[**O(n2) algorithms 2**](#_Toc3466019)

[**Bubble Sort 2**](#_Toc3466020)

[**Selection Sort 3**](#_Toc3466021)

[**Insertion Sort 4**](#_Toc3466022)

[**O(n log n) algorithms 5**](#_Toc3466023)

[**Mergesort 5**](#_Toc3466024)

**O(n2) algorithms**

**Bubble Sort**

The algorithm works by comparing each item in the list with the item next to it, and swapping them if required. In other words, the largest element has bubbled to the top of the array. The algorithm repeats this process until it makes a pass all the way through the list without swapping any items.

void bubbleSort(int ar[])

{

for (int i = (ar.length - 1); i >= 0; i--)

{

for (int j = 1; j ≤ i; j++)

{

if (ar[j-1] > ar[j])

{

int temp = ar[j-1];

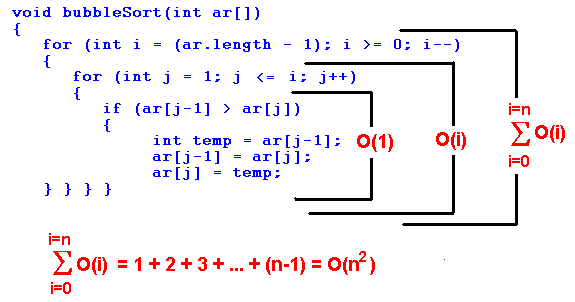
ar[j-1] = ar[j];

ar[j] = temp;

} } } }

**Example.** Here is one step of the algorithm. The largest element - 7 - is bubbled to the top:

**7, 5**, 2, 4, 3, 9  
5, **7, 2**, 4, 3, 9  
5, 2, **7, 4**, 3, 9  
5, 2, 4, **7, 3**, 9  
5, 2, 4, 3, **7, 9**  
5, 2, 4, 3, 7, 9



**Selection Sort**

The algorithm works by selecting the smallest unsorted item and then swapping it with the item in the next position to be filled.

The selection sort works as follows: you look through the entire array for the smallest element, once you find it you swap it (the smallest element) with the first element of the array. Then you look for the smallest element in the remaining array (an array without the first element) and swap it with the second element. Then you look for the smallest element in the remaining array (an array without first and second elements) and swap it with the third element, and so on. Here is an example,

void selectionSort(int[] ar){

for (int i = 0; i ‹ ar.length-1; i++)

{

int min = i;

for (int j = i+1; j ‹ ar.length; j++)

if (ar[j] ‹ ar[min]) min = j;

int temp = ar[i];

ar[i] = ar[min];

ar[min] = temp;

} }

**Example.**

**29**, 64, 73, 34, **20**,   
20, **64**, 73, 34, **29**,   
20, 29, **73**, **34**, 64   
20, 29, 34, **73**, **64**   
20, 29, 34, 64, 73

The worst-case runtime complexity is O(n2).

**Insertion Sort**

To sort unordered list of elements, we remove its entries one at a time and then insert each of them into a sorted part (initially empty):

void insertionSort(int[] ar)

{

for (int i=1; i ‹ ar.length; i++)

{

int index = ar[i]; int j = i;

while (j > 0 && ar[j-1] > index)

{

ar[j] = ar[j-1];

j--;

}

ar[j] = index;

} }

**Example.** We color a sorted part in green, and an unsorted part in black. Here is an insertion sort step by step. We take an element from unsorted part and compare it with elements in sorted part, moving form right to left.

29, 20, 73, 34, 64   
**29**, 20, 73, 34, 64   
**20, 29**, 73, 34, 64   
**20, 29, 73**, 34, 64   
**20, 29, 34, 73**, 64   
**20, 29, 34, 64, 73**

Let us compute the worst-time complexity of the insertion sort. In sorting the most expensive part is a comparison of two elements. Surely that is a dominant factor in the running time. We will calculate the number of comparisons of an array of N elements:

we need 0 comparisons to insert the first element  
we need 1 comparison to insert the second element  
we need 2 comparisons to insert the third element  
...  
we need (N-1) comparisons (at most) to insert the last element

Totally,

1 + 2 + 3 + ... + (N-1) = O(n2)

The worst-case runtimecomplexity is O(n2).What is the best-case runtime complexity? O(n). The advantage of insertion sort comparing it to the previous two sorting algorithm is that insertion sort runs in linear time on nearly sorted data.

## O(n log n) algorithms

### Mergesort

Merge-sort is based on the divide-and-conquer paradigm. It involves the following three steps:

* Divide the array into two (or more) subarrays
* Sort each subarray (Conquer)
* Merge them into one (in a smart way!)

**Example**. Consider the following array of numbers

**27 10 12 25 34 16 15 31**

divide it into two parts

**27 10 12 25 34 16 15 31**

divide each part into two parts

**27 10 12 25 34 16 15 31**

divide each part into two parts

**27 10 12 25 34 16 15 31**

merge (cleverly-!) parts

**10 27 12 25 16 34 15 31**

merge parts

**10 12 25 27 15 16 31 34**

merge parts into one

**10 12 15 16 25 27 31 34**

How do we merge two sorted subarrays? We define three references at the front of each array.