Sorting in Java

Bubble Sort

7	8	3	1	2

It is a sorting algorithm where there shall be (n-1) comparisons, provided there are 'n' elements present.

In each iteration, every elements (starting from the first) is compared with it's next element and the largest element is pushed to the right end.

with every operation, the comparable positions are reduced because the largest element in the previous iteration is already pushed to the right end.

Step 1- 7, 3, 8, 1, 8, 2, 8 → | 7 | 3 | 1 | 2 | 8 |

Step 2 → 3, 7, 1, 7, 2, 7, 8 → | 3 | 1 | 2 | 7 | 8 |

Step 3 → 1, 3, 2, 3, 7, 8 → | 1 | 2 | 3 | 7 | 8 |

Step 4 → 1, 2, 3, 7, 8 → | 1 | 2 | 3 | 7 | 8 |

So, there are 5 elements, and we made a total no. of (5-1) = 4 comparisons.

Also, after every iteration, the last position is sorted and hence not touched in the next operation. So, after step 1, the last position is sorted. After Step 2, the 2nd last position is also sorted and so on.

Time complexity is $O(n^2)$, which is not good compared to other sorting techniques/algorithms.

* time complexity = $O(n^2)$.
on asymptotic notations, we ignore the constants, so, it is $O(n^2)$.

code :-

```
class bubbleSort
{
Public static void printArray(int arr[]){
    for (int i=0; i< arr.length ; i++)
      { s.o.p( arr[i] + " ");
      }
} s.o.pln(" ");
Public static void main (String[] args)
{
int arr[] = {7, 8, 3, 1, 2};
for (int i=0 ; i< arr.length-1; i++) // n-1
{
  for(int j=0; j< arr.length - i - 1; j++) // innerloop
  {
    if (arr[j] > arr[j+1])
    {
      int temp = arr[j];   // Swap
      arr[j] = arr[j+1];
      arr[j+1] = temp;
    }}
  }
  printArray(arr);
}}
```

* outer loop runs for (n-1) times.
* inner loop runs for n-1, n-2, n-3, and so on.
ignoring constants, we get $O(n^2)$.

Selection Sort

In this sorting technique, with every iteration, swapping happens only once per iteration. The smallest element is found out and pushed to the left. Then, the 2nd smallest element is found, and pushed to the 2nd last and so on.

Given :- | 7 | 8 | 3 | 1 | 2 |

Step 1 :- i=0 (we assume that smallest element is at i=0).

| 7 | 8 | 3 | 1 | 2 |

= | 1 | 8 | 3 | 7 | 2 |

(i=0 element is compared with all other elements & smallest pushed to the left)

only 1 swap per iteration

Step 2 :-

i=1 (we assume that smallest is at i=1)

| 1 | 8 | 3 | 7 | 2 |

= | 1 | 2 | 3 | 7 | 8 |

i=1 → '8' compared to all other elements, and smallest = '2' pushed to the 2nd left position at i=1.

Step 3 :-

| 1 | 2 | 3 | 7 | 8 |

Now i=3th element compared with right side elements and no swapping happened as i=3th itself is the smallest, so, | 1 | 2 | 3 | 7 | 8 |

Similarly in step 4 & 5, i=4th and i=5th element is compared and swapped if found a smaller element.

112-114

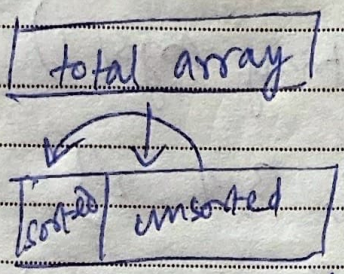
Code :-

```
public static void main(String args[])
{ int arr[] = { 7, 8, 3, 1, 2};
for(int i=0; i<arr.length-1; i++)
{ int smallest = i;
for(int j= i+1; j < arr.length; j++)
{
    if (arr[smallest] > arr[j]) {
        smallest = j;
    }
}
int temp = arr[smallest];
arr[smallest] = arr[i];
arr[i] = temp;
}
```

