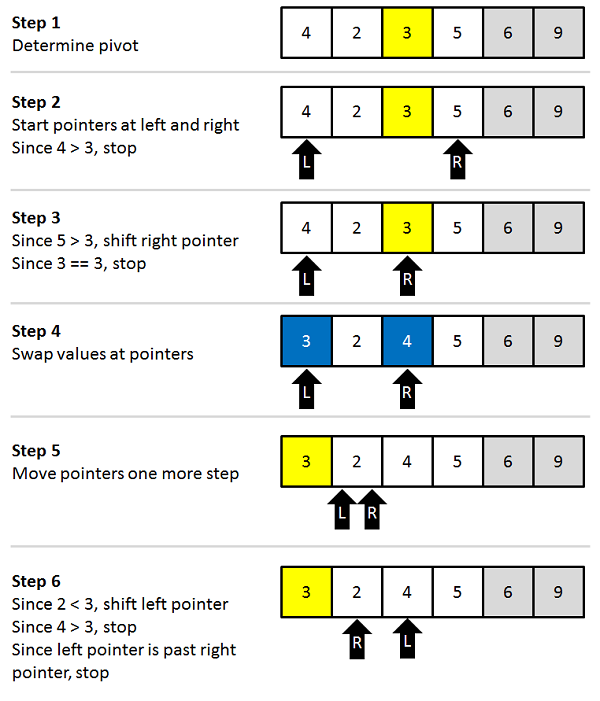
Sort

<http://java67.blogspot.com.by/2014/07/quicksort-algorithm-in-java-in-place-example.html>

QuickSort Example in Java using Recursion - Sorting Algorithm Implementation

Quicksort is one of the very popular sorting algorithm in programming, often used to sort large list of numbers. Though their are numerous algorithm available to sort list of objects, including integer, string and floating point number, quicksort is best for general purpose. It's a divide and conquer algorithm, where we divide the given array with respect to a particular element, known as 'pivot' such that the lower partition of the array are less than the pivot and upper partition elements of the array are higher than the pivot. Quicksort is also one of the best example of [recursion](http://javarevisited.blogspot.sg/2012/12/recursion-in-java-with-example-programming.html). It's naturally recursive, because it sort the large list by dividing into smaller sub-list and then applying same algorithm on those. Base case of recursion is when list contains either one or zero element, in that case they are sorted. Quicksort is well ahead with primitive sorting algorithms e.g. insertion sort, selection sort and [bubble sort](http://java67.blogspot.sg/2012/12/bubble-sort-in-java-program-to-sort-integer-array-example.html). Average time complexity of quicksort is O(nlogn), while in worst case it's performance is similar to bubble sort i.e. O(n^2). Apparently worst case of quicksort is the best case of insertion sort, where they have to sort an already sorted list. In this article, we will learn *how to implement quicksort algorithm in Java using recursion*. We will also learn how quicksort works, and how it sorts large list of unsorted number. In last section, we will see some important things about quicksort.

## How QuickSort Algorithm Perform Sorting

An old saying is, a picture is worth more than a thousand words. This is completely true in case of understanding *how sorting algorithm works*. In past, I have understood insertion sort, selection sort and quicksort much better by following a diagram rather then reading about it. That's why I am sharing this diagram which explains how quicksort algorithm works, how it [sort a list of integers](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html). It's similar to flowchart but doesn't use the notation flowchart uses, instead it practically shows how sorting happens. Once you go through this diagram, read the explanation, it will make more sense.

As I told before *QuickSort is a recursive algorithm*, it divides the big list into smaller list around pivot until those lists are individually sorted. First step of Quicksort algorithm is to determine pivot, it's general practice to choose [middle element of array](http://java67.blogspot.sg/2012/12/how-to-remove-element-from-array-in-java-example.html) as pivot, but you are free to choose any index. Now you have two list, next step is to ensure that left partition only contains numbers less than pivot and right partition only contains numbers greater than pivot. We start pointer from left and right of pivot, and as soon as left pointer encounter 4, it stops because 4 is greater than 3. Similarly, right pointer stops at 3 because all numbers on right side of the list is greater than 3. Now it's time to swap, so 3 takes place of 4 and vice-versa. Now, we move pointer to one more step, since 2 > 3, left pointer shifts but since 4 > 3, it stopped. Since left point is also past right pointer it stopped. Now if we repeat this process one more time, list will be sorted.

### Concept of  Pivot and Partition

Though we often select middle element of array as pivot, there is no such there is no such rule and pivot can be any element of the given array. You can even consider the first element as the pivot in every partition. It's experienced that choice of pivot effects the distribution of the elements in partitioning and affects the complexity of the quicksort algorithm. As per rule of partition, numbers in lower partition should be less than the pivot and upper partition numbers should be higher than the pivot. Running time of partition logic is linear.

### Complexity of Quicksort Algorithm:

On an average Quicksort Algorithm has the complexity of O(nlogn) and in the worst case it has O(n²) when the elements of the input array are already sorted in ascending or descending order. Good thing about Quicksort is that it's an *in place algorithm*, which means it does not takes any additional space, except those used by method stack. By the way, there are **some tricks to improve performance of quicksort**, even in worst case. As suggested in one of the best algorithm design book, The Algorithm Design Manual, from Steven Skiena, you can apply following recommendation to improve your quicksort algorithm implementation.

1) Randomization

You can avoid worst case performance of O(n²) when sorting nearly-sorted data by random permutation of keys. Though it incur some cost of permutation but still gives better performance than O(n²)

2) Leave small sub-arrays for Insertion sort

finish Quicksort recursion and switch to insertion sort when fewer then 20 elements:

By the way, there is a drawback of using recursion to implement quicksort algorithm, It will not scale, because [JVM](http://java67.blogspot.sg/2013/02/difference-between-jit-and-jvm-in-java.html) has no tail call optimization, it will simply grow the method call stack to something proportional to the array to sort, and it will fail for very large array.

### Java Program to Sort Integer Array using QuickSort Algorithm

Here is our recursive implementation of QuickSort sorting algorithm. We have used it to sort an array of randomly distributed integers. We have two set of input, one which doesn't contain any repeated number and other which contains duplicates. Logic of quicksort is encapsulated in method recursiveQuickSort(int[] array, int startIdx, int endIdx) and partition(int[] array, int left, int right), which implements partitioning logic. In order to hide implementation details, we have only exposed a convenient static utility method called quickSort(int[] array), which takes an integer array and sort that in-place.

package test;

import java.util.Arrays;

/\*\*

\* Java program to Sort integer array using QuickSort algorithm using recursion.

\* Recursive QuickSort algorithm, partitioned list into two parts by a pivot,

\* and then recursively sorts each list.

\* @author Javin Paul

\*/

public class QuickSort{

public static void main(String args[]) {

int[] input = { 23, 31, 1, 21, 36, 72};

System.out.println("Before sorting : " + Arrays.toString(input));

quickSort(input); // sort the integer array using quick sort algorithm

System.out.println("After sorting : " + Arrays.toString(input));

// input with duplicates

int[] withDuplicates = { 11, 14, 16, 12, 11, 15};

System.out.println("Before sorting : " + Arrays.toString(withDuplicates));

quickSort(withDuplicates); // sort the integer array using quick sort algorithm

System.out.println("After sorting : " + Arrays.toString(withDuplicates));

}

/\*\*

\* public method exposed to client, sorts given array using QuickSort

\* Algorithm in Java

\* @param array

\*/

public static void quickSort(int[] array) {

recursiveQuickSort(array, 0, array.length - 1);

}

/\*\*

\* Recursive quicksort logic

\*

\* @param array input array

\* @param startIdx start index of the array

\* @param endIdx end index of the array

\*/

public static void recursiveQuickSort(int[] array, int startIdx, int endIdx) {

int idx = partition(array, startIdx, endIdx);

// Recursively call quicksort with left part of the partitioned array

if (startIdx < idx - 1) {

recursiveQuickSort(array, startIdx, idx - 1);

}

// Recursively call quick sort with right part of the partitioned array

if (endIdx > idx) {

recursiveQuickSort(array, idx, endIdx);

}

}

/\*\*

\* Divides array from pivot, left side contains elements less than

\* Pivot while right side contains elements greater than pivot.

\*

\* @param array array to partitioned

\* @param left lower bound of the array

\* @param right upper bound of the array

\* @return the partition index

\*/

public static int partition(int[] array, int left, int right) {

int pivot = array[left]; // taking first element as pivot

while (left <= right) {

//searching number which is greater than pivot, bottom up

while (array[left] < pivot) {

left++;

}

//searching number which is less than pivot, top down

while (array[right] > pivot) {

right--;

}

// swap the values

if (left <= right) {

int tmp = array[left];

array[left] = array[right];

array[right] = tmp;

//increment left index and decrement right index

left++;

right--;

}

}

return left;

}

}

Output:

Before sorting : [23, 31, 1, 21, 36, 72]

After sorting : [1, 21, 23, 31, 36, 72]

Before sorting : [11, 14, 16, 12, 11, 15]

After sorting : [11, 11, 12, 14, 15, 16]

### Things to know about QuickSort Algorithm in Java

As I said, QuickSort is one of the most popular sorting algorithm between programmers, may be just next to Bubble sort, which is ironically worst algorithm to [sort large list of numbers](http://java67.blogspot.sg/2012/08/how-to-sort-arraylist-in-java-list.html). But one thing is common between QuickSort and Bubble Sort, do you know what? In worst case both have complexity of O(n^2).

1) QuickSort is a divide and conquer algorithm, which means it sort a large array of numbers by dividing them into smaller array and then individually sorting them (conquer).

2) Average case complexity of Quicksort is O(n log(n)) and worst case complexity of Quicksort is O(n²).

3) Quicksort is a comparison sort and, in efficient implementations, it's not a stable sort, which means equal numbers may not retain their original position after sorting.

4) Quicksort algorithm can be implemented in-place, which means no additional space will be required. This makes it suitable to sort large array of numbers.

