Computational Thinking with Algorithms - Simple Sorts

Dominic Carr

Atlantic Technological University

dominic.carr@atu.ie

Overview

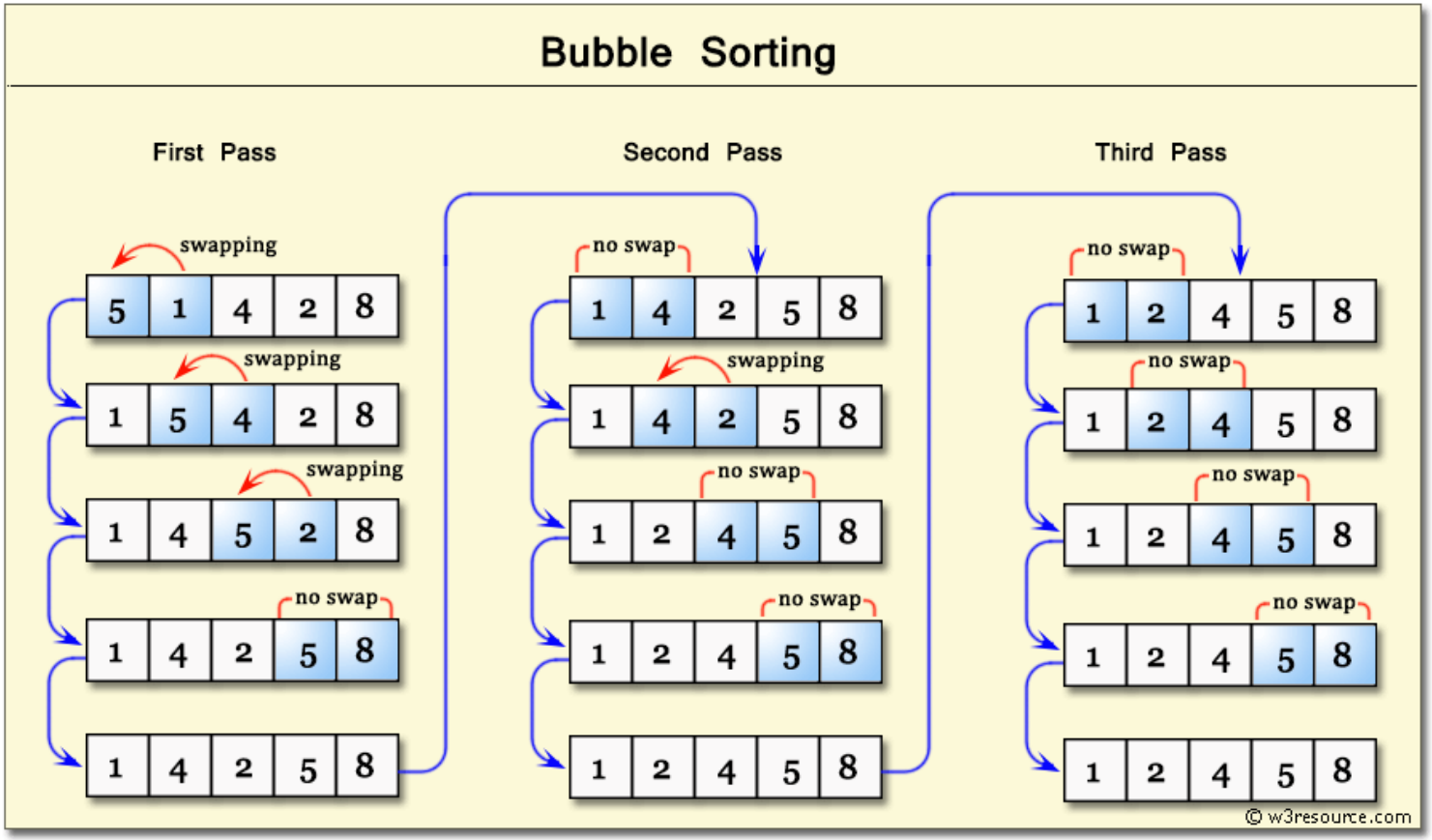
Bubblesort I

- Fairly inefficient sorting algorithm
- Advantage: **simple** algorithm
- At each step, if two adjacent elements are out of order, they are swapped.
- As such smaller values bubble to the start of the array, towards index 0
- ...and larger values bubble to the end of the array, towards index N-1
- Can be flipped, such that the larger values are at the beginning.

Bubblesort II

- Named for the way larger values in a list "bubble up" to the end as sorting takes place
- Bubble Sort was first analysed as early as 1956
- time complexity is n in best case, and n^2 in worst and average cases)
- Comparison based
- In place sorting algorithm (i.e. uses a constant amount of additional working space in addition to the memory required for the input)
- Simple to understand and implement, but it is slow and impractical for most problems even when compared to Insertion Sort
- Can be practical in some cases on data which is nearly sorted

Bubblesort III



Bubblesort IV

Algorithm 1: Bubblesort

```
1   input: array of numbers or anything else sortable
2   output: The function does not return any value
3   side-effects: the array is modified in-place
4
5   function bubbleSort(A : list of sortable items)
6       n := length(A)
7       repeat
8           swapped := false
9           for i := 1 to n-1 inclusive do
10              // if this pair is out of order
11              if A[i-1] > A[i] then
12                  // swap them and remember something changed
13                  swap(A[i-1], A[i])
14                  swapped := true
15              end if
16          end for
17      until not swapped
18  end function
```

