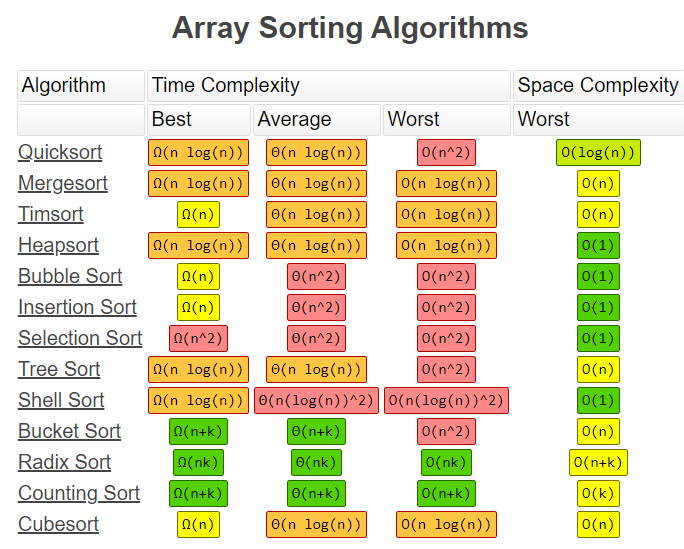
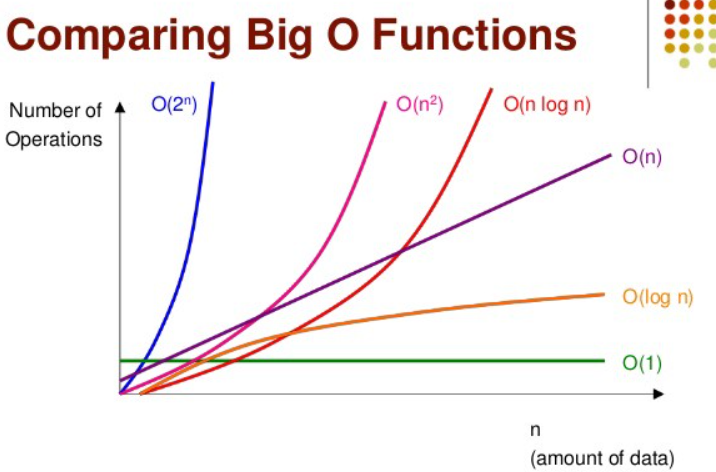
# **SORTING ALGHORITHM**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **NAME** | **BEST** | **AVERAGE** | **WORST** | **SPACE COMPLEXITY** | **IN-PLACE** | **STABLE** | **ONLINE** |
| **BUBBLE SORT** | Ω(n2) | Θ (n2) | O(n2) | O(1) | Yes |  |  |
| **SELECTION SORT** | Ω(n2) | Θ (n2) | O(n2) | O(1) | Yes |  |  |
| **INSERTION SORT** | Ω(1) | Θ (n2) | O(n2) | O(1) | Yes | Yes | Yes |
| **SHELL SORT** |  |  |  |  |  |  |  |
| **MERGE SORT** | Ω(nlog(n)) | Θ (nlog(n)) | O(nlog(n)) | O(n) | No | Yes |  |
| **QUICK SORT** |  |  |  |  | Yes |  |  |





# **STABLE VS UNSTABLE SORTING**

**Stable sorting** - does not change the order of ‘like’ elements in the array

# 

# **BUBBLE SORT**

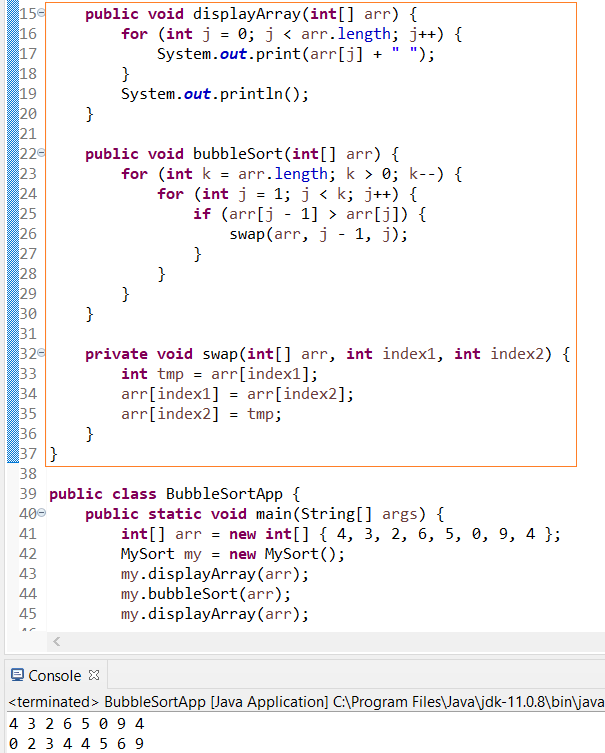
**Bubble sort** – the slowest sorting technique. It compares adjacent elements and move max value to the right