5) Arrays.sort() method in Java use quicksort to sort array of primitives e.g. array of integers or float and uses Mergesort to sot objects e.g. [array of String](http://java67.blogspot.sg/2012/09/java-program-to-convert-string-arraylist-to-string-array.html).

That's all about **how to implement QuickSort algorithm in Java**. QuickSort is one of the fast and efficient sorting algorithm, perfect for sorting large arrays, but some programmer find it extremely hard to understand. One reason of this could be that because quicksort is in-place algorithm due to which programmers find it bit confusing, but it's very efficient. Otherwise if you choose simplicity you can always implement it in other ways.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.com.by/2014/08/quicksort-sorting-algorithm-in-java-in-place-example.html>

Quicksort Sorting Algorithm in Java

Quicksort algorithm is one of the most used sorting algorithm, especially to sort large list and most of the programming languages, library have implemented it in one or another way. In Java, Arrays.sort() method sorts primitive data types using double pivot Quicksort algorithm, authored by Joshua Bloach and others. This implementation provides better performance for lot of data sets, where traditional quicksort algorithm reduced into quadratic performance. This method also uses MergeSort, another good sorting algorithm, to sort objects. QuickSort implementations are also available in C++ STL library. Have you ever thought *why quicksort is so popular?* because on average it is one of the fastest sorting algorithm we have. On average quicksort is a O(n log n) algorithm, while it's worst case is O(n^2), which is much better comparing with Bubble Sort or Insertion Sort. It's also one of the [popular algorithm interview question](http://javarevisited.blogspot.sg/2013/03/top-15-data-structures-algorithm-interview-questions-answers-java-programming.html), so as a programmer you must know *how QuickSort works as well* as *how to implement Quicksort in Java* or any other programming language. One of the most important thing interviewer look in your quicksort implementation is choice of pivot and whether you are sorting in place or not. In "*in-place"* sorting, actual sorting takes place in same array and no additional space is needed. Due to this reason, quicksort is very efficient in sorting large list of numbers, as no additional memory is required, a very space efficient sorting algorithm. Quicksort is also one of the naturally recursive algorithm and serves a good exercise for Java programmers to master [art of recursion](http://javarevisited.blogspot.sg/2012/12/recursion-in-java-with-example-programming.html).

### How QuickSort Algorithm works

Quicksort is a divide and conquer algorithm, which means original list is divided into multiple list, each of them is sorted individually and then sorted output is merged to produce the sorted list. Here is step by step explanation of how quicksort algorithm works :

Steps to implement Quick sort algorithm in place:

1) Choose an element, called pivot, from the list or array. Generally pivot is the middle element of array.

2) Reorder the list so that all elements with values less than the pivot come before the pivot, and all elements with values greater than the pivot come after it (equal values can go either way). This is also known as *partitioning*. After partitioning the pivot is in its final position.

3) Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values. If the array contains only one element or zero elements then the array is sorted.

Following GIF image will help you to understand [working of Quick sort algorithm](http://java67.blogspot.sg/2014/07/quicksort-algorithm-in-java-in-place-example.html) little better. In this image we have an array of integers which is not sorted and we need to sort them in ascending order. Our array is {6, 5, 3, 1, 8, 7, 2, 4} and we first choose 3 as pivot. Now partitioning starts and we pick 6 on left side of side, because its greater than 3. Now on right side, we leave 4 because its greater than 3, and pick 2 for swapping with 6. After swapping our list look like {2, 5, 3, 1, 8, 7, 6, 4}. Now we pick 5 on left side, and 1 on right side because it's greater than 3 and swap them again. Now, our array looks like {2, 1, 3, 5, 8, 7, 6, 4}. Since we are done with all elements with respect to 3 as pivot, we can now take the sub-array at left side of 3 and apply same procedure. This will sort the left array. Now on right side, we choose 4 as pivot, and repeat same procedure, which result in 4 swapped against 5. Now we take right side again with 6 as pivot and apply same procedure.

**Sorting an array of integer using QuickSort sorting algorithm**

### Java Program to implement QuickSort Algorithm

Here is a Java program to [sort an array of integers using QuickSort algorithm](http://java67.blogspot.sg/2014/08/4-examples-to-sort-array-in-java.html). It is an in-place, recursive implementation of QuickSort. Logic is encapsulated in QuickSort class, and method quickSort(int low, int high). This method is called recursively to sort the array. This algorithm work exactly as explained in above GIF image, so if you understand the logic there, its very easy to write by your own.

import java.util.Arrays;

/\*\*

\* Test class to sort array of integers using Quicksort algorithm in Java.

\* @author Javin Paul

\*/

public class QuickSortDemo{

public static void main(String args[]) {

// unsorted integer array

int[] unsorted = {6, 5, 3, 1, 8, 7, 2, 4};

System.out.println("Unsorted array :" + Arrays.toString(unsorted));

QuickSort algorithm = new QuickSort();

// sorting integer array using quicksort algorithm

algorithm.sort(unsorted);

// printing sorted array

System.out.println("Sorted array :" + Arrays.toString(unsorted));

}

}

/\*\*

\* Java Program sort numbers using QuickSort Algorithm. QuickSort is a divide

\* and conquer algorithm, which divides the original list, sort it and then

\* merge it to create sorted output.

\*

\* @author Javin Paul

\*/

class QuickSort {

private int input[];

private int length;

public void sort(int[] numbers) {

if (numbers == null || numbers.length == 0) {

return;

}

this.input = numbers;

length = numbers.length;

quickSort(0, length - 1);

}

/\*

\* This method implements in-place quicksort algorithm recursively.

\*/

private void quickSort(int low, int high) {

int i = low;

int j = high;

// pivot is middle index

int pivot = input[low + (high - low) / 2];

// Divide into two arrays

while (i <= j) {

/\*\*

\* As shown in above image, In each iteration, we will identify a

\* number from left side which is greater then the pivot value, and

\* a number from right side which is less then the pivot value. Once

\* search is complete, we can swap both numbers.

\*/

while (input[i] < pivot) {

i++;

}

while (input[j] > pivot) {

j--;

}

if (i <= j) {

swap(i, j);

// move index to next position on both sides

i++;

j--;

}

}

// calls quickSort() method recursively

if (low < j) {

quickSort(low, j);

}

if (i < high) {

quickSort(i, high);

}

}

private void swap(int i, int j) {

int temp = input[i];

input[i] = input[j];

input[j] = temp;

}

}

Output :

Unsorted array :[6, 5, 3, 1, 8, 7, 2, 4]

Sorted array :[1, 2, 3, 4, 5, 6, 7, 8]

### Import points about Quicksort algorithm

Now we know how quick sort works and how to implement quicksort in Java, its time to revise some of the important points about this popular sorting algorithm.

1) QuickSort is a divide and conquer algorithm. Large list is divided into two and sorted separately (conquered), sorted list is merge later.

2) On "in-place" implementation of quick sort, list is sorted using same array, no additional array is required. Numbers are re-arranged pivot, also known as partitioning.

3) Partitioning happen around pivot, which is usually middle element of array.

4) Average case time complexity of Quicksort is O(n log n) and worst case time complexity is O(n ^2), which makes it one of the fasted sorting algorithm. Interesting thing is it's worst case performance is equal to [Bubble Sort](http://javarevisited.blogspot.sg/2014/08/bubble-sort-algorithm-in-java-with.html) :)

5) Quicksort can be implemented with an in-place partitioning algorithm, so the entire sort can be done with only O(log n) additional space used by the stack during the recursion.

6) Quicksort is also a good example of algorithm which makes best use of CPU caches, because of it's divide and conquer nature.

7) In Java, Arrays.sort() method uses quick sort algorithm to sort array of primitives. It's different than our algorithm, and uses two pivots. Good thing is that it perform much better than most of the quicksort algorithm available on internet for different data sets, where traditional quick sort perform poorly. One more reason, not to reinvent the wheel but to use the library method, when it comes to write production code.

That's all about **Quicksort sorting algorithm in Java**. It is one of the must know algorithm for all level of Java programmers, not that you need it often to implement it but to do well on interviews and use the lesson learned while implementing quick sort in Java. In our example, we have implemented quicksort "in-place", which is what you should do if asked to write quicksort in Java. Remember as Java programmer, you don't need to write your own implementation as library implementation are much better implemented and tested. You should use  Arrays.sort()  method to sort your array instead of writing your own sort method. One more reason of using library method is that they are usually improved over different version, and can take advantage of new machine instructions or native improvement.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.com.by/2014/09/insertion-sort-in-java-with-example.html>

Insertion Sort in Java with Example

Insertion sort is next simple sorting algorithm after Bubble Sort. You may not have realized but you must have used insertion sort in lot of places in your life. One of the best example of insertion sort is, *how you sort your hand in playing cards*. We pick one card from deck, we assume it's sorted, and then we insert subsequent card in their proper position. For example, if our first card is Jack, and our next card Queen then we put that after Jack. Now if next card is King, we put it after queen, and if we get 9, we put it before jack. So if you look closely, insertion sort is perfect sorting algorithm to insert new value in already sorted list. That's why best case complexity of insertion sort is O(n), in which case you insert a new number in already sorted list of integers. Another thing to keep in mind is size of list, insertion sort is very good for small list or array, but not so for large list, where [QuickSort](http://java67.blogspot.sg/2014/07/quicksort-algorithm-in-java-in-place-example.html), MergeSort and HeapSort rules. Let's see one more example of insertion sort from real life. Have you noticed, how does tailors arrange shirts in wardrobe, according to size. So they insert new shirt at proper position, for that they shift existing shirts, until they find the proper place. If you consider wardrobe as array and shirts as element, you will find out that we need to shift existing elements to find right place for new element. This is the *core of insertion sort algorithm*, if you understand these example, even you can come up with a step by step coding algorithm to sort an array of integer using insertion sort in Java. In this article, we will learn that by first understanding insertion sort with flowchart and by walking through an example. After than writing a Java method to do insertion sort will be very easy.