Bubblesort V

Listing: Bubble Sort implemented in Java for Integers

```
public void bubblesort(int[] arr){
    System.out.println("In the beginning " + Arrays.toString(arr));
    int n = arr.length;
    boolean swapped = false;
    do {
        swapped = false;
        for (int i = 1; i < n; i++){
            if (arr[i-1] > arr[i]){
                System.out.println("Swapping " + arr[i] + " and " + arr[i-1]);
                int temp = arr[i-1];
                arr[i-1] = arr[i];
                arr[i] = temp;
                swapped = true;
                System.out.println("After the swap: " + Arrays.toString(arr));
            }
        }
    } while (swapped);
    System.out.println("Nothing was swapped, we are done");
}
```

Bubblesort VI

```
In the beginning [5, 1, 4, 2, 8]
Swapping 1 and 5
After the swap: [1, 5, 4, 2, 8]
Swapping 4 and 5
After the swap: [1, 4, 5, 2, 8]
Swapping 2 and 5
After the swap: [1, 4, 2, 5, 8]
Swapping 2 and 4
After the swap: [1, 2, 4, 5, 8]
Nothing was swapped, we are done
[1, 2, 4, 5, 8]
```

Insertion Sort I

Definition

"An sorting algorithm which moves elements **one at a time into the correct position**. The algorithm consists of **inserting one element at a time** into the previously sorted part of the array, moving higher ranked elements up as necessary. To start off, the first (or smallest, or any arbitrary) element of the unsorted array is considered to be the sorted part."

Insertion Sort II

`https://en.wikipedia.org/wiki/Insertion_sort#/media/File:Insertion-sort-example-300px.gif` provides a very good visual example of insertion sort algorithm

Insertion Sort III

Algorithm 1: Pseudocode for Insertion Sort

```
1   input: array of numbers or anything else sortable
2   output: The function does not return any value
3   side-effects: the array is modified in-place
4
5   function insertionSort(array A)
6       i = 1
7       while i < length(A)
8           j = i
9           while j > 0 and A[j-1] > A[j]
10              swap A[j] and A[j-1]
11              j = j - 1
12          end while
13          i = i + 1
14  end while
```

Insertion Sort IV

Listing: Insertion sort for integers in Java

```
public static void insertionSort(int[] arr){
    int i = 1;
    System.out.println("In the beginning " + Arrays.toString(arr));

    while (i < arr.length){
        int j = i;
        while (j>0 && (arr[j-1] > arr[j])){
            System.out.println("Swapping " + arr[j] + " and " + arr[j-1]);
            int temp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = temp;
            System.out.println("After the swap: " + Arrays.toString(arr));
            j--;
        }
        i++;
    }
    System.out.println("In the end " + Arrays.toString(arr));
}
```

Insertion Sort V

```
In the beginning [12, 11, 13, 5, 6]
Swapping 11 and 12
After the swap: [11, 12, 13, 5, 6]
Swapping 5 and 13
After the swap: [11, 12, 5, 13, 6]
Swapping 5 and 12
After the swap: [11, 5, 12, 13, 6]
Swapping 5 and 11
After the swap: [5, 11, 12, 13, 6]
Swapping 6 and 13
After the swap: [5, 11, 12, 6, 13]
Swapping 6 and 12
After the swap: [5, 11, 6, 12, 13]
Swapping 6 and 11
After the swap: [5, 6, 11, 12, 13]
In the end [5, 6, 11, 12, 13]
```

Selection Sort I

Definition

Selection Sort is similar to Insertion Sort. Basically we find the smallest value in the unsorted section of the array, and it put it at the lowest index of the unsorted section of the array (initially none of the array is sorted so this would be index 0).

Selection Sort II

- Comparison based
- In place
- Unstable
- Simple to implement
- Time complexity is n^2 in best, worst and average cases
- Generally gives better performance than Bubble Sort, but still impractical for real world tasks with a significant input size
- In every iteration of Selection Sort, the minimum element (when using ascending order) from the unsorted subarray on the right is picked and moved to the sorted subarray on the left

Selection Sort III

```
1   input: array for sortable items
2   output: no return value
3   side-effects: input array is modified
4
5   function selection Sort(arr: an array of sortable items)
6       n is assigned the length of the array
7
8       for i up to n:
9           minIndex = i
10          for i + 1 up to n
11              if arr[j] < arr[minIndex]
12                  minIndex = j
13          swap(i, minIndex)
```

Selection Sort IV

Listing: Example of Selection Sort (https://en.wikipedia.org/wiki/Selection_sort)

```
64 25 12 22 11 // initial state of the array

11 25 12 22 64 // sorted sublist = {11}

11 12 25 22 64 // sorted sublist = {11, 12}

11 12 22 25 64 // sorted sublist = {11, 12, 22}

11 12 22 25 64 // sorted sublist = {11, 12, 22, 25}

11 12 22 25 64 // sorted sublist = {11, 12, 22, 25, 64}
```

Selection Sort V

Listing: Selection Sort for Integer Arrays in Java

```
public static void selectionSort(int arr[]) {
        int n = arr.length;

        System.out.println("unsorted: " +Arrays.toString(arr));
        for (int i = 0; i < n - 1; i++) {

            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }

            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;

            int[] newArr = Arrays.copyOfRange(arr, 0, i+1);
            System.out.println("Sorted sub list: " + Arrays.toString(newArr));
        }
        System.out.println("sorted: " +Arrays.toString(arr));
    }
```

The End