Classe SelSort.java

```
public class SelSort
   /**
       Trova l'elemento più piccolo nella porzione finale di un array.
       @param a l'array da esplorare
       @param from la posizione iniziale dell'intervallo
       @return la posizione dell'elemento più piccolo della porzione
        a[from]...a[a.length - 1]
    */
   {  public static int minimumPosition(int[] a, int from)
      {  int minPos = from;
         for (int i = from + 1; i < a.length; i++)
            if (a[i] < a[minPos]) minPos = i;
         return minPos;
      }
```

```
/**
      Ordina un array.
      @param a  l'array da ordinare
*/
  public static void sort(int[] a)
  {  for (int n = 0; n < a.length - 1; n++)
     {  int minPos = minimumPosition(a, n);
        if (minPos != n)
           ArrayUtil.swap(a, minPos, n);
     }
  }
}
```

Classe ArrayUtil.java

```
/**
    Questa classe contiene metodi di utilità per la manipolazione di un array.
*/

public class ArrayUtil
{/**
        Scambia due elementi in un array.
        @param a l'array con gli elementi da scambiare
        @param i l'indice del primo elemento
        @param j l'indice del secondo elemento
   */
   public static void swap(int[] a, int i, int j)
   {  int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

Classe MergeSort.java

```
public class MergeSort
{  /**
       Fonde due intervalli adiacenti e ordinati di un array in un unico intervallo
       ordinato
       @param a  l'array con le voci da fondere
       @param from  l'indice del primo elemento del primo intervallo
       @param mid  l'indice dell'ultimo elemento del primo intervallo
       @param to  l'indice dell'ultimo elemento del secondo intervallo
   */
   public static void merge(int[] a, int from, int mid, int to)
   {  int n = to - from + 1;
          //  dimensione dell'intervallo da fondere
         //  fonde entrambe le metà in un array temporaneo  b
      int[] b = new int[n];
      int i1 = from;
         //  prossimo elemento da considerare nel primo intervallo
      int i2 = mid + 1;
         //  prossimo elemento da considerare nel secondo intervallo
```

```
    int j = 0;
        // prossima posizione libera in  b  in cui copiare il piu piccolo
        // tra a[i1] e a[i2]

       // il ciclo viene ripetuto fino ad esaurire uno dei due sottointervalli
    while (i1 <= mid && i2 <= to)
    {  if (a[i1] < a[i2])
       {  b[j] = a[i1]; i1++; }
       else
       {  b[j] = a[i2]; i2++; }

       j++;
    }
    // si osservi che viene eseguito uno solo dei due cicli while  che seguono
    // copia eventuali elementi rimanenti del primo sottointervallo
    while (i1 <= mid)
    {  b[j] = a[i1]; i1++; j++; }
    // copia eventuali elementi rimanenti del secondo intervallo
    while (i2 <= to)
    {  b[j] = a[i2]; i2++; j++;}
    // ricopia dall'array temporaneo nell'array originario
    for (j = 0; j < n; j++) a[from + j] = b[j];
}
```

```
/**
    Ordina una porzione di un array, usando l'algoritmo  dell'ordinamento per fusione.
    @param a  l'array da ordinare
    @param from  il primo indice dell'intervallo da ordinare
    @param to  l'ultimo indice dell'intervallo da ordinare
*/
public static void MergeSort(int[] a, int from, int to)
{  if (from < to)
   {
   int mid = (from + to) / 2;
    // ordina la prima e la seconda metà
   mergeSort(a, from, mid);
   mergeSort(a, mid + 1, to);

   // fonde le due metà ordinate
   merge(a, from, mid, to);
   }
}
/**
    Ordina un array, usando l'algoritmo di ordinamento per fusione.
    @param a  l'array da ordinare
*/
public static void sort(int[] a)
{  mergeSort(a, 0, a.length - 1);
}
}
```

Classe LinearSearch.java

```
public class LinearSearch
{
    /**
        Trova un valore in un array, usando l'algoritmo della ricerca lineare.
        @param a  l'array
        @param v  il valore da cercare
        @return  l'indice nel quale si trova il valore, oppure –1  se il valore non è
                 presente nell'array
    */
    public static int search(int[] a, int v)
    {
        int i=0;
        boolean trovato = false;
        while (!trovato && i<a.length)
            {
              if (a[i]==v) trovato = true;
                 else i=i+1;
            }
        if (!trovato) i=-1;
        return i;
    }
```

Classe BinarySearch.java

```
public class BinarySearch
{
  /**
     Trova un valore in un intervallo di un array ordinato con l'algoritmo della ricerca binaria
     @param a  un array ordinato
     @param from  il primo indice nell'intervallo in cui cercare
     @param to  l'ultimo indice nell'intervallo in cui cercare
     @param v  il valore da cercare
     @return  l'indice nel quale si trova il valore, oppure -1  se non è presente nell'array
  */
  public static int binarySearch(int[] a, int from, int to, int v)
  {  if (from > to) return -1;
     else {
          int mid = (from + to) / 2;
          if (a[mid] == v)  return mid;
          else
              if a[mid] < v
                  return binarySearch(a, mid + 1, to, v);
              else
                  return binarySearch(a, from, mid - 1, v);
           }
  }
```

```
    /**
        Trova un valore in un array ordinato, utilizzando l'algoritmo
          della ricerca binaria.
        @param a  l'array ordinato
        @param v  il valore da cercare
        @return  l'indice nel quale il valore si trova oppure -1
        se non è presente nell'array
    */
    public static int search(int[] a, int v)
    {  return binarySearch(a, 0, a.length -1, v);
    }
}
```