* Bubble sort=Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
* **package** shorting;
* **publicclass** Bouble\_short {
* **publicstaticvoid** printArray(**int**arr[]) {
* **for**(**int**i =0; i<arr.length;i++) {
* System.***out***.print(arr[i]+ " ");
* }
* System.***out***.println();
* }
* **publicstaticvoid** main(String args[]) {
* **int**arr[] = {4,5,15,7,4,3};
* **for**(**int**i =0; i<arr.length - 1; i++) {
* **for** (**int**j= 0 ; j<arr.length-i-1 ; j++) {
* **if**(arr[j]>arr[j+1]) {
* //swap
* **int**temp = arr[j];
* arr[j] = arr[j+1];
* arr[j+1]= temp;

* }
* }
* }
* *printArray*(arr);
* }
* }
* Selection sort= This algorithm is called selection sort **because it repeatedly selects the next-smallest element and swaps it into place**.
* Time complexity of selection sort is O(n^2)

**package** shorting;

**publicclass** Selection\_sort {

**publicstaticvoid** printArray(**int**arr[]) {

**for**(**int**i = 0 ; i<arr.length;i++) {

System.***out***.print(arr[i] + " ");

}

System.***out***.println();

}

**publicstaticvoid** main(String args[]) {

**int**arr[] = {8,4,1,9,0,9,1,3,};

**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;

}

*printArray*(arr);

}

}

Output = 0 1 1 3 4 8 9 9

* Insertion Sort = Insertion sort is a sorting algorithm that **places an unsorted element at its suitable place in each iteration**.

**package** shorting;

**publicclass** Insertion\_sort {

**publicstaticvoid** printArray(**int**arr[]) {

**for**(**int**i= 0 ; i<arr.length; i++) {

System.***out***.print(arr[i]+ " ");

}

System.***out***.println();

}

**publicstaticvoid** main(String args[]) {

**int**arr[] = {7,4,15,1,8,2};

//short

**for**(**int**i = 1; i<arr.length ; i++) {

**int**temp = arr[i];

**int**j = i-1;

**while**(j>=0 &&temp<arr[j]) {

arr[j+1] = arr[j];

j--;

}

arr[j+1] = temp;

}

*printArray*(arr);

}

}

Out put = 1 2 4 7 8 15

| **Algorithm** | **Time Complexity** | | | **Space Complexity** |
| --- | --- | --- | --- | --- |
|  | **Best** | **Average** | **Worst** | **Worst** |
| [Selection Sort](http://geeksquiz.com/selection-sort/) | Ω(n^2) | θ(n^2) | O(n^2) | O(1) |
| [Bubble Sort](http://geeksquiz.com/bubble-sort/) | Ω(n) | θ(n^2) | O(n^2) | O(1) |
| [Insertion Sort](http://geeksquiz.com/insertion-sort/) | Ω(n) | θ(n^2) | O(n^2) | O(1) |
| [Heap Sort](http://geeksquiz.com/heap-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) | O(1) |
| [Quick Sort](http://geeksquiz.com/quick-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) | O(n) |
| [Merge Sort](http://geeksquiz.com/merge-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) | O(n) |
| [Bucket Sort](https://www.geeksforgeeks.org/bucket-sort-2/) | Ω(n +k) | θ(n +k) | O(n^2) | O(n) |
| [Radix Sort](https://www.geeksforgeeks.org/radix-sort/) | Ω(nk) | θ(nk) | O(nk) | O(n + k) |
| [Count Sort](https://www.geeksforgeeks.org/counting-sort/) | Ω(n +k) | θ(n +k) | O(n +k) | O(k) |
| [Shell Sort](https://www.geeksforgeeks.org/shellsort/) | Ω(n) | θ(n log(n)) | O(n log(n)) | O(1) |
| [Tim Sort](https://www.geeksforgeeks.org/timsort/) | Ω(n) | θ(n log(n)) | O(n log (n)) | O(n) |
| [Tree Sort](https://www.geeksforgeeks.org/tree-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) | O(n) |
| [Cube Sort](https://www.geeksforgeeks.org/sort-the-array-according-to-their-cubes-of-each-element/) | Ω(n) | θ(n log(n)) | O(n log(n)) | O(n) |

* Merge sort

Merge sort is **one of the most efficient sorting algorithms**. It is based on the divide-and-conquer strategy. Merge sort continuously cuts down a list into multiple sublists until each has only one item, then merges those sublists into a sorted list.

**package** shorting;

**publicclass** Merge\_sort {

**publicstaticvoid** conquer(**int**arr[], **int**start , **int**mid,**int**end) {

**int**merger[] = **newint** [end - start+1];

**int**idx1 = start;

**int**idx2 = mid +1;

**int**x = 0;

**while**(idx1<=mid&&idx2<=end) {

**if**(arr[idx1]<=arr[idx2]) {

merger[x++] = arr[idx1++] ;

}**else** {

merger[x++] = arr[idx2++];

}

}

**while**(idx1<=mid) {

merger[x++]= arr[idx1++];

}

**while**(idx2<= end) {

merger[x++] = arr[idx2++];

}

**for**(**int**i =0,j=start;i<merger.length;i++,j++) {

arr[j]=merger[i];

}

}

**publicstaticvoid** divide(**int**arr[],**int**start ,**int**end) {

**if**(start>= end) {

**return**;

}

**int**mid = start+(end-start)/2;

*divide*(arr,start, mid);

*divide*(arr,mid+1,end);

*conquer*(arr,start,mid,end);

}

**publicstaticvoid** main(String args[]) {

**int**arr[] = {6,3,45,7,4,6};

**int**n = arr.length;

*divide*(arr,0,n-1);

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

System.***out***.print(arr[i]+ " ");

}

System.***out***.println();

}

}

Out put :: 3 4 6 6 7 45

* Quick sort

Quicksort is **a popular sorting algorithm that is often faster in practice compared to other sorting algorithms**. It utilizes a divide-and-conquer strategy to quickly sort data items by dividing a large array into two smaller arrays.

**package** shorting;

**publicclass** Quick\_sort {

**publicstaticint** partition(**int**arr[],**int**low,**int**high) {

**int**pivot = arr[high];

**int**i = low-1;

**for**(**int**j=low;j<high;j++) {

**if**(arr[j]<pivot) {

i++;

**int**temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

i++;

**int**temp = arr[i];

arr[i]= pivot;

arr[high]= temp;

**return**i;

}

**publicstaticvoid** quickSort(**int**arr[],**int**low, **int**high) {

**if**(low<high) {

**int**pidx = *partition*(arr,low,high);;

*quickSort*(arr,low,pidx-1);

*quickSort*(arr,pidx+1,high);;

}

}

**publicstaticvoid** main(String args[]) {

**int** [] arr = {4,5,6,3,5,6};

**int**n = arr.length;

*quickSort*(arr,0,n-1);

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

System.***out***.print(arr[i] + " ");

}

System.***out***.println();

}

}

Output is:: 3 4 5 5 6 6