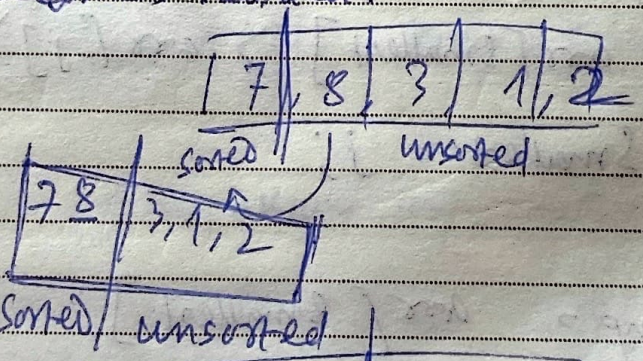
* Time complexity = $O(n^2)$.

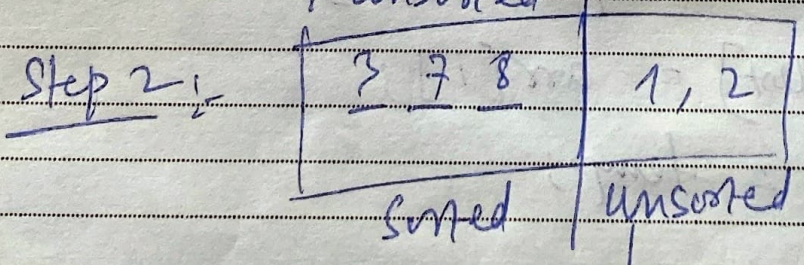
The inner loop runs n, n-1, n-2 times ...
and the outer loop runs approx. n times.
We can say that it's like an A.P series.

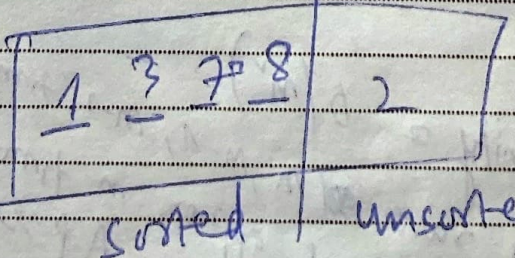
Insertion Sort

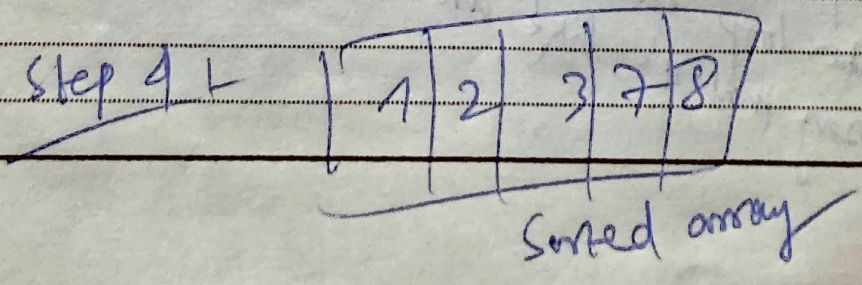


In Insertion sort, we divide the whole array in 2 parts → sorted and unsorted. We keep on adding 1 by 1 element from the unsorted array to the sorted part. In this manner, the sorted array keeps on getting larger and the unsorted array keeps on getting smaller. So, at the end, the entire array becomes sorted.









Code :-

```
Public static void main (String args[])
{
    int arr = {7, 8, 3, 1, 2};
// insertion sort
for(int i = 1; i < arr.length; i++) {
    int current = arr[i];
    int j = i-1;
while (j>=0 && current < arr[j])
    {
        arr[j+1] = arr[j];
        j--;
    }
// Placement
        arr[j+1] = current;
}
```

Note :- Time complexity of insertion sort is also $O(n^2)$.

Apart from Bubble Sort, Selection Sort & insertion Sort there are quick Sort & merge sort, where the time complexity is $O(n \log n)$.