```java
public static void selectionSort(int[] v, int vSize){
    for(int i = 0; i < vSize; i++){
        // trovo la posizione del minimo tra I e vSize
        int minPos = findMinPosition(v, vSize, i);
        // swappo v[i], ovvero la prima cella non ordinata disponibile
        // con v[min] ovvero il valore minimo dell'array disordinato
        swap(v, minPos, i);
    }
}

private static int findMinPosition(int[] v, int vSize, int from){
    int min = v[from], minPos = from;
    for(int i = from + 1 ; i < vSize; i++)
        if(v[i] < min){ // mi salvo il minimo
            min = v[i];
            minPos = i;
        }

    return minPos;
}

private static void swap (int[] v, int min, int first){
    int temp = v[first];
    v[first] = v[min];
    v[min] = temp;
}
```

```java
public static void mergeSort(int[] v, int vSize){
    if(vSize < 2) // se un array ha un solo elemento, per definizione è ordinato
        return;

    // trovo l'indice medio
    int mid = vSize/2;

    // creo due array che conterranno la prima e la seconda metà di v
    int[] left = new int[mid];
    int[] right = new int[vSize - mid]; // necessario vSize - mid per le lunghezze dispari

    // copio la prima e la seconda metà di v nei due array
    System.arraycopy(v, 0, left, 0, mid);
    System.arraycopy(v, mid, right, 0, vSize - mid);

    // passo ricorsivo
    mergeSort(left, mid);
    mergeSort(right, vSize - mid);

    // unisco i valori dei due array su quello originale
    merge(v, left, right);
}

private static void merge(int[] v, int[] left, int[] right){
    int vIndex = 0, leftIndex = 0, rightIndex = 0;

    // finche non finisco i valori su uno o sull'altro array continuo ad iterare
    while(leftIndex < left.length && rightIndex < right.length)
        if(left[leftIndex] < right[rightIndex]) // salvo il minore dei due valori in v
            v[vIndex++] = left[leftIndex++];
        else
            v[vIndex++] = right[rightIndex++];

    // i due while servono nel caso siano rimasti ancora dei valori su uno dei due array
    // non sapendo quale dei due sia faccio il while su entrambi
    while(leftIndex < left.length)
        v[vIndex++] = left[leftIndex++];

    while(rightIndex < right.length)
        v[vIndex++] = right[rightIndex++];
}
```

```java
/*
    Questo algoritmo di ordinamento è ottimo per ordinare un array che è
    già parzialmente ordinato o meglio, che ad ogni inseriemnto deve essere ordinato nuovamente.
    Nel caso migliore infatti le sue prestazioni sono N e non NlogN come il mergeSort oppure N^2 come il selection.
*/

public static void insertionSort(int[] v, int vSize){
    for(int i = 0; i < vSize; i++){
        // salvo il primo valore della parte non ordinata dell'array
        int temp = v[i];

        int j;
        // controllo la parte già ordinata dell'array
        // finchè temp è minore dei numeri della parte ordinata
        // sposto tutti i valori maggiori di temp di una casella a destra
        for(j = i; j > 0 && temp < v[j - 1]; j--) // se i dati sono già ordinati questo for non itera mai
            v[j] = v[j - 1];
        // inserisco temp
        v[j] = temp;
    }
}
```

```java
public static void bubbleSort(int[] v, int vSize){
    boolean done = false;
    while(!done){
        done = true;
        for(int i = 0; i < vSize - 1; i++)
            if(v[i] > v[i + 1]){
                // se fa almeno uno swap vuol dire che
                // l'array deve ancora essere ordinato del tutto
                swap(v, i, i + 1);
                done = false;
            }
    }
}

private static void swap(int[] v, int pos1, int pos2){
    int temp = v[pos1];
    v[pos1] = v[pos2];
    v[pos2] = temp;
}
```