### How Insertion Sort Algorithm works

If you know how to sort hand of cards, you know how insertion sort works; but for many programmers it's not easy to translate real world knowledge into a working code example. This is where natural programming ability comes into play. A good programmer has the ability to code any algorithm, and convert a real life example to algorithm. Now, [how do you sort an array of integer](http://java67.blogspot.sg/2012/12/bubble-sort-in-java-program-to-sort-integer-array-example.html) using this algorithm? You can say that we can treat this array as deck of card, and we will use another array to pick and place element from one place to another. Well that will work, but it's waste of space (memory), because what you are doing is comparing and shifting, which can also be done *in-place* in same array. Here are the **step by step guide to code insertion sort algorithm in Java**:

1) Consider first element is sorted and its on proper place, that is index 0 for your array.

2) Now go to second element (index 1 in array), and compare it with what is in your hand (the part of the array, which is already sorted). Which means you compare this element going backward towards index zero.

3) If current number is smaller than previous number (which is in proper place), we need to put our current number before that. How will we do that? Well for that we need to shift existing number. But what if there is another element which is greater than our current element. It means we need to continue comparing until we found proper place for our current number, which again means current number> existing number or we are at start of the list ([index 0 in array](http://java67.blogspot.com/2014/08/10-points-about-array-in-java.html)).

4) We need to repeat this process for all the numbers in the list. Once we finish that, we have a sorted list or array.

In short, insertion sort is all about finding proper place for current number. Once you find the proper place, you need to shift existing element to make place for this new number. By the way, This algorithm can be better understood by looking at flowchart or a real example with numbers, as shown in following diagram

## Pictorial explanation of Insertion Sort Algorithm

It's said that *"A picture is worth thousand words",* this is quite true in case of understanding sorting algorithm. Earlier we had seen how easy it was to [understand QuickSort algorithm using a GIF](http://javarevisited.blogspot.com/2014/08/quicksort-sorting-algorithm-in-java-in-place-example.html) image, and now we will again learn how Insertion sort works by following this diagram, It become extremely easy to explain how insertion sort works with this example. Here we have an integer array of both positive and negative numbers in random order. Our task is to sort this unsorted array using Insertion Sort in ascending order, which means smallest element should be at start of array and largest element must be at end of array. To start working we assume that our first element is in proper position (remember first card in your hand) and start with second integer, which is  -5. Now we compare it with 7, since - 5 is less than 7, we first move 7 in place of -5. After this we don't need to compare -5 with any other number because we have reached the left boundary, so we will put -5 at current place. Now, we pick the third element which is 2. We compare 2 with 7 and found that 2 is also less than 7, which means 7 shifted in place of 2. Next we compare 2 with -5, now 2 is greater than -5 so we insert it at this place. After this, we pick fourth element which is 16. Since 16 is greater than 7, no need to shift anyone, 16 will remain at it place. Now last element 4, it is less than 16, so 16 will move in place of 4, next we compare 4 with 7, again 4 is less than so 7 will be shifted, after this we compare 4 with 2, wow it's greater that 2, so we have found a proper place for 4. We insert 4 there. Now there is no more element to process in array, so [our array is now sorted](http://java67.blogspot.sg/2014/08/4-examples-to-sort-array-in-java.html).

You can see that at last step our array is sorted in increasing order, starting from - 5 and ending at 16.

### Insertion Sort in Java with Example

It's very easy to implement Insertion sort in Java.  All you need to do is to iterate over array and find proper position of each element, for that you need to shift element and you can do it by swapping. The logic of sorting integer array using insertion sort algorithm is inside method insertionSort(int[]). In Java you can also sort any object e.g. String using this algorithm, all you need to do is to use [Comparable interface](http://java67.blogspot.com/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html), because that will provide you mechanism to compare two object. Now instead of using > (greater than) or < (less than) operator, we need to use compareTo() method. For this we have decided to [overload our insertionSort() method](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html), where overloaded version takes an Object array instead of int array. Both method sorts element using insertion sort logic. By the way, in real world you don't need to reinvent wheel, java.util.Arrays class provides several utility method to operate upon arrays and one of them is sort. There are couple of overloaded version of sort() method available to sort primitive and object arrays. This method uses double pivot QuickSort to sort primitive array and MergeSort to [sort object array](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html). Anyway, here is our complete code example to run Insertion sort in Java. If you are using Eclipse IDE then just copy paste the code in src folder of your Java project and Eclipse will create packages and source file with same name by itself. All you need to is that to run it as Java program.

import java.util.Arrays;

/\*\*

\* Java program to sort an array using Insertion sort algorithm.

\* Insertion sort works great with already sorted, small arrays but not suitable for

\* large array with random order.

\*

\* @author Javin Paul

\*/

public class InsertionSort {

public static void main(String args[]) {

// getting unsorted integer array for sorting

int[] randomOrder = getRandomArray(9);

System.out.println("Random Integer array before Sorting : " + Arrays.toString(randomOrder));

// sorting array using insertion sort in Java

insertionSort(randomOrder);

System.out.println("Sorted array uisng insretion sort : " + Arrays.toString(randomOrder));

// one more example of sorting array using insertion sort

randomOrder = getRandomArray(7);

System.out.println("Before Sorting : " + Arrays.toString(randomOrder));

insertionSort(randomOrder);

System.out.println("After Sorting : " + Arrays.toString(randomOrder));

// Sorting String array using Insertion Sort in Java

String[] cities = {"London", "Paris", "Tokyo", "NewYork", "Chicago"};

System.out.println("String array before sorting : " + Arrays.toString(cities));

insertionSort(cities);

System.out.println("String array after sorting : " + Arrays.toString(cities));

}

public static int[] getRandomArray(int length) {

int[] numbers = new int[length];

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

numbers[i] = (int) (Math.random() \* 100);

}

return numbers;

}

/\*

\* Java implementation of insertion sort algorithm to sort

\* an integer array.

\*/

public static void insertionSort(int[] array) {

// insertion sort starts from second element

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

int numberToInsert = array[i];

int compareIndex = i;

while (compareIndex > 0 && array[compareIndex - 1] > numberToInsert) {

array[compareIndex] = array[compareIndex - 1]; // shifting element

compareIndex--; // moving backwards, towards index 0

}

// compareIndex now denotes proper place for number to be sorted

array[compareIndex] = numberToInsert;

}

}

/\*

\* Method to Sort String array using insertion sort in Java.

\* This can also sort any object array which implements

\* Comparable interface.

\*/

public static void insertionSort(Comparable[] objArray) {

// insertion sort starts from second element

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

Comparable objectToSort = objArray[i];

int j = i;

while (j > 0 && objArray[j - 1].compareTo(objectToSort) > 1) {

objArray[j] = objArray[j - 1];

j--;

}

objArray[j] = objectToSort;

}

}

}

Output:

Random Integer array before Sorting : [74, 87, 27, 6, 25, 94, 53, 91, 15]

Sorted array uisng insretion sort : [6, 15, 25, 27, 53, 74, 87, 91, 94]

Before Sorting : [71, 5, 60, 19, 4, 78, 42]

After Sorting : [4, 5, 19, 42, 60, 71, 78]

String array before sorting : [London, Paris, Tokyo, NewYork, Chicago]

String array after sorting : [Chicago, London, NewYork, Paris, Tokyo]

Another useful thing to learn from this example is [how to generate Random numbers in Java](http://javarevisited.blogspot.com/2013/05/how-to-generate-random-numbers-in-java-between-range.html). You can see that our getRandomArray(int length) method creates a random array of given length. This uses static utility method Math.random() which returns a double value between 0.0 to 0.1, if you need to convert it to integer, in range of 0 to 99, you need to multiply it with 100. After that you can cast it to int to get rid of decimals.

That's all about **Insertion sort in Java**. It's one of the really beautiful algorithm and works best for already sorted list. It has lots of practical uses, but has limitations also. *You should not use Insertion sort for sorting a big list of numbers*, as its best case performance is in order of O(n), which can be very high for a list of say 1 million integers. To short those list, you need sorting algorithms which has logarithmic complexity e.g. quicksort, mergesort or heapsort, which provides best case complexity of O(nLogn), because log reduce power of 10^n into n i.e. 1 million will become 10^6 means 6. In order to remember Insertion sort algorithm, just remember how you sort your hand in poker or any card game. If that is tough, just remember how you arrange your shirts in wardrobe.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.sg/2014/08/4-examples-to-sort-array-in-java.html>

4 Examples to Sort Array in Java

You can use Arrays.sort() method to sort both primitive and object array in Java. This method sorts given array into ascending order, which is numeric order for primitives and defined by compareTo() or compare() method for objects. For primitive arrays e.g. int,  short, character, float, double or long this method uses  dual-pivot Quicksort sorting algorithm implemented by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloach (author of Effective Java) . This algorithm offers **O(n log(n))** performance on many data sets that cause other quicksort algorithms to degrade into their worst quadratic performance e.g. **O(n^2)**, and is typically faster than [traditional (one-pivot) Quicksort implementations](http://java67.blogspot.com/2014/07/quicksort-algorithm-in-java-in-place-example.html). That's why I always said that prefer library method your own, you can get it right but amount of exposure library method gets, you will never get for your implementations. On the other hand object array is sorted using stable MergeSort algorithm, which ensures that equal elements keep their original position in sorted array. Implementation of mergesort used in sort(Object[]) is stable, adaptive, and iterative that requires much lesser than O(n log(n)) comparisons when the input array is partially sorted, while offering the performance of a traditional mergesort when the input array is randomly ordered. In best case, when input array is almost sorted, this implementation requires approximately **O(n)** comparisons. By the way temporary storage requirements vary from a small constant for nearly sorted input arrays to **n/2** object references for randomly ordered input arrays. In order to sort different types of array in Java, you can use any of the overloaded version of sort() method from Arrays class. It also has two special method for sorting object array, one sorts the array in natural order, while other sort them in custom order of provided comparator. Since [two dimensional array is also array of array in Java](http://java67.blogspot.sg/2014/08/10-points-about-array-in-java.html), you can use any this method to sort multi dimensional array in Java also. We will see step by step examples of sorting all kinds of array in Java in subsequent section.

