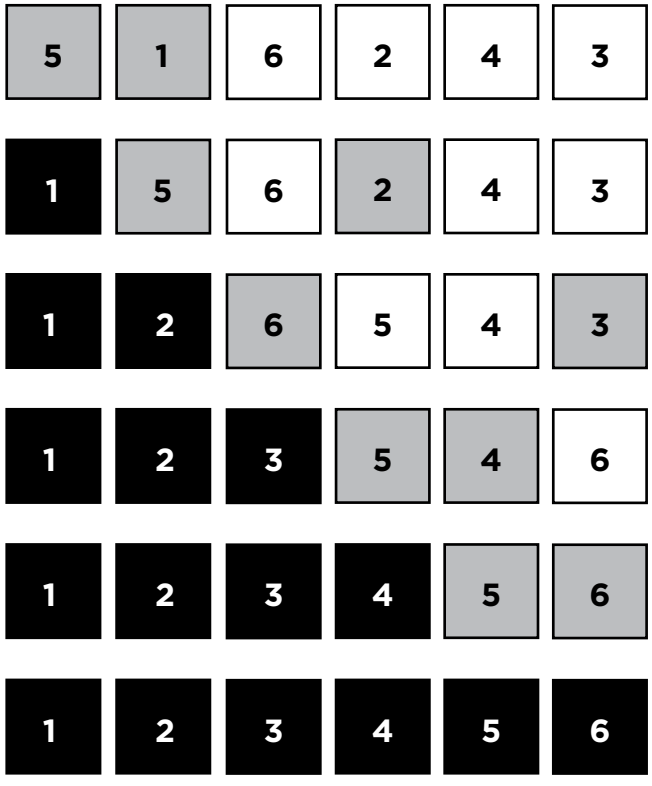
Selection sort works by splitting the array into two parts: a sorted **array** and an unsorted array. If we are given an array of the numbers 5, 1, 6, 2, 4, and 3 and we wanted to sort it using selection sort, our **pseudocode** might look something like this:

```
repeat for the amount of elements in the array
       find the smallest unsorted value
       swap that value with the first unsorted value
```

When this is implemented on the example array, the program would start at `array[0]` (which is 5). We would then compare every number to its right (1, 6, 2, 4, and 3), to find the smallest element. Finding that 1 is the smallest, it gets swapped with the element at the current position. Now 1 is in the sorted part of the array and 5, 6, 2, 4, and 3 are still unsorted. Next is `array[1]`, 5 is our current element and we need to check all elements to the right to check for the smallest (note that by only checking to the right we are only looking at the unsored array). Finding that 2 is the smallest element of the unsorted array we swap it with 5, and so on.

Notice that in the second-to-last iteration, we can clearly see that the array is now sorted, but a computer cannot look at the larger picture like we can. It can only process the information directly in front of it, and so therefore, we continue the process. Once 5 is recognized as the smallest element of the unsorted array, only then can the algorithm be stopped, since the "unsorted" array is of size one and any list of size 1 is necessarily sorted.

Step-by-step process for selection sort



Sorted Arrays

Unlike bubble sort, it is not necessary to keep a counter of how many swaps were made. To optimize this algorithm, it might seem like a good idea to check if the entire array is sorted after every successful swap to avoid what happened in the last two steps of the pseudocode above. This process too, comes at a cost, that is even more comparisons that we have to make. We are guaranteed though, that no matter the order of the original array, a sorted array can be formed after *n-1* swaps, which is significantly fewer than that of bubble sort. In selection sort, in the worst case scenario, n^2 comparions and *n-1* swaps are made. Unfortunately, that's also the case in the best-case scenario!