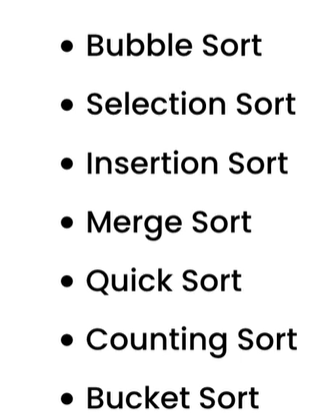
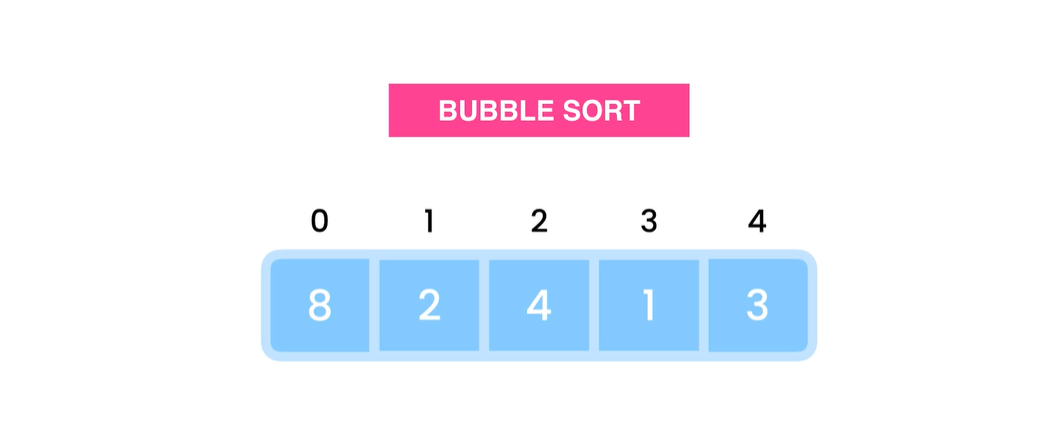
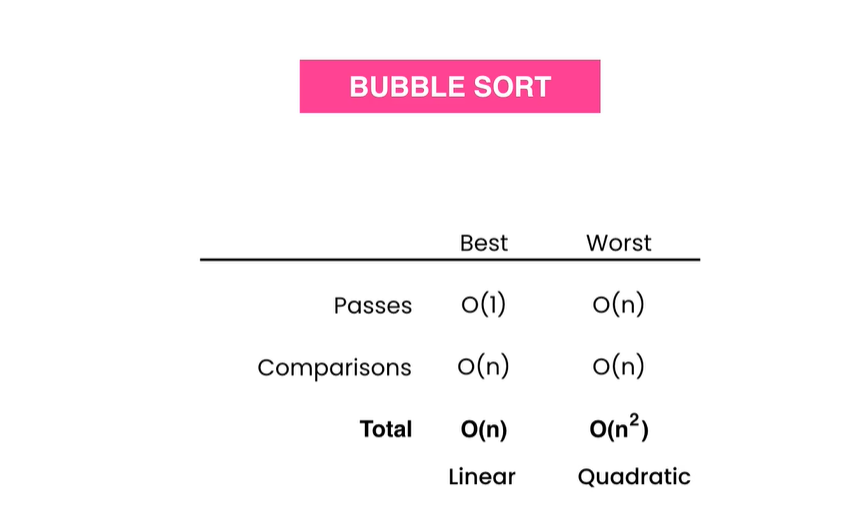
**Sorting Algorithms**