## Sorting Array in Ascending Order

Sorting any primitive or object array in ascending order is very easy, all you need to know is sort() method from java.util.Arrays class. It provides overloaded method to sort byte, short, char, int, long, float, double and object arrays. This method sorts given array in increasing order using two pivot quicksort algorithm. You can use this method to sort any array of objects which implements either [Comparable or Comparator method](http://java67.blogspot.sg/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html). It has and overloaded version, which accept Comparator for custom sorting. Here is an example to sort an int primitive array in ascending order in Java.

int[] random = { 33, 22, 11, 21, 55, 32, 3, 4 };

System.out.println("Array before sorting : " + Arrays.toString(random));

Arrays.sort(random);

System.out.println("Array after sorting in ascending order : " + Arrays.toString(random));

Output:

Array before sorting : [33, 22, 11, 21, 55, 32, 3, 4]

Array after sorting in ascending order : [3, 4, 11, 21, 22, 32, 33, 55]

**How to Sort Integer Array in Java**

Let's see one more example of sort() method, this time we will sort an array of Integer object instead of int primitives. First line may look similar, but remember [autoboxing](http://java67.blogspot.sg/2012/09/autoboxing-enum-generics-varargs-java-5-features.html) will convert each int value to Integer, but it can not convert an int[] to Integer[]. That's why sort method used here is sort(Object[]) and not sort(int[]), this is also obvious when we sort Integer array into reverse order, and passed a reverse order comparator from Collections class.

Integer[] elevens = { 44, 22, 55, 11, 33, 66, 77 };

Arrays.sort(elevens);

System.out.println("increasing order : " + Arrays.toString(elevens));

Arrays.sort(elevens, Collections.reverseOrder());

System.out.println("reverse order : " + Arrays.toString(elevens));

Output:

increasing order : [11, 22, 33, 44, 55, 66, 77]

reverse order : [77, 66, 55, 44, 33, 22, 11]

### Sorting String Array in Java - Ascending and Descending Order

String is not a numeric data, it defines it's own order which is called lexicographic order, also known as alphabetic order. When you [sort an array of String](http://javarevisited.blogspot.sg/2012/01/sort-array-in-java-ascending-and.html) using sort() method, it sorts array into natural order defined by Comparable interface, as shown below :

**Increasing Order**

String[] names = {"John", "Steve", "Shane", "Adam", "Ben"};

System.out.println("String array before sorting : " + Arrays.toString(names));

Arrays.sort(names);

System.out.println("String array after sorting in ascending order : " + Arrays.toString(names));

Output:

String array before sorting : [John, Steve, Shane, Adam, Ben]

String array after sorting in ascending order : [Adam, Ben, John, Shane, Steve]

**Decreasing Order**

Arrays.sort(names, 0, names.length, Collections.reverseOrder());

System.out.println("String array after sorting in descending order : " + Arrays.toString(names));

Output:

String array after sorting in descending order : [Steve, Shane, John, Ben, Adam]

### Sorting Object Array in Java

In order to sort an object array, all elements must implement either Comparable or Comparator interface to define an order. You can use either use sort(Object[]) method to sort an object array on its natural order, you must ensure that all elements in the array must implement Comparable. Furthermore, they must be mutually comparable as well, for example e1.compareTo(e2) must not throw a [ClassCastException](http://javarevisited.blogspot.sg/2012/12/how-to-solve-javalangclasscastexception-java.html) for any elements e1 and e2 in the array. Alternatively you can sort an Object array on custom order using sort(T[], Comparator) method. as shown in following example.

// How to Sort Object Array in Java using Comparator and Comparable

Course[] courses = new Course[4];

courses[0] = new Course(101, "Java", 200);

courses[1] = new Course(201, "Ruby", 300);

courses[2] = new Course(301, "Python", 400);

courses[3] = new Course(401, "Scala", 500);

System.out.println("Object array before sorting : " + Arrays.toString(courses));

Arrays.sort(courses);

System.out.println("Object array after sorting in natural order : " + Arrays.toString(courses));

Arrays.sort(courses, new Course.PriceComparator());

System.out.println("Object array after sorting by price : " + Arrays.toString(courses));

Arrays.sort(courses, new Course.NameComparator());

System.out.println("Object array after sorting by name : " + Arrays.toString(courses));

Output :

Object array before sorting : [#101 Java@200 , #201 Ruby@300 , #301 Python@400 , #401 Scala@500 ]

Object array after sorting in natural order : [#101 Java@200 , #201 Ruby@300 , #301 Python@400 , #401 Scala@500 ]

Object array after sorting by price : [#101 Java@200 , #201 Ruby@300 , #301 Python@400 , #401 Scala@500 ]

Object array after sorting by name : [#101 Java@200 , #301 Python@400 , #201 Ruby@300 , #401 Scala@500 ]

### How to sort Array in reverse order in Java

Sorting an object array in descending order is easy because Java API provides a sort() method which takes both [array](http://java67.blogspot.sg/2014/04/array-length-vs-arraylist-size-java.html) and [Comparator](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html), which can be used to sort array in reverse order. For example you can sort a String array in reverse order by using Collections.reverseComparator() method, which is a built-in reverse comparator from Collections utility class. You can sort all numeric arrays e.g. Integer, Double or Float using same technique. But, when it comes to sort primitive array in reverse order, you left with nothing. You can not sort primitive array with reverse comparator and Java doesn't provide a direct method for sorting in descending order. You can do two things, write your own method using any efficient sorting algorithm e.g. [quicksort](http://java67.blogspot.sg/2014/07/quicksort-algorithm-in-java-in-place-example.html), to sort the array in descending order, or just sort the array using Arrays.sort() method and reverse it. Former can be a clean approach but you would likely going to get efficiency which comes with a library method like Arrays.sort(), which is a double pivot quicksort implementation and offers O(n log(n)) performance on many data sets that cause other quicksorts to degrade to quadratic performance. Second approach is better but will cost you additional O(n) performance. There is one more way, going via Collection route, you can convert array to list and sort the list on reverse order, but you will not get any performance benefit.

### How to Sort Two dimensional Array in Java

There is no easy way to *sort a multi-dimensional array in Java*, Java API provides no direct method to sort a two or three dimensional array, may be because it's not clear how do you want to sort a multi-dimensional array. If you want to sort your two dimensional array on columns then you can use our ColumnComparator class which implements Comparator interface to sort column data. You can see the full code of this class in our program section. It also uses [Java enum](http://java67.blogspot.sg/2013/07/java-enum-code-example-softdrinks-how.html) to define sorting order e.g. ASCENDING and DESCENDING, which is much better than a blank boolean variable. In following examples, we first sort a two dimensional array in ascending order on first column, while in second example we sort it on increasing order but  this time only second column. If you want to sort all columns of a multidimensional array, you can just extend this program to [iterate over array](http://java67.blogspot.sg/2013/08/how-to-iterate-over-array-in-java-15.html), and passing column index to ColumnCompartor.

**Ascending Order on First Column**

Integer[][] numbers = { {9, 6, 5}, {3, 2, 4}, {1, 5, 7} };

System.out.println("Two dimensional array before sorting : " + Arrays.deepToString(numbers));

Arrays.sort(numbers, new ColumnComparator(0, SortingOrder.ASCENDING));

System.out.println("2D array after sorting in ascending order on first column : " + Arrays.deepToString(numbers));

Output

Two dimensional array before sorting : [[9, 6, 5], [3, 2, 4], [1, 5, 7]]

2D array after sorting in ascending order on first column : [[1, 5, 7], [3, 2, 4], [9, 6, 5]]

**Ascending Order on Second Column**

Arrays.sort(numbers, new ColumnComparator(1,SortingOrder.DESCENDING));

System.out.println("Sorting two dimensional String array in Java, Second column, Ascending order : " + Arrays.deepToString(numbers));

Output

Sorting two dimensional String array in Java, Second column, Ascending order : [[9, 6, 5], [1, 5, 7], [3, 2, 4]]

### Java Program to Sort Array in Java

Here is our complete Java program, which you can just copy paste and run in Eclipse by right click. Alternatively you can copy the source in a text file with name same as main class, compile it using [javac](http://javarevisited.blogspot.sg/2012/12/javac-is-not-recognized-as-internal-or-external-command.html) command and run it by using [java](http://java67.blogspot.sg/2013/03/helloworld-in-java-how-to-write-compile-example-tutorial.html) command directly from command prompt. This contains code to sort a primitive array on increasing order, sorting integer and string array in ascending and descending order, sorting object arrays and sorting 2D array columns. If you have any questions or face any problem while running this sample program, please let us know.

import java.util.Arrays;

import java.util.Collections;

import java.util.Comparator;

/\*\*

\* Couple of examples of Multi-dimensional array in Java. It shows how to

\* declare multidimensional array in Java, how to initialise them both inline

\* and using for loop and how to access particular elements from two dimensional

\* array.

