1. Scrivere un metodo per la ricerca *lineare* o *sequenziale* di un intero `v` in un array di interi `a`. Se `v` occorre in `a`, il metodo restituisce la posizione della prima occorrenza di `v` in `a`. Altrimenti, il metodo restituisce `-1`.

```
public static int ricercaSeq (int[] a, int v) {
  for (int i=0; i<a.length; i++) {
    if (a[i] == v)
      return i;
  }
  return -1;
}
```

2. Scrivere un metodo che restituisce la posizione del valore minimo di un array di interi a partire da una data posizione dell'array.

```
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;
}
```

3. Scrivere un metodo che scambia gli elementi nelle posizioni `i` e `j` di un array `a` di interi.

```
public static void swap (int[] a, int i, int j) {
  int temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}
```

4. Scrivere un metodo che ordina un array di interi in senso crescente tramite selezione del valore minimo (utilizzando i metodi definiti ai punti precedenti).

```
public static void sort (int[] a) {
  for (int n=0; n<a.length-1; n++) {
    int minPos = minimumPosition(a,n);
    if (minPos != n)
      swap(a,minPos,n);
  }
}
```

Esercizio
Scrivere un metodo che ordina un array di interi in senso decrescente tramite selezione del valore massimo.

5. Scrivere un metodo *iterativo* che effettua la *ricerca binaria* di un intero in un array di interi (ordinato in senso crescente).

```
public static int RicercaBinariaIt (int[] a, int v) {
  int from = 0;
  int to = a.length-1;
  while (from <= to) {
    int mid = (from+to)/2;
    if (a[mid] == v)
      return mid;
    else
      if (a[mid] < v)
        from = mid+1;
      else
        to = mid-1;
  }
  return -1;
}
```

6. Scrivere un metodo *ricorsivo* che effettua la *ricerca binaria* di un intero in un array di interi (ordinato in senso crescente).

```
public static int RicercaBinaria (int[] a, int v) {
  return RicercaBinaria(a,0,a.length-1,v);
}

public static int RicercaBinaria (int[] a, int from, int to, int v) {
  if (from > to) return -1;
  int mid = (from+to)/2;
  if (a[mid] == v)
    return mid;
  else
    if (a[mid] < v)
      return RicercaBinaria(a,mid+1,to,v);
    else
      return RicercaBinaria(a,from,mid-1,v);
}
```

Esercizio
Dire che cosa fa (o che cosa non fa) il metodo seguente `swapInt`. Giustificare la risposta.

```
public static void swapInt (int n, int m) {
  int temp = n;
  n = m;
  m = temp;
}
```

1. (*Esame scritto del 17/7/2002, Esercizio 4.*)
   Scrivere un metodo ricorsivo che dato un array di interi restituisca `true` se tutti i suoi elementi sono identici, e `false` altrimenti.

```
public static boolean ugualiRic (int[] a) {
  return ugualiRic(a,0);
}

public static boolean ugualiRic (int[] a, int i) {
  if (i == a.length-1)
    return true;
  if (a[i] != a[i+1])
    return false;
  return ugualiRic(a,i+1);
}
```

2. (*Esame scritto del 18/7/2003, Esercizio 4*)
   Scrivere un metodo ricorsivo che, dato un array `a` di interi, restituisce la somma alternante di `a`, ovvero il valore ottenuto aggiungendo gli elementi di `a` in posizione pari e sottraendo gli elementi di `a` in posizione dispari. Ad esempio, dato l'array `a = {7,3,4,-6,-11}`, il metodo restituisce `3`.

```
public static int altSum (int[] a) {
   return altSum(a,0,0);
}

public static int altSum (int[] a, int sum, int i) {
  if (i == a.length) return sum;
  if (i%2 == 0) return altSum(a,sum+a[i],i+1);
  return altSum(a,sum-a[i],i+1);
}
```

3. (*Esame scritto del 10/12/2001, una versione dell'Esercizio 4.*)
   Si consideri il seguente metodo ricorsivo:

```
public static String metodo (String s) {
  if (s.length() == 0)
    return "";
  else
    return s.charAt(0) + metodo(s.substring(1)) + s.charAt(0);
}
```

   Determinare la stringa restituita dalla chiamata `metodo("ciao")`. Scrivere inoltre un metodo non ricorsivo in linguaggio Java che realizzi la stessa funzione.

   Soluzione
   La stringa restituita dalla chiamata `metodo("ciao")` è `"ciaooaic"`.
   Un metodo non ricorsivo che realizza la stessa funzione è il seguente:

```
public static String metodoNonRic (String s) {
 String aux = s;
 for (int i = s.length()-1; i >= 0; i--)
  aux = aux + s.charAt(i);
 return aux;
}
```

4. Una stringa si dice *palindroma* se è identica al suo contrario, ovvero la stringa, letta da sinistra a destra, risulta uguale alla stringa letta da destra a sinistra. Ad esempio, `"abba"`, `"radar"`, etc. sono palindrome. Scrivere un metodo ricorsivo che controlla se una stringa è palindroma.

```
public static boolean palindroma (String s) {
  int n = s.length();
  if (n<2) return true;
  if (s.charAt(0) == s.charAt(n-1)) {
    String t = s.substring(1,n-1);
    return palindroma(t);
  }
  else return false;
}
```

Una soluzione senza generare una nuova stringa ad ogni chiamata ricorsiva è la seguente:

```
public static boolean palindroma (String s) {
  return palindroma(s,0,s.length()-1);
}

public static boolean palindroma (String s, int i, int j) {
  if (i >= j)                // stringa vuota o con un solo carattere
    return true;
  if (s.charAt(i) != s.charAt(j))
    return false;
  return palindroma(s,i+1,j-1);
}
```

5. Scrivere un metodo ricorsivo che stampa le mosse del problema delle Torri di Hanoi per un numero `n` di dischi dato in ingresso (per la descrizione del problema vedere Cay S. Horstmann, "Concetti di informatica e fondamenti di JAVA 2", Seconda edizione, Apogeo, 2002, pag. 639).

```
public static void hanoi (int n, String sorg, String dest, String tr) {
  if (n == 1)
    System.out.println("Sposta disco dal piolo "+sorg+" al piolo "+dest);
  else {
    hanoi(n-1,sorg,tr,dest);
    hanoi(1,sorg,dest,tr);
    hanoi(n-1,tr,dest,sorg);
  }
}
```