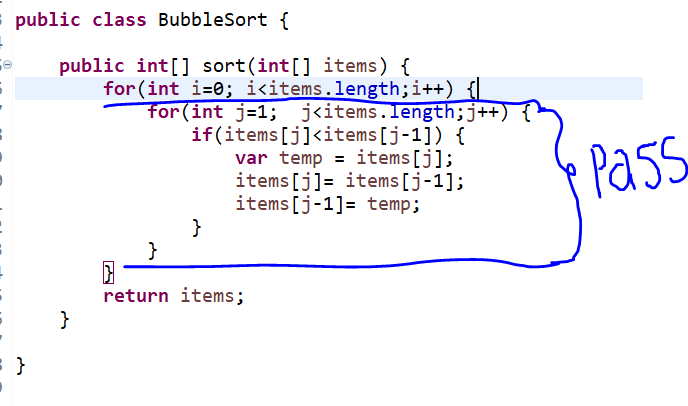
**Bubble Sort:**

This is the simplest of all sorting algorithms. Lets say we have an array integers and we want to sort them increasing order. With Bubble sort, we scan all the items and if they are out of order then we swap them. We start of the items at the indices 0 and 1, if right item is less than left item then we swap them. We continue for the next two indices 1, 2 and so on until we do for the last two items. Now it completes on pass or iteration. We need multiple passes to finish sorting. After each pass one largest item will bubble up and go to the far right.



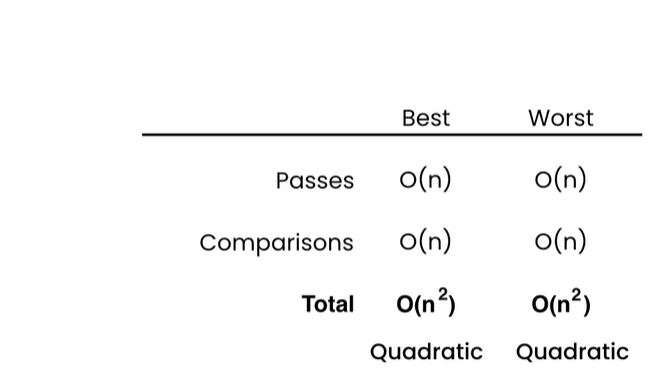


Time complexity for is calculated using number of passes used and comparisons. In best case scenario, number of passes is O(1) and comparisons O(n) so total is O(n). In the worst case scenario number of passes is O(n) and comparisons O(n) so total O(n2).



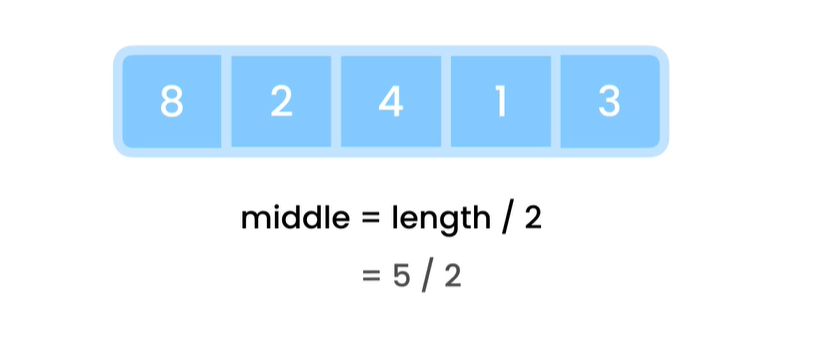
**Selection Sort:**

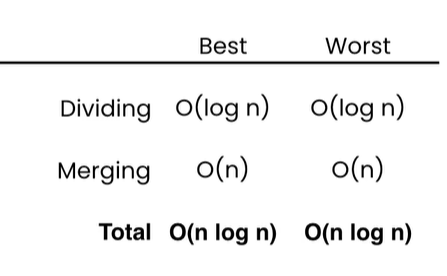
In selection sort, we have to find the minimum number in the given list of numbers, then swap it with the number in index 0. Now we are settled with the value in index 0, we have to consider the minimum value in the rest of array and swap it with number in index 1. Like this swap continuous for the rest of indices and minimum values.



**Merge Sort:**

The idea of merge sort is to break down lists into smaller sub lists, sort them and then merge them all to get the sorted list. We start this by dividing the array into half. We find the index of middle element by dividing array length by 2. Now we will have two new arrays with divided array items. We will divide these arrays further until we cannot divide them anymore. Now we will start merging them back by sorting. This is basically a divide and conquer algorithm.





Merge sort is faster than bubble sort, insertion sort, selection sort. But merge sort take more space.