\*

\* @author Javin Paul

\*/

public class ArraySorter {

public static void main(String args[]) {

// How to sort Integer array in Java - ascending order

int[] random = { 33, 22, 11, 21, 55, 32, 3, 4 };

System.out.println("Array before sorting : " + Arrays.toString(random));

Arrays.sort(random); // sorts primitive array using quicksort algorithm

System.out.println("Array after sorting in ascending order : "

+ Arrays.toString(random));

// How to sort String array in Java

String[] names = {"John", "Steve", "Shane", "Adam", "Ben"};

System.out.println("String array before sorting : " + Arrays.toString(names));

Arrays.sort(names); // sorts object array using mergesort algorithm

System.out.println("String array after sorting in ascending order : "

+ Arrays.toString(names));

// How to sort String array in descending order in Java

Arrays.sort(names, 0, names.length, Collections.reverseOrder());

System.out.println("String array after sorting in descending order : "

+ Arrays.toString(names));

// How to Sort Object Array in Java using Comparator and Comparable

Course[] courses = new Course[4];

courses[0] = new Course(101, "Java", 200);

courses[1] = new Course(201, "Ruby", 300);

courses[2] = new Course(301, "Python", 400);

courses[3] = new Course(401, "Scala", 500);

System.out.println("Object array before sorting : " + Arrays.toString(courses));

Arrays.sort(courses);

System.out.println("Object array after sorting in natural order : " + Arrays.toString(courses));

Arrays.sort(courses, new Course.PriceComparator());

System.out.println("Object array after sorting by price : " + Arrays.toString(courses));

Arrays.sort(courses, new Course.NameComparator());

System.out.println("Object array after sorting by name : " + Arrays.toString(courses));

// How to sort two dimensional array in Java on first column, increasing order

Integer[][] numbers = { {9, 6, 5}, {3, 2, 4}, {1, 5, 7} };

System.out.println("Two dimensional array before sorting : " + Arrays.deepToString(numbers));

Arrays.sort(numbers, new ColumnComparator(0, SortingOrder.ASCENDING));

System.out.println("2D array after sorting in ascending order on first column : " + Arrays.deepToString(numbers));

// sorting 2D array on second column in descending order

Arrays.sort(numbers, new ColumnComparator(1,SortingOrder.DESCENDING));

System.out.println("Sorting two dimensional String array in Java, Second column, Ascending order : " + Arrays.deepToString(numbers));

}

}

/\*

\* Simple Enum to represent sorting order e.g. ascending and descending order

\*/

enum SortingOrder{

ASCENDING, DESCENDING;

};

/\*

\* Utility Comparator class to sort two dimensional array in Java

\*/

class ColumnComparator implements Comparator<Comparable[]> {

private final int iColumn;

private final SortingOrder order;

public ColumnComparator(int column, SortingOrder order) {

this.iColumn = column;

this.order = order;

}

@Override

public int compare(Comparable[] c1, Comparable[] c2) {

int result = c1[iColumn].compareTo(c2[iColumn]);

return order==SortingOrder.ASCENDING ? result : -result;

}

}

class Course implements Comparable<Course>{

int id;

String name;

int price;

public Course(int id, String name, int price){

this.id = id;

this.name = name;

this.price = price;

}

@Override

public int compareTo(Course c) {

return this.id - c.id;

}

@Override

public String toString() {

return String.format("#%d %s@%d ", id, name, price);

}

public static class PriceComparator implements Comparator<Course>{

@Override

public int compare(Course c1, Course c2) {

return c1.price - c2.price;

}

}

public static class NameComparator implements Comparator<Course>{

@Override

public int compare(Course c1, Course c2) {

return c1.name.compareTo(c2.name);

}

}

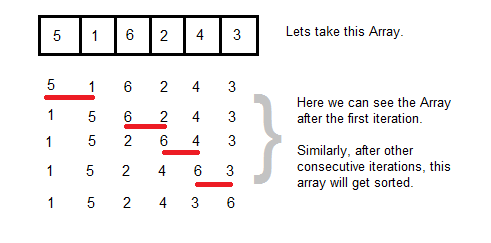
}

That's all about **how to sort an array in Java**. We have learned *sorting both primitive and object array in both ascending and descending order*. Only piece which is bit tricky is sorting primitive arrays in reverse order because their is no direct method to do that in Java. You need to go through steps like first sorting them in increasing order and then reversing or writing your method to do the job or converting array to list and vice-versa. In any case, use library method to do sorting for performance and robustness, as you know how using a two pivot quicksort method on Arrays sort() method gives better performance for arrays for which other similar sorting algorithm result in O(n^2). Autoboxing can not help to convert an int[] to Integer[] so there is no short-cut also. [Prefer List over array](http://java67.blogspot.sg/2012/11/java-arraylist-example-contains-add-set.html) due to this reason but for performance critical code use primitive array to save memory and reduce GC cost.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.sg/2014/08/bubble-sort-algorithm-in-java-with.html>

Bubble Sort Algorithm in Java with Example

[](http://3.bp.blogspot.com/-rzUTGx1fUfY/U-zCFncNoRI/AAAAAAAABx4/YBwz3fXou9A/s1600/Java%2BBubble%2BSort%2BExample.png)Bubble Sort is the first sorting algorithm I learned during my college day, and after so many years it's the one I remember by heart. It's kind of weird that one of the most popular sorting algorithm is also one of the worst performing sorting algorithm. Bubble sort's average case performance is in O(n^2), which means as the size array grows, the time it take to sort that array increases quadratic. Due to this reason, bubble sort is not used in production code, instead quick sort and merge sort are preferred over it. In fact, Java's own Arrays.sort() method, which is the [easiest way to sort an array in Java](http://javarevisited.blogspot.com/2012/01/sort-array-in-java-ascending-and.html) also uses two pivot quicksort to sort primitive array and stable mergesort algorithm to sort object arrays. The reason or slow performance of this algorithm is excessive comparison and swapping, since it compare each element of array to another and swaps if it is on right side. Due to quadratic performance, bubble sort is best suited for small, almost sorted list e.g. {1, 2, 4, 3, 5} , where it just need to do one swapping. Ironically, best case performance of bubble sort, which is O(n) beats quicksort's best case performance of O(NlogN). Someone may argue that why teaching an algorithm which that poor performance, why not teach insertion or selection sort which is as easy as bubble sort, and performs better. IMHO, easiness of algorithm depends upon programmer as much as on algorithm itself. Many programmer will find *insertion sort* easier than *bubble sor*t but again there will be a lot many who will find bubble sort easier to remember, including myself. This is true, despite many of them have used insertion sort unknowingly in real life, e.g. sorting playing cards in hand. Another reason for learning this sorting algorithm is for comparative analysis, how you improve algorithms, how you come up with different algorithms for same problems. In short, despite of all its shortcomings, [bubble sort](http://java67.blogspot.sg/2012/12/bubble-sort-in-java-program-to-sort-integer-array-example.html) is still the most popular algorithm. In this tutorial, we will learn *how bubble sort works*, complexity and performance of bubble sort algorithm,  implementation and source code in Java and a step by step example of bubble sort.

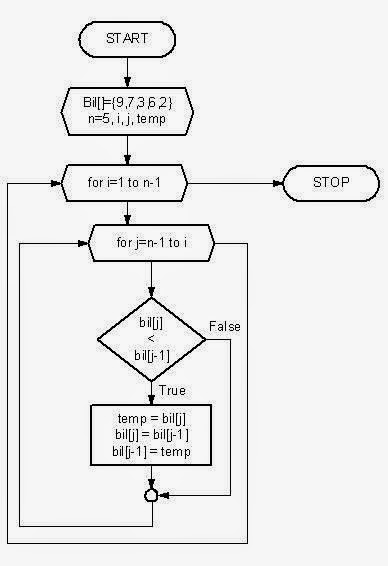
### How Bubble Sort Algorithm works

If you are the one who focus on names, then you might have got an idea how bubble sort works. Just like a bubble comes up from water, in bubble sort smallest or largest number, depending upon whether you are sorting array on ascending or descending order, bubbles up towards start or end of the array. We need at least N pass to sort the array completely and at the end of each pass one elements are sorted in its proper position. You can take first element from array, and start comparing it with other element, [swapping](http://javarevisited.blogspot.sg/2013/02/swap-two-numbers-without-third-temp-variable-java-program-example-tutorial.html) where it's lesser than the number you are comparing. You can start this comparison from start or from end, as we have compared elements from end in our bubble sort example. It is said that a picture is worth more than a thousand word and it's particularly true in case of understanding sorting algorithm. Let' see an *step by step example to sort array using bubble sort*, as I said after each pass largest number is sorted.

In this [array](http://javarevisited.blogspot.sg/2013/11/java-array-101-for-programmers-and.html), we start from index 0, which is 5 and starts comparing elements from start to end. So first element we compare 5 is 1, and since 5 is greater than 1 we swap them ( because ascending order sorted array will have larger number towards end). Next we compare 5 to 6, here  no swapping because 6 is greater than 5 and it's on higher index than 5. Now we compare 6 to 2, again we need swapping to move 6 towards end. At the end of this pass 6 reaches (bubbles up) at the top of the array. In next iteration 5 will be sorted on its position and after n iteration all elements will be sorted. Since we compare each element with another, we need two for loops and that result in complexity of O(n^2).

### FlowChart of Bubble Sort Algorithm

Another cool way to understand an algorithm is to draw it's flowchart. It will walk through each iteration in loop and how decisions are made during algorithm execution. Here is flowchart of our bubble sort algorithm, which complements our implementation of this sorting algorithm.



Here we have integer array {9, 7, 3, 6, 2} and start with four variable i, j, temp and array length which is stored in variable n. We have two for loop, outer loop runs from 1 to n-1. Our inner loop runs from n-1 to i. Many programmer make mistake here, if you start outer loop with second element than make sure to use j>=i condition on inner loop, or if you start with first element e.g. i=0, make sure you use j>i to avoid [ArrayIndexOutOfBound exception](http://javarevisited.blogspot.sg/2014/05/exception-in-thread-main-arrayindexoutofboundsexception-java.html). Now we compare each element and swap them to move smaller element towards front of array. As I said depending upon your navigation direction either largest element will be sorted at highest index in first pass or smallest element will be placed in lowest index. In this case, after first pass, smallest number will be sorted. This loop runs until j>=i after than it finishes and i becomes i + 1. This whole process repeats until outer loop is finished and that time your array is sorted. In flowchart, a diamond box is used for decision making, which is equivalent of if-else statement in code. You can see here decision box is inside inner loop, which means we do N comparison in each iteration, totals to NxN comparisons.

### Complexity and Performance of Bubble Sort Algorithm

As I said before compared to other sorting algorithm like quicksort, merge sort or shell sort, bubble sort performs poorly. These algorithm has average case complexity of O(NLogN), while average case complexity of bubble sort O(n^2). Ironically in best case bubble sort do better than [quicksort](http://java67.blogspot.com/2014/07/quicksort-algorithm-in-java-in-place-example.html) with O(n) performance.  Bubble sort is three times slower than quicksort or mergesort even for n = 100 but it's easier to implement and remember. here is the summary of bubble sort performance and complexity :

Bubble sort Worst case performance       O(n^2)

Bubble sort Best case performance         O(n)

Bubble sort Average case performance    O(n^2)

You can further explore insertion sort and selection sort, which also does sorting in similar time complexity. By the you can not only sort the array using bubble sort but ArrayList or any other collection class as well. Though you should really use Arrays.sort() or Collections.sort() for those purpose.