**+:**

* Simple. It is used in educational purposes
* In-place – does not use a lot of extra space

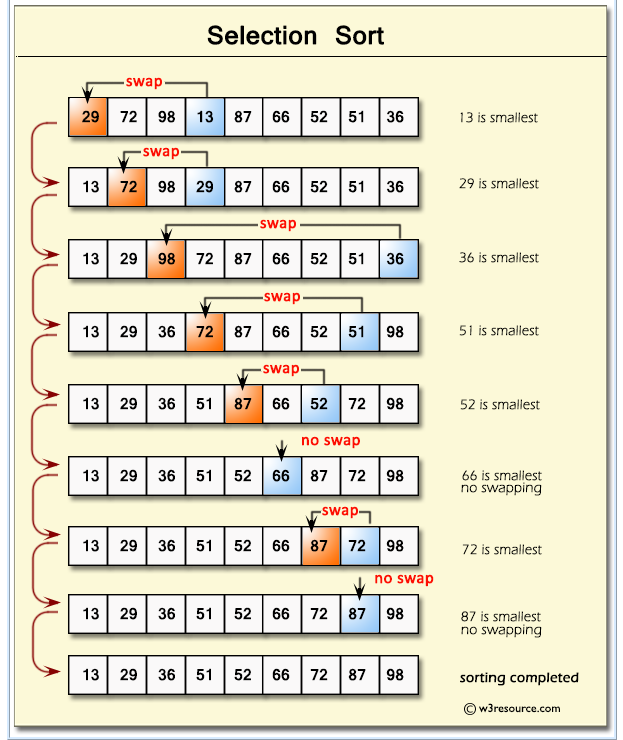
**-:**

* Slow. Best, Average and Worst time complexity is quadratic
* Almost nobody use it



# **SELECTION SORT**

**Selection sort** – it is also slow sorting technique. It find the lowest element in array an move to the left It can be useful in application when an operation of swapping is an expensive and it can minimize a number of swaps

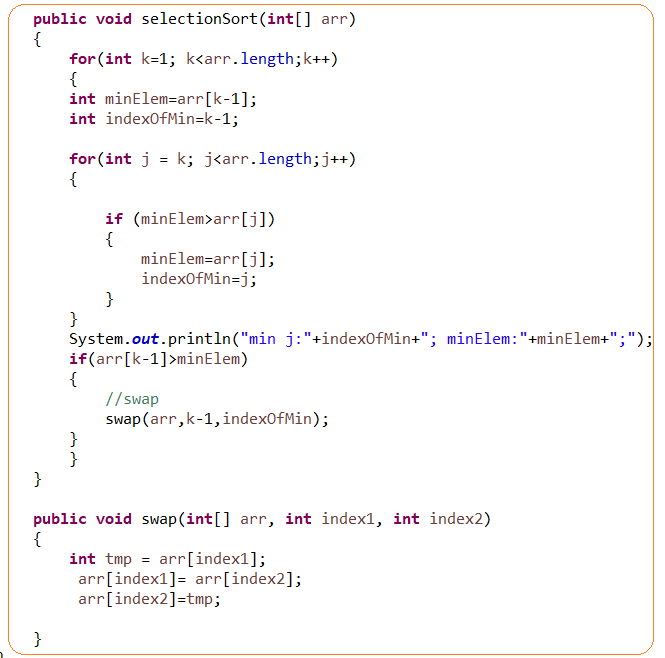


**+:**

* It can be useful in application when an operation of swapping is expensive and it can minimize a number of swaps

**-:**

* O(n2) time complexity



# **INSERTION SORT**

Insertion sort – it

**+:**

* Only efficient for small data sets
* The most efficient through slowest techniques like bubble and selection sort
* O(n2) – is the worst time complexity. It can happen when an array in a reverse order

# **MERGE SORT**

Merge Sort – is a divide and conquer algorithm.

