Lesson:

Bubble Sort

PW SKILLS

Pre-Requisites

- Functions
- Loops
- Arrays

List of Concepts Involved

- Bubble Sort Algorithm
- Bubble Sort Time Complexity
- Bubble Sort Space Complexity
- Worst case scenario in Bubble Sort
- How to optimize the bubble sort in the case of nearly sorted arrays?
- Stable and Unstable sort

As we are now familiar with the concept of Recursion from the previous lesson, we will now expand our knowledge further to learn sorting algorithms.

Topic: Bubble Sort Algorithm

This algorithm works on the principle of swapping adjacent elements if they are not in sorted order and it keeps swapping until the final array becomes sorted. At each pass we keep on checking the adjacent elements and reach the end of the array, this will make the last element of the array sorted and keep repeating this process will result in formation of sorted array at the end part.

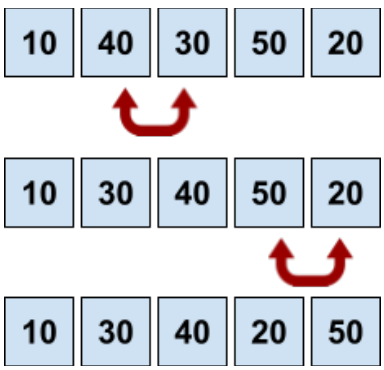
Example:
Array = [10, 40, 30, 50, 20]

In the above example, we keep comparing the adjacent elements and perform the required operations as follows:
First Pass: Check 1 is between 10 and 40, so no need to swap here. The 2nd check is between 40 and 30 so here, we need to swap them. The new array looks like [10,30,40,50,20].

The 3rd check is between 40 and 50, they are in correct order so no need to swap them, and the 4th check is between 50 and 20, and we know that 50 > 20, so we need to swap them.

Array after first pass: [10, 30, 40, 20, 50]

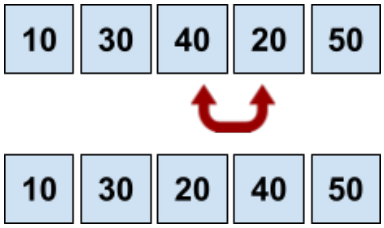
First Pass



Second Pass: In the second pass, we already know that the last element will be the maximum element as all swaps in First pass will move the maximum element to the last index. So we only need to compare elements just one before that only. We start with the 1st check of 10 and 30. They are in order so no need to swap them. Then we check 30 and 40, they are also in order so no need to swap them. The 3rd check will be of 40 and 20, now we know that 40 > 20, so we need to swap them.

The array now becomes: [10, 30, 20, 40, 50].

Second Pass

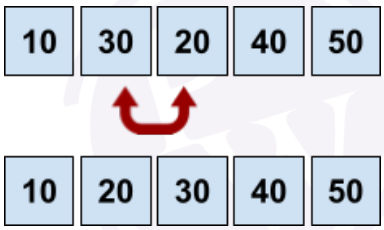


Third Pass: Now we know that the last 2 elements of the array will be at their correct positions in sorted order as 2 passes are already done, so we do not need to check them.

First check will be of 10 and 30, they are already in sorted order so no need to swap them, 2nd check will be of 30 and 20 and we know that 30 > 20, so we need to swap them.

Array after third pass: [10, 20, 30, 40,50]

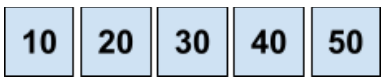
Third Pass



Fourth Pass:
Before starting the fourth pass, the last 3 elements will be at their correct positions as 3 passes are already done, So we do not need to check them. So the first check is 10 and 20, they are already at the correct position so no need to swap them. No need to check further as they are already sorted .

Array after fourth pass: [10, 20, 30, 40, 50]

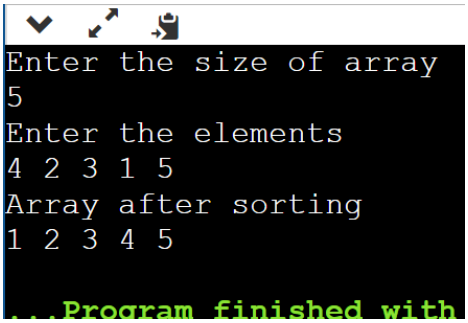
Fourth Pass



This way we have our sorted array at the end.

Bubble Sort Code:
https://pastebin.com/UgtZa2hC

Output



Topic - Bubble Sort Time Complexity:

The time complexity of the following algorithm will be **O(n²)** as at each pass we are traversing the array and swapping adjacent elements whenever needed. Each traversal is O(n) and there are n traversals in total so **O(n * n) = O(n²) in worst case time complexity.**

Topic - Bubble Sort Space Complexity:

It does not require any extra space so space complexity is O(1). Thus it will be an in-place sorting algorithm.