### Bubble Sort Implementation in Java

here is the Java program to implement bubble sort algorithm using Java programming language. Don't surprise with import of java.util.Array, we have not used it's sort method here, instead it is used to [print arrays in readable format](http://javarevisited.blogspot.sg/2012/12/3-example-to-print-array-values-in-java.html). I have created a swap function to swap numbers and improve readability of code, if you don't like you can in-line the code in the swap method inside if statement of inner loop. Though I have used main method for testing, as it demonstrate better, I would suggest you to write some unit test case for your bubble sort implementation. If you don't know how to do that, you can see this [JUnit tutorial](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html).

import java.util.Arrays;

/\*\*

\* Java program to implement bubble sort algorithm and sort integer array using

\* that method.

\*

\* @author Javin Paul

\*/

public class BubbleSort{

public static void main(String args[]) {

bubbleSort(new int[] { 20, 12, 45, 19, 91, 55 });

bubbleSort(new int[] { -1, 0, 1 });

bubbleSort(new int[] { -3, -9, -2, -1 });

}

/\*

\* This method sort the integer array using bubble sort algorithm

\*/

public static void bubbleSort(int[] numbers) {

System.out.printf("Unsorted array in Java :%s %n", Arrays.toString(numbers));

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

for (int j = numbers.length -1; j > i; j--) {

if (numbers[j] < numbers[j - 1]) {

swap(numbers, j, j-1);

}

}

}

System.out.printf("Sorted Array using Bubble sort algorithm :%s %n",

Arrays.toString(numbers));

}

/\*

\* Utility method to swap two numbers in array

\*/

public static void swap(int[] array, int from, int to){

int temp = array[from];

array[from] = array[to];

array[to] = temp;

}

}

Output

Unsorted array in Java : [20, 12, 45, 19, 91, 55]

Sorted Array using Bubble sort algorithm : [12, 19, 20, 45, 55, 91]

Unsorted array in Java : [-1, 0, 1]

Sorted Array using Bubble sort algorithm : [-1, 0, 1]

Unsorted array in Java : [-3, -9, -2, -1]

Sorted Array using Bubble sort algorithm : [-9, -3, -2, -1]

### How to improve Bubble Sort Algorithm

In interview one of the popular follow-up question is how do you improve a particular algorithm, and Bubble Sort is no different than that. If you wrote bubble sort like the one we have shown here, interviewer will definitely going to ask about how do you improve your bubble sort method. In order to improve any algorithm, you must understand how each step of that algorithm works, then only you will be able to spot any deficiency in code. If you follow the tutorial, you will find that [array is sorted by moving elements to their correct position](http://java67.blogspot.sg/2014/08/4-examples-to-sort-array-in-java.html). In worst case situation if array is reverse sorted then we need to move every element, this will require n-1 passes, n-1 comparison in each pass and n-1 exchanges, but how about best case if array is already sorted, our existing bubble sort method is still going to take n-1 pass, same number of comparison but no exchange. If you observe carefully, you will find that after one pass through the array, the largest element will moved to the end of the array, but many other elements also moved toward their correct positions, as bubbles move toward the water’s surface. By leveraging this property, you can deduce that during a pass, if no pair of consecutive entries is out of order, then the array is sorted. Our current algorithm is not taking advantage of this property. If we track exchanges then we can decide whether additional iteration over array is needed or not. Here is an i*mproved version of Bubble Sort algorithm*, which will only take 1 iteration and n-1 comparison in best case, when array is already sorted. This will also improve Bubble sort's average case performance, as compared to our existing method which will always take N - 1 passes.

/\*

\* An improved version of Bubble Sort algorithm, which will only do

\* 1 pass and n-1 comparison if array is already sorted.

\*/

public static void bubbleSortImproved(int[] number) {

boolean swapped = true;

int last = number.length - 2;

// only continue if swapping of number has occurred

while (swapped) {

swapped = false;

for (int i = 0; i <= last; i++) {

if (number[i] > number[i + 1]) {

// pair is out of order, swap them

swap(number, i, i + 1);

swapped = true; // swapping occurred

}

}

// after each pass largest element moved to end of array

last--;

}

}

Now let's test this method for two input, one in which array is sorted (best case) and other on which only one pair is out of order. If we pass int array {10, 20, 30, 40, 50, 60} to this method,  initially will go inside while loop and make swapped=false. Then it will go inside for loop. when i =0 it will compare number[i] to number[i+1] i.e. 10 to 20 and check if 10 > 20, since it's not it will not go inside if block and no swapping will be occurred. When i=1, it will compare 20 > 30 again no swapping, next when i =2 30> 40 which is false so no exchange again, next i =3 so 40> 50, which is again false, so no swapping. Now last pair comparison i=4, it will compare 50 > 60 again this is false, so control will not go inside if block and no exchange will be made. Because of that swapped will remain false and control will not go inside while loop again. So you know that your [array is sorted just after one pass](http://javarevisited.blogspot.sg/2012/01/sort-array-in-java-ascending-and.html).

Now consider another example, where just one pair is out of order, let's say String array names = {"Ada", "C++", "Lisp", "Java", "Scala"}, here only one pair is out of order e.g. "Lisp" should come after "Java". Let's see how our improved bubble sort algorithm work here. In first pass, comparison will continue without exchange until we compare "Lisp" to "Java", here "Lisp".compareTo("Java") > 0 will become true and swapping will occur, which means Java will go to the Lisp place, and Lisp will take Java's place. this will make boolean variable swapped=true, Now in last comparison on this pass, we compare "Lisp" to "Scala" and again no exchange. Now we will reduce last index by 1 because Scala is sorted at last position and will not participate  further. But now swapped variable is true, so control will again go inside while loop, and for loop but this time no exchanged will be made so it will not take another pass. Our array is now sorted in just two pass compared to N-1 pass of earlier implementation. This bubble sort implementation is much better and even perform better than Selection sort algorithm in average case because, now sorting is not proportional to total number of elements but only with number of pairs which are out-of-order.

By the way to sort String array using Bubble Sort, you need to overload BubbleSortImproved() method to accept String[] and also need to use compareTo() method to [compare two String object lexicographically](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html). Here is Java program to sort String array using Bubble Sort :

import java.util.Arrays;

class BubbleSortImproved {

public static void main(String args[]) {

String[] test = {"Ada", "C++", "Lisp", "Java", "Scala"};

System.out.println("Before Sorting : " + Arrays.toString(test));

bubbleSortImproved(test);

System.out.println("After Sorting : " + Arrays.toString(test));

}

/\*

\* An improved implementation of Bubble Sort algorithm, which will only do

\* 1 pass and n-1 comparison if array is already sorted.

\*/

public static void bubbleSortImproved(String[] names) {

boolean swapped = true;

int last = names.length - 2;

// only continue if swapping of number has occurred

while (swapped) {

swapped = false;

for (int i = 0; i <= last; i++) {

if (names[i].compareTo(names[i + 1]) > 0) {

// pair is out of order, swap them

swap(names, i, i + 1);

swapped = true; // swapping occurred

}

}

// after each pass largest element moved to end of array

last--;

}

}

public static void swap(String[] names, int fromIdx, int toIdx) {

String temp = names[fromIdx]; // exchange

names[fromIdx] = names[toIdx];

names[toIdx] = temp;

}

}

Output:

Before Sorting : [Ada, C++, Lisp, Java, Scala]

After Sorting : [Ada, C++, Java, Lisp, Scala]

### Which one is better Selection Sort vs Bubble Sort?

Though both Selection Sort and Bubble sort has complexity of O(n^2) in worst case. On average, we expect the bubble sort to perform better than Selection sort, because bubble sort will finish sorting sooner than the selection sort due to more data movements for the same number of comparisons, because *we compare elements in pair on Bubble Sort*. If we use our improved implementation Bubble Sort then a boolean test to not enter on while loop when array gets sorted will also help. As I said, The worst case of the bubble sort happens when the original array is in descending order, while in best case, if the original array is already sorted, the bubble sort will perform only one pass whereas the selection sort will perform N - 1 passes. Given this, I think on average Bubble sort is better than Selection sort.

That's all about **Bubble Sort in Java**. We have learned *how bubble sort algorithm works* and how do you implement it in Java. As I said, it is one of the simplest sorting algorithm and very easy to remember, but also it doesn't have any practical use apart from academics and in data structure and algorithm training classes. It's worst case performance is quadratic which means it not suitable for large array or list. If you have to use bubble sort, it's best suited for small, already sorted array in which case it has to very few swapping and it's performance is in O(n). If you love algorithms, you can see this problem of [finding cycle on linked list](http://javarevisited.blogspot.sg/2013/05/find-if-linked-list-contains-loops-cycle-cyclic-circular-check.html).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.sg/2012/12/bubble-sort-in-java-program-to-sort-integer-array-example.html>

Bubble sort in Java - program to sort integer array

Bubble sort is one of the classic sorting algorithm which is used to explain sorting during various computer and engineering courses. Because of its algorithmic nature and simplicity its often used in various Java and C++ programming exercises. You may expect questions like Write Java program to sort integer array using bubble sort during any programming interview. Since algorithmic questions are always tricky question and not easy to code. Even simplest of them can lead to confusion, especially if you are not gifted with a natural programming head. I have seen many developers fumble if asked to code on the spot. That's why its advisable to do algorithmic and logical programming during training and learning programming and OOPS to get this skill of converting logic into code. Let's come back to Bubble sort, In Bubble sort algorithm we sort an unsorted array by starting from first element and comparing with adjacent element. If former is greater than later then we swap and by doing this we get the largest number at the end after first iteration. So in order to sort n elements you require n-1 iteration and almost n-1 comparison. For recap here is the logic for bubble sort sorting algorithm :

1) start comparing a[0] to a[1]

2) if a[0] > a[1] then swap numbers e.g. a[0]=a[1] and a[1]=a[0]

3) compare a[1] to a[2] and repeat till you compare last pair

4) This is referred as one pass and at the end of first pass largest number is at last

5) repeat this comparison again starting from a[0] but this time going till second last pair only

Now let's see Java program which implements this bubble sort logic to sort unsorted integer array.