The idea is:

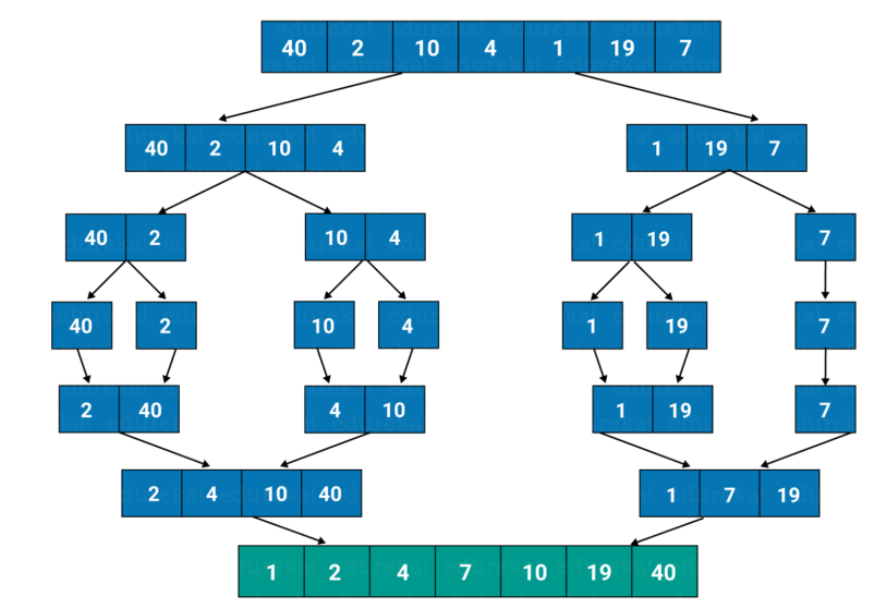
1. **Divide** (split): In this step, the input array is divided into 2 halves, the pivot is the midpoint of the array. This step is carried out recursively for all the half arrays until there are no more half arrays to divide further.
2. **Conquer** (sort and merge): In this step, we sort and merge the divided arrays from bottom to top and reach towards our sorted array.

**+:**

* Best, average and worst time complexity is the same: Ω(nlog(n)), Θ (nlog(n)), O(nlog(n))
* Powerful technique for larger datasets
* stable

**-:**

* it requires additional space to store splitted arrays. It can be an issue for large data sets



# **QUICKSORT**

QuickSort – is divide and conquer algorithm.

* Worst-case time complexity for quicksort is O(n²), although this is an algorithm that rarely falls into its worst-case performance, especially with minor amounts of customization. Typically the Big-O for quicksort is O(n log n). However, a worst case of O(n²) is a knock against it.
* It ironically named because it’s really not even the fastest

It chooses the [pivot] point and partitioning the collection of data around the pivot

**Pivot** – is an arbitrary element.

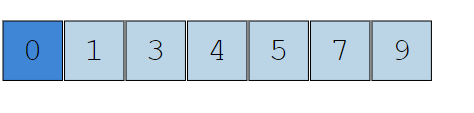
Pivot element can be chosen as

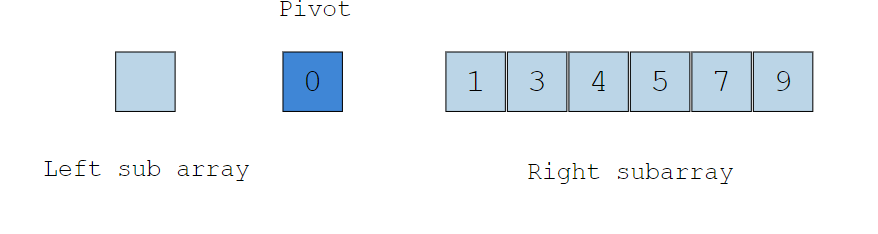
* First element
* Last element
* Medium
* Randomly

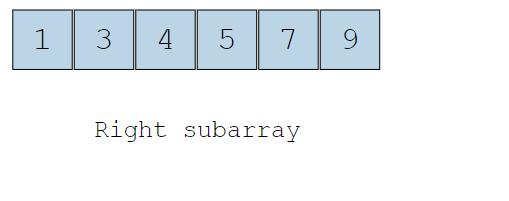
Note: poor chosen element provides the worst time complexity of algorithm as O(n)

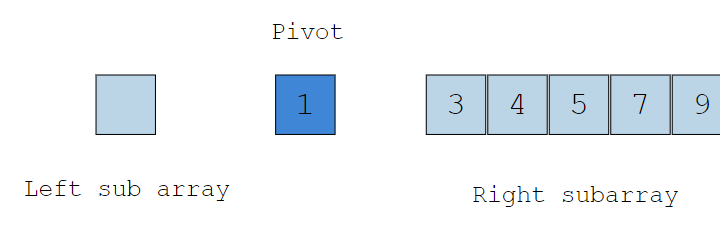
There are two worst cases:

* You sort already sorted array and choose as a pivot the first elemnt
* A lot of same elements









Partition – array of data is divided into 2 parts (recursively repeats)

+:

-:

O(n2) – is the worst time complexity. It cat happen when a pivot element is chosen badly