Topic - Maximum number of swaps in worst case in Bubble Sort:

At max, the maximum number of swaps in 1st pass can be (n-1) when everything has to be swapped.

Maximum number of swaps in 2nd pass will be (n-2) as last element will already be maximum after 1st pass.

Maximum number of swaps in 3rd pass will be (n-3) as last 2 elements will already be sorted after 2nd pass.

Similarly in the last pass or (n-1)th pass, the maximum number of swaps can be 1.

Total number of maximum swaps = (n-1) + (n-2) + (n-3) + ... + 1 = **(n*(n-1))/2**

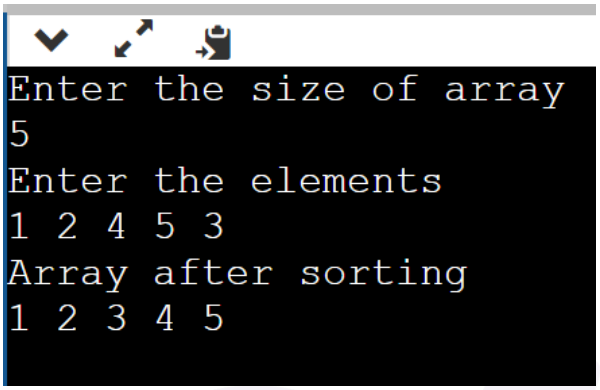
Note: The maximum number of swaps occurs when the array is strictly decreasing in nature.

Topic- how to optimize the bubble sort in case of nearly sorted arrays?

If we can identify that the array is sorted, then execution of further passes should be stopped. This is the optimization over the original bubble sort algorithm in case of nearly sorted arrays.

If you find there is no swapping in a particular pass, it means the array has become sorted, so you should not go for the further passes. For this you can use a flag variable which is set to true before each pass and is made to false whenever a swapping operation is executed.

Code Link
https://pastebin.com/0r8yUC5K

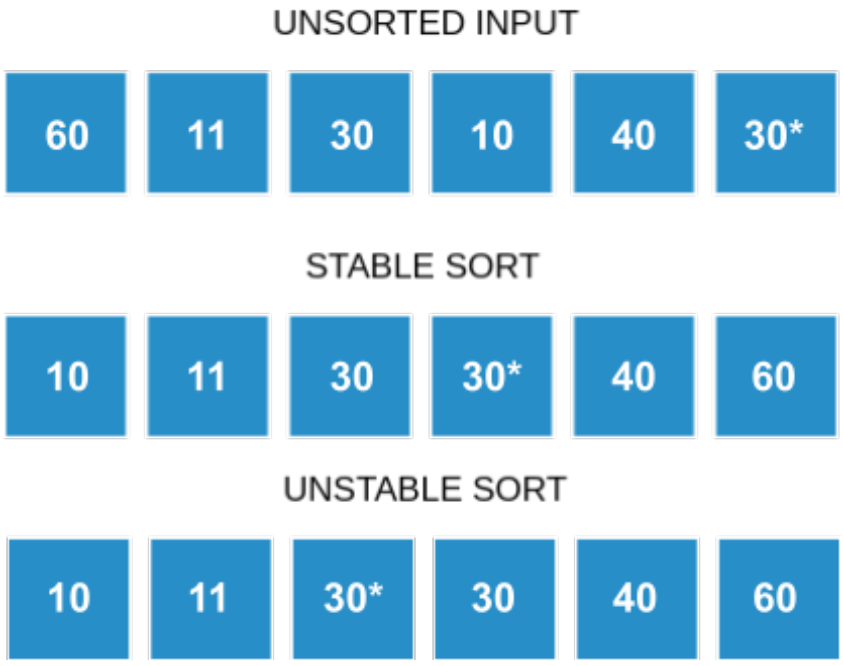


Note- that if all the passes are performed, then our optimized algorithm will in fact perform a little slower than the original one.

Topic -Stable and Unstable Sort

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appeared in the input unsorted array.

For example, Take a look at the picture below. The unsorted array has two elements with value 30. Note the order of both these elements in the stable and unstable sorted arrays.



Is Bubble Sort Stable?
Yes, Bubble Sort is a stable sorting algorithm. We swap elements only when A is less than B. If A is equal to B, we do not swap them, hence relative order between equal elements will be maintained.

Upcoming Class Teasers

- Selection sort algorithm