How to sort integer array using bubble sort in Java

Java program for Bubble sort algorithm with exampleHere is complete code example of bubble sort in Java. It uses same algorithm as explained in first pass, it uses two loops. Inner loop is used to compare adjacent elements and outer loop is used to perform Iteration. because of using two loops it result in order of n^2 which is not great in terms of performance. If you are using Array List instead of array than you can sort them using Collections.sort method for better performance, for details check How to sort Array List in ascending and descending order.

package test;

import java.util.Arrays;

/\*\*

\* Java program to sort integer array using bubble sort sorting algorithm.

\* bubble sort is one of the simplest sorting algorithm but performance

\* of bubble sort is not good, its average and worst case performance

\* ranges in O(n2) and that's why it is not used to sort large set of

\* unsorted data. Bubble sort can be used for educational and testing

\* purpose to sort small number of data to avoid performance penalty.

\* This program is also a good example of how to print contents of Array in Java

\*

\* @author http://java67.blogspot.com

\*/

public class BubbleSort {

public static void main(String args[]) {

//testing our bubble sort method in Java

int[] unsorted = {32, 39,21, 45, 23, 3};

bubbleSort(unsorted);

//one more testing of our bubble sort code logic in Java

int[] test = { 5, 3, 2, 1};

bubbleSort(test);

}

/\*

\* In bubble sort we need n-1 iteration to sort n elements

\* at end of first iteration larget number is sorted and subsequently numbers smaller

\* than that.

\*/

public static void bubbleSort(int[] unsorted){

System.out.println("unsorted array before sorting : " + Arrays.toString(unsorted));

// Outer loop - need n-1 iteration to sort n elements

for(int i=0; i<unsorted.length -1; i++){

//Inner loop to perform comparision and swapping between adjacent numbers

//After each iteration one index from last is sorted

for(int j= 1; j<unsorted.length -i; j++){

//If current number is greater than swap those two

if(unsorted[j-1] > unsorted[j]){

int temp = unsorted[j];

unsorted[j] = unsorted[j-1];

unsorted[j-1] = temp;

}

}

System.out.printf("unsorted array after %d pass %s: %n", i+1, Arrays.toString(unsorted));

}

}

}

Output:

unsorted array before sorting : [32, 39, 21, 45, 23, 3]

unsorted array after 1 pass [32, 21, 39, 23, 3, 45]:

unsorted array after 2 pass [21, 32, 23, 3, 39, 45]:

unsorted array after 3 pass [21, 23, 3, 32, 39, 45]:

unsorted array after 4 pass [21, 3, 23, 32, 39, 45]:

unsorted array after 5 pass [3, 21, 23, 32, 39, 45]:

unsorted array before sorting : [5, 3, 2, 1]

unsorted array after 1 pass [3, 2, 1, 5]:

unsorted array after 2 pass [2, 1, 3, 5]:

unsorted array after 3 pass [1, 2, 3, 5]

That's all on How to sort integer array using Bubble sort in Java. We have seen a complete Java program for bubble sort and also printed output after each pass or iteration, if you look at carefully you will find that after each pass largest number gets sorted and number of comparison decreased. As I said Bubble sort is not a high performance sorting algorithm and you should by using Collection.sort() method from standard Java library to sort Collections or Arrays.sort() to sort Array in Java. Also this program demonstrate How to print contents of Array using Arrays.toString() as array in Java doesn’t override toString and simply printing array using System.out.println(array) will only show defulat toString from java.lang.Object class instead of contents of array.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.sg/2013/03/top-15-data-structures-algorithm-interview-questions-answers-java-programming.html>

Top 15 Data Structures and Algorithm Interview Questions for Java programmer – Answers

Data structures and algorithm questions are important part of any programming job interview, be it a Java interview, C++ interview or any other programming language. Since data structures is core programming concept, its mandatory for all programmers, to know basic data structures like stack, linked list, queue, array, tree and graph. Though tree and graph are on tough side, I still see programmers to get familiar will all these. Any list of programming job interview questions is incomplete without questions from data structures and algorithms. Similarly while going on questions from data structure you may get some programming exercise as well e.g. [swapping numbers without temp variable](http://javarevisited.blogspot.com/2013/02/swap-two-numbers-without-third-temp-variable-java-program-example-tutorial.html) . Linked list and arrays are favorite topics in any data structure interview, questions like reversing linked list, traversing linked list or deleting nodes from linked list, which involves algorithm and data structures are quite common. Similarly, finding duplicates in array, finding missing numbers, sorting arrays are very popular. You can also expect questions from stack, queue, array, linked list, tree, graph and HashMap are most common in any data structure interview. In this tutorial, we will see couple of data structure questions answers from these topic. Let us know, if you have any interesting questions from data structures and algorithm, which you faced during any Java interviews.   
  
I also suggest to look on data structure and algorithm questions on [Cracking the Coding Interview](http://www.amazon.com/dp/098478280X/?tag=javamysqlanta-20) book, as that book also contains some good questions, which helps you to prepare better.   
  
One more suggestion I have to read [Introduction to Algorithm by Thomas Cormen](http://www.amazon.com/dp/0072970545/?tag=javamysqlanta-20), if you have not read already. This books is bible of algorithm and IMHO every programmer should read this book. 

## Data Structures and Algorithm Interview Questions for Java Programmers

[Data Structure and Algorithm Interview Questions answers for Java programmer](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)This is combined list of questions from various data structure e.g. array, linked list, stack or queue. It includes some coding questions as well, which gel with data structures.

### Question 1 : How to find middle element of linked list in one pass?

One of the most popular question from data structures and algorithm, mostly asked on telephonic interview. Since many programmer know that, in order to find length of linked list we need to first traverse through linked list till we find last node, which is pointing to null, and then in second pass we can find middle element by traversing only half of length. They get confused when interviewer ask him to do same job in one pass. In order to find middle element of linked list in one pass you need to maintain two pointer, one increment at each node while other increments after two nodes at a time, by having this arrangement, when first pointer reaches end, second pointer will point to middle element of linked list. See this trick to [find middle element of linked list in single pass](http://javarevisited.blogspot.com/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html) for more details.

### Question 2 : How to find if linked list has loop ?

This question has bit of similarity with earlier algorithm and data structure interview question. I mean we can use two pointer approach to solve this problem. If we maintain two pointers, and we increment one pointer after processing two nodes and other after processing every node, we are likely to find a situation where both the pointers will be pointing to same node. This will only happen if linked list has loop.

### Question 3 : How to find 3rd element from end in a linked list in one pass?

This is another frequently asked linked list interview question. This question is exactly similar to [finding middle element of linked list in single pass](http://javarevisited.blogspot.sg/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html). If we apply same trick of maintaining two pointers and increment other pointer, when first has moved upto 3rd element, than when first pointer reaches to the end of linked list, second pointer will be pointing to the 3rd element from last in a linked list.

### Question 4 : In an integer array, there is 1 to 100 number, out of one is duplicate, how to find ?

This is a rather simple data structures question, especially for this kind of. In this case you can simply add all numbers stored in array, and total sum should be equal to n(n+1)/2. Now just subtract actual sum to expected sum, and that is your duplicate number. Of course there is a brute force way of checking each number against all other numbers, but that will result in performance of O(n^2) which is not good. By the way this trick will not work if array have multiple duplicates or its not numbers forming arithmetic progression. Here is example of one way to [find duplicate number in array](http://javarevisited.blogspot.com/2012/02/how-to-check-or-detect-duplicate.html).

### Question 6 : How to reverse String in Java ?

This is one of my favorite question. Since String is one of the most important type in programming, you expect lot of question related to String any data structure interview. There are many ways to reverse Sting in Java or any other programming language, and interviewer will force you to solve this problem by using without API i.e. without using reverse() method of StringBuffer. In follow-up he may ask to reverse String using recursion as well. See [3 ways to reverse String in Java](http://javarevisited.blogspot.com/2012/01/how-to-reverse-string-in-java-using.html) to learn reversing String using both loops and [recursion in Java](http://javarevisited.blogspot.com/2012/12/recursion-in-java-with-example-programming.html).

## Question 7 : Write a Java program to sort a array using Bubble Sort algorithm?

I have always send couple of questions from searching and sorting in data structure interviews. Bubble sort is one of the simplest sorting algorithm but if you ask anyone to implement on the spot it gives you an opportunity to gauge programming skills of a candidate. See [How to sort array using Bubble Sort in Java](http://java67.blogspot.com/2012/12/bubble-sort-in-java-program-to-sort-integer-array-example.html) for complete solution of this datastrucutre interview question.

### Question 8 : What is difference between Stack and Queue data structure ?

One of the classical datastrucutre interview question. I guess every one know, No? Any way main difference is that Stack is LIFO(Last In First Out) data structure while Queue is a FIFO(First In First Out) data structure.

### Question 9 : How do you find duplicates in array if there is more than one duplicate?

Sometime this is asked as follow-up question of earlier datastrucutre interview question, related to finding duplicates in Array. One way of solving this problem is using a [Hashtable or HashMap](http://javarevisited.blogspot.com/2013/02/how-to-get-key-from-value-in-hashtable.html) data structure. You can traverse through array, and store each number as key and number of occurrence as value. At the end of traversal you can find all duplicate numbers, for which occurrence is more than one. In Java if a number already exists in [HashMap](http://java67.blogspot.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html) then calling get(index) will return number otherwise it return null. this property can be used to insert or update numbers in HashMap.

### Question 10 : What is difference between Singly Linked List and Doubly Linked List data structure?

This is another classical interview question on data structure, mostly asked on telephonic rounds. Main difference between singly linked list and doubly linked list is ability to traverse. In a single linked list, node only points towards next node, and there is no pointer to previous node, which means you can not traverse back on a singly linked list. On the other hand doubly linked list maintains two pointers, towards next and previous node, which allows you to navigate in both direction in any linked list.