**public** **class** MergeSort {

**public** **void** sort(**int**[] array) {

**if**(array.length<2) **return**;

// Divide this array into half

**var** middle = array.length/2;

**int**[] left = **new** **int**[middle];

**for**(**int** i=0;i<middle;i++) {

left[i]=array[i];

}

**int**[] right = **new** **int**[array.length-middle];

**for**(**int** i=middle;i<array.length;i++) {

right[i-middle]=array[i];

}

//sort each half

sort(left);

sort(right);

//merge result

merge(left, right, array);

System.***out***.println(Arrays.*toString*(array));

}

**private** **void** merge(**int**[] left, **int**[] right, **int**[] result) {

**int** i=0,j=0,k=0;

**while**(i<left.length && j<right.length) {

**if**(left[i] <= right[j])

result[k++] = left[i++];

**else**

result[k++] = right[j++];

}

**while**(i<left.length) {

result[k++]=left[i++];

}

**while**(j<right.length) {

result[k++]=right[j++];

}

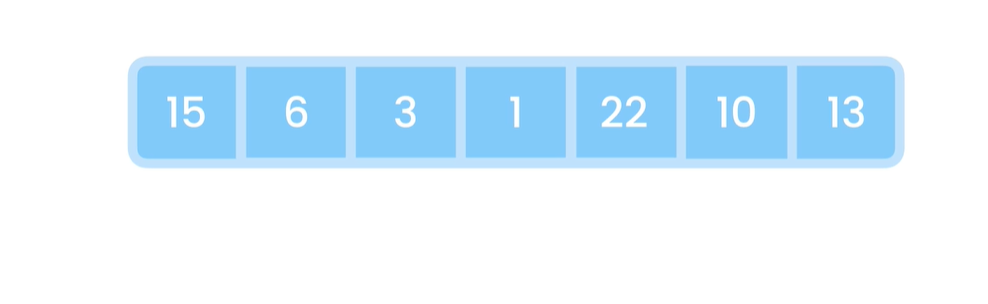
}

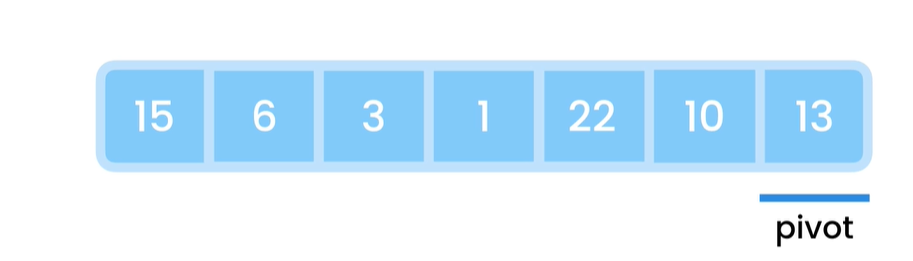
}

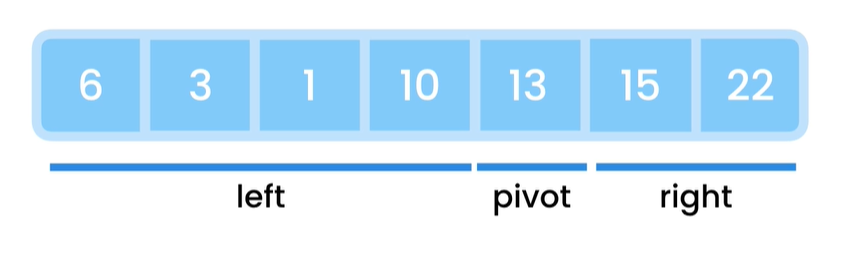
**Quick Sort:**

Quick sort algorithm is the most widely used algorithm and it takes less space than merge sort for sorting.

We start of with an item called pivot and the we rearrange the rest of items into two partitions with items in left are less than pivot and items in the right are greater than pivot. Now after this pivot will be in the correct index position. So now we pick last item of left partition as pivot and do the same until the end. Repeat the same for right partition as well. Eventually we will have sorted items.







**Search Algorithms**

**String Manipulations**