Bubble Sort

Array = [3, 1, 5, 4, 2]

First Pass

→ Start with first 2 elements
Compare and Sort and Continue

3 1 5 4 2

1 3 5 4 2

Not needed

1 3 4 5 2

1 3 4 2 5

* At the end of the first pass, the largest element will be at the end

Second Pass

1 3 4 2 5

Swap

1 3 2 4 5

In the second pass, the second largest element will come at the second last place

Third pass

1 3 2 4 5

1 2 3 4 5

The third largest element is now at the third from the last place

Bubble sort also known as

1. Sinking Sort
2. Exchange Sort

Note: (only First pass)

Index: 0 1 2 3 4

(1) Arr = 3 1 5 4 2

If $A[j] < A[j-1] \Rightarrow$ Swap

After Swap $[j = j + 1]$
$= 1 + 1 = 2$

(2) 1 3 5 4 2

Is $A[j] > A[j-1]$, No $\rightarrow j = 3$

(3) 1 3 5 4 2 $\quad j = 4$

Is $A[j] > A[j-1]$, Swap $\quad j = Ar.len - 1$

[Reset j]

(4) 1 3 4 5 2

Is $A[j] > A[j-1]$, Swap

(5) 1 3 4 2 5 [Second pass

j

How many times to sort / passes.

→ Since every time we run, it sort the largest in the set

3 1 5 4 2

First pass : sort 5

2nd pass : sort 4

3rd pass : sort 3

4th pass : sort 2

No of passes = (N-1)

This is the counter = i (outer loop)

For Second pass

1 3 4 2 5 (Not need to compare)

1 3 2 4 5 (Not needed to compare)

After every pass, the last elements can be eliminated from the sort process

For third pass

0	1	2	3	4
1	3	2	4	5

only check this part

As this is already sorted

So in third pass, j traverses the until the condition

$$\boxed{j < length - i}$$

(or)

$$\boxed{j <= length - i - 1}$$

[length = no of elements = 5]

Complexity of bubble sort

Space Complexity = O(1) // Constant

This means no extra space is consumed

(example : Copying the Array

This means, In place sorting algorithms (since no extra space is required (or) created)

Time Complexity

Best Case : $O(N)$ ⇒ Sorted

Worst Case : $O(N^2)$ ⇒ Sorted in descending order

(N = no of Comparisons)

As the size is Increasing = No of Comparisons is also Increasing

---

Time Complexity (Cont..)

Best Case

First pass

$i = 0$

1 2 3 4 5

Note: when j never swaps for a value of i [i^{th} pass], that means the array is sorted

Cue : To exit the program

Best Case Comparisons : N

Its actually N-1, but in time Complexity constants are ignored

Worst Case

$i = 0$

0	1	2	3	4
5	4	3	2	1
4	5	3	2	1
4	3	5	2	1
4	3	2	5	1
4	3	2	1	5

Total no of swaps $= N-1$

$= 4$ Swap

$i = 1$

4	3	2	1	5
3	4	2	1	
3	2	4	1	
3	2	1	4	

Ignore = already sorted

Total swaps $= 3$

$i = 2$

3	2	1	4
2	3	1	
2	1	3	

Total swaps $= 2$

$i = 3$

2	1	3
1	2	

Total swaps $= 1$

$i = 4$ [Not Needed] $\boxed{1}$ $\boxed{2}$ $\boxed{X}$

The outer loop only ran $N-1$ times, $\overbrace{0,1,2,3}^{i}$
$= 4$ times only

Total Comparisons (All passes included)

$$= (N-1) + (N-2) + (N-3) + (N-4)$$

$$= 4N - (1+2+3+4)$$

$$= 4N - \left(\frac{N * (N+1)}{2}\right)$$

$$= \frac{8N - N^2 - N}{2} = \frac{7N - N^2}{2}$$

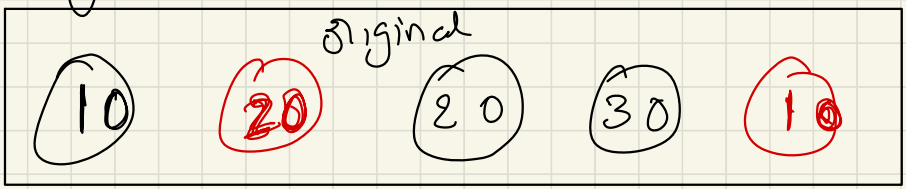
$7N$ = less dominating terms

2 = Constants

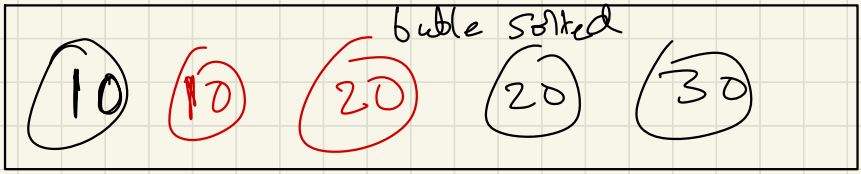
(Time Complexity Lecture) removed

$$= \frac{7\cancel{N} - N^2}{\cancel{2}} = O(N^2)$$

⇒ Bubble Sort is a Stable Sorting algorithm



Apply bubble sort



In the original array, the black ball of "10" come before the red ball of 10. After the bubble sort, this order is still maintained.

Say for example, the sort results are below