**Bubble sort:** In this sorting element bubbled in a order, thats why its called bubble sort.

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

public static void bubbleSort(int[] arr){  
for (int lastUnSortedIndex = arr.length -1; lastUnSortedIndex > 0 ; lastUnSortedIndex--) {  
 for (int i = 0; i < lastUnSortedIndex ; i++) { // after each iteration greatest element will come out  
 if(arr[i]>arr[i+1]){  
 *swap*(arr ,i, i+1);  
 }  
 }  
 }

}

**Selection Sort:**  Main Problem with bubble sort is that its swap elements most of time un necessarily, to avoid these swap is avoid in most of elements.

In this sort we select largest element index and swap with last elements in arrays

public static void selectionSort(int[] arr){  
  
 for (int lastUnSortedIndex = arr.length-1; lastUnSortedIndex >0 ; lastUnSortedIndex--) {  
 int largest =0;  
 for (int i = 1; i <= lastUnSortedIndex; i++) {  
 if(arr[i]>arr[largest]) {  
 largest =i; //selecting largest element index in array  
 }  
 }  
 *swap*(arr, largest,lastUnSortedIndex);  
 }  
}

**Insertion sort:**  In insertion sort we take a un sorted element and inserted at correct position.

Take examples of playing card where you pick a card and insert this card at sorting position in yours hand

public static void sort(int[] arr){  
 for (int firstUnSortedIndex = 1; firstUnSortedIndex < arr.length ; firstUnSortedIndex++)  
 {  
 int insertedElement = arr[firstUnSortedIndex]; //take inserted elements  
 int i;  
 for (i = firstUnSortedIndex; i >0 && arr[i-1]>insertedElement ; i--) //moved the elements in //subset of sorted elements  
 {  
 arr[i] = arr[i-1];  
 }  
 arr[i] =insertedElement;  
 }  
}

**Quick sort:**

This is divide and conquer algorithm

*//first get the partionIndex, where all the element in left would less than  
// and right of it all the elements are greater than  
// after fisrt iteration pivat is 5  
// int[] unSortedArr = {1,-17,-9,5};  
//sort the elements from start to partition index  
//and from partition index to end of elements  
//this is again call recursive, and first sort the elements of left side  
//then sort the element of right side recursively*private static void quickSort(int[] unSortedArr, int start, int end) {  
 if(start <end){  
 int pIndex = *partition*(unSortedArr, start, end);  
  
 *quickSort*(unSortedArr, start, pIndex -1);  
 *quickSort*(unSortedArr, pIndex + 1, end);  
 }  
}  
  
*//chosen last element as pivot  
//will move all the elements  
//such that all the elements less then pivot would be left side of pivot  
// and all the elements greater then pivot would be right side of pivot  
//finally swap the partionIndex with pivot  
//than return the partition index*private static int partition(int[] unSortedArr, int start, int end) {  
 int pivot = unSortedArr[end];  
 int partionIndex = start;  
 for (int i = start; i < end ; i++) {  
 if(unSortedArr[i] <=pivot){  
 *swap*(unSortedArr, i, partionIndex);  
 partionIndex ++;  
 }  
 }  
 *swap*(unSortedArr, partionIndex, end); *//swap the pivot/end index with partition index* return partionIndex;  
  
}