### Question 11 : Write Java program to print Fibonacci series ?

This is not a data structures question, but a programming one, which many times appear during data structure interview. Fibonacci series is a mathematical series, where each number is sum of previous two numbers e.g. 1,1, 2, 3, 5, 8, 13, 21. Interviewer is often interested in two things, a function which returns nth number in Fibonacci series and solving this problem using recursion in Java. Though, its easy question, recursion part often confuses beginners. See this link to [find nth Fibonacci number in Java](http://java67.blogspot.sg/2012/07/java-program-fibonacci-series-with.html).

### Question 12 : Write Java program to check if a number is palindrome or not?

This is similar to previous question, not directly related to data structures, but quite popular along with other questions. A number is called palindrome, if reverse of number is equal to number itself. Interviewer ask to solve this problem without taking help from Java API or any open source library. Any way it’s simple question, you can use division operator (/) and remainder operator (%) to solve this question. Just remember, division operator can be used to get rid of last digit e.g. 1234/10 will give you 123, and modulus operator can give you last digit e.g. 1234%10 will return 4. By the way, here is a [Java program check if number is palindrome or not](http://javarevisited.blogspot.com/2012/12/how-to-check-if-number-is-palindrome-or-not-example.html).

### Question 13 : What is binary search tree?

This is a data structure question from Tree data structures. Binary Search Tree has some special properties e.g. left nodes contains items whose value is less than root , right sub tree contains keys with higher node value than root, and there should not be any duplicates in the tree. Apart from definition, interview can ask you to implement binary search tree in Java and questions on tree traversal e.g. IN order, preorder, and post order traversals are quite popular data structure question.

### Question 14 : How to reverse linked list using recursion and iteration?

This is another good question on data structures. There are many algorithm to reverse linked list and you can search of them using google. I am thinking of writing another blog post to explain linked list reversal and will share with you later.

### Question 15 : Write a Java program to implement Stack in Java?

You can implement Stack by using array or linked list. This question expect you to implement standard method provided by stack data structure e.g. push() and pop(). Both push() and pop() should be happen at top of stack, which you need to keep track. It’s also good if you can implement utility methods like contains(), isEmpty() etc. By the way JDK has java.util.Stack class and you can check it’s code to get an idea. You can also check [Effective Java book](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20), where Josh Bloch has explains how an incorrect implementation of stack can cause memory leak in Java.

That's all on this **list of data structure interview questions and answers**. This is one topic, which is always asked in any programming interview, doesn't matter if you are C, C++, or Java developer, basic knowledge of data structure like array, linked list, stack, queue, tree are must to clear any programming interview.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.sg/2012/08/how-to-sort-arraylist-in-java-list.html>

How to sort ArrayList in java - List Sorting - Ascending Descending Order

How to sort ArrayList ascending descending order   
Sorting ArrayList in Java is not difficult, by using Collections.sort() method you can [sort ArrayList in ascending and descending order in Java](http://javarevisited.blogspot.sg/2012/01/how-to-sort-arraylist-in-java-example.html). Collections.sort() method optionally accept a Comparator and if provided it uses Comparator's compare method to compare Objects stored in Collection to compare with each other, in case of no explicit Comparator, Comparable interface's [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) is used to compare objects from each other. If object's stored in [ArrayList](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html) doesn't implements Comparable than they can not be sorted using Collections.sort() method in Java.

## Sorting ArrayList in Java – Code Example

Here is a complete code example of How to sort ArrayList in Java, In this Sorting we have used [Comparable](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html) method of String for sorting String on there natural order, You can also use [Comparator](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html) in place of Comparable to sort String on any other order than natural ordering e.g. in reverse order by using Collections.reverseOrder() or in case insensitive order by using String.CASE\_INSENSITIVE\_COMPARATOR.

**import** java.util.ArrayList;   
**import** java.util.Collections;   
/\*\*   
 \*   
 \* Java program to demonstrate **How to sort ArrayList in Java** in both ascending   
 \* and descending order by using core Java libraries.  
 \*   
 \* @author Javin  
 \*/   
**public** **class** CollectionTest {   
  
      
    **public** **static** **void** main(**String** args[]) {   
      
        *//Creating and populating ArrayList in Java for Sorting*   
        **ArrayList**<**String**> unsortedList = **new** **ArrayList**<**String**>();   
          
        unsortedList.add("Java");   
        unsortedList.add("C++");   
        unsortedList.add("J2EE");   
          
        **System**.err.println("unsorted ArrayList in Java : " + unsortedList);   
          
        *//Sorting ArrayList in ascending Order in Java*   
        **Collections**.sort(unsortedList);   
        **System**.out.println("Sorted ArrayList in Java - Ascending order : " + unsortedList);   
          
        *//Sorting ArrayList in descending order in Java*   
        **Collections**.sort(unsortedList, **Collections**.reverseOrder());   
        **System**.err.println("Sorted ArrayList in Java - Descending order : " + unsortedList);   
    }   
}   
**Output**:   
unsorted **ArrayList** in Java : [Java, C++, J2EE]   
Sorted **ArrayList** in Java - Ascending order : [C++, J2EE, Java]   
Sorted **ArrayList** in Java - Descending order : [Java, J2EE, C++]

That's all on **How to Sort ArrayList in Java** on both ascending and descending order. Just remember that Collections.sort() will sort the ArrayList in ascending order and if you provide reverse comparator it will sort the ArrayList in descending order in Java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.com.by/2012/01/sort-array-in-java-ascending-and.html>

How to Sort Array in Java ascending and descending order of elements with Example

Quite often we need to **sort array in Java** luckily java.util.Arrays class provide several utility method to sort java array of any type e.g. primitive, object or int, String etc. Arrays is in java.util package and exposed all sorting related method as static utility functions. you can access sort() as Arrays.sort() and just pass your array and it will sort that array object. You can sort array in ascending order, descending order or any custom order defined by custom comparator in Java. In last article we have seen [how to Sort ArrayList in Java](http://javarevisited.blogspot.com/2012/01/how-to-sort-arraylist-in-java-example.html) and in this java article we will see **examples of sorting array in ascending order**, **descending order** and **sorting sub arrays in java**.

## Sort Array in Java on ascending and descending order

### Sorting Java Array into Ascending Order:

[how to sort array in java ascending descending order example](http://javarevisited.blogspot.com/2011/09/spring-interview-questions-answers-j2ee.html)In order to sort an array (String, int or any type) in ascending order we don't need to to anything special. Arrays.sort() method by default sort elements in there **natural order** implemented by [CompareTo method](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html). String array in Java will be sorted **lexicographically**. integer will be sorted **numerically** ascending order. look on code section for complete example of *how to sort java array into ascending order*. You can also sort [anonymous array in Java](http://javarevisited.blogspot.com/2012/01/anonymous-array-example-java-create.html) by this method but since you don’t have any name to refer that it doesn’t make much sense.

### Sorting Java Array into Descending Order:

In order to sort a java array into **descending** **order** you need to provide an [external Comparator in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) which can sort elements in there reverse order and luckily we have that built int as java.util.Collections.reverseOrder(), which returns a **reverse order comparator** which gives reverse of natural ordering of element objects. pass this reverse comparator to sort method and it will *sort array into descending order* as shown in code example on last section. There is another way in which you can [convert Array into ArrayList](http://javarevisited.blogspot.com/2011/06/converting-array-to-arraylist-in-java.html) and than [sort the arraylist](http://javarevisited.blogspot.com/2012/01/how-to-sort-arraylist-in-java-example.html) but that’s again a two step process so its better to sort array using Arrays.sort() method

### Sorting Sub Array in Java

Arrays class also provide methods to [sort](http://javarevisited.blogspot.com/2011/08/unix-sort-command-example-tutorial.html) part of java array or sub array. So in case if you have big array and you only need to sort a particular section than just provide start index and end index to java.util.Arrays.sort(array,index,index) and it will sort the array only for that range. this is much faster than sorting whole array and than getting a range of values out of it. here is code example of sorting part of array in Java:

### Code Example of Sorting Array in Java

In this section of code example we will see how to sort Java array into **ascending order**, **descending order** and **sorting sub arrays in Java**:

**import** java.util.Arrays;

**import** java.util.Collections;

**public** **class** HashtableDemo {

**public** **static** **void** main(String args[]) {

String[] companies = { "Google", "Apple", "Sony" };

**// sorting java array in ascending order**

System.*out*.println("Sorting String Array in Ascending order in Java Example");

System.*out*.println("Unsorted String Array in Java: ");

*printNumbers*(companies);

Arrays.*sort*(companies);

System.*out*.println("Sorted String Array in ascending order : ");

*printNumbers*(companies);

**// sorting java array in descending order**

System.*out*.println("Sorting Array in Descending order in Java Example");

System.*out*.println("Unsorted int Array in Java: ");

*printNumbers*(companies);

Arrays.*sort*(companies, Collections.*reverseOrder*());

System.*out*.println("Sorted int Array in descending order : ");

*printNumbers*(companies);

System.*out*.println("Sorting part of array in java:");

**int**[] numbers = { 1, 3, 2, 5, 4 };

Arrays.*sort*(numbers, 0, 3);

System.*out*.println("Sorted sub array in Java: ");

**for** (**int** num : numbers) {

System.*out*.println(num);

}

}

**public** **static** **void** printNumbers(String[] companies) {

**for** (String company : companies) {

System.*out*.println(company);

}

}

}

### Important points about Sort method of Arrays:

Here are some important and worth noting point about sort method of Arrays class for quick references:

1. Arrays.sort() is an [overloaded method](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html) and can sort int, byte, short, char or object[] arrays.

2. Arrays.sort)( also allows you to sort a sub-array or part of array in Java.

3. Arrays class also contains utility methods e.g. binarySearch for performing search on array

That’s all on **how to sort java array on ascending and descending order** and part of array in Java. You can also use the workaround by converting array into arraylist and than sorting but I think Arrays.sort() is preferred method for sorting arrays in Java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*