Metodi iterativi con array monodimensionali

1. Scrivere un metodo che, dato un array di interi `a`, restituisce il valore minimo in `a`.

```
public static int minimo (int[] a) {
  int min = a[0];
  for (int i=1; i<a.length; i++)
    if (a[i] < min)
      min = a[i];
  return min;
}
```

Esercizio
Scrivere un metodo che, dato un array di interi `a`, restituisce il valore massimo in `a`.

2. Scrivere un metodo che, dati un array di interi `a` ed un intero `n`, restituisce il numero delle occorrenze di `n` in `a`.

```
public static int occorrenze (int[] a, int n) {
  int cont = 0;
  for (int i=0; i<a.length; i++) {
    if (a[i] == n)
      cont++;
  }
  return cont;
}
```

3. Scrivere un metodo che, dato un array di interi `a`, restituisce la somma degli elementi in `a`.

```
public static int sommaA (int[] a) {
  int sum = 0;
  for (int i=0; i<a.length; i++) {
     sum = sum + a[i];
  }
  return sum;
}
```

Esercizio
Scrivere un metodo che, dato un array di interi `a`, restituisce il prodotto degli elementi in `a`.

4. Scrivere un metodo che, dati un array di interi `a` ed un intero `n`, restituisce `true` se `n` compare in `a`, `false` altrimenti.

```
public static boolean occorre (int[] a, int n) {
  int i = 0;
  boolean trovato = false;
```

```
  while (i<a.length && !trovato) {
    if (a[i] == n)
      trovato = true;
    i++;
  }
  return trovato;
}
```

Un metodo equivalente che sfrutta il costrutto `return` senza introdurre una variabile booleana è il seguente:

```
public static boolean occorre (int[] a, int n) {
  int i = 0;
  while (i<a.length) {
    if (a[i] == n)
      return true;
    i++;
  }
  return false;
}
```

5. Scrivere un metodo che, dati un array di interi `a` ed un intero `n`, restituisce la posizione della prima occorrenza di `n` in `a`, e `-1` se `n` non compare in `a`.

```
public static int posizione (int[] a, int n) {
  int i = 0;
  boolean trovato = false;
  while (i<a.length && !trovato) {
    if (a[i] == n)
      trovato = true;
    else i++;
  }
  if (trovato)
    return i;
  else return -1;
}
```

6. Scrivere un metodo che, dati un array di interi `a` ed un intero `k`, restituisce `true` se in `a` compaiono *almeno* `k` numeri strettamente positivi, `false` altrimenti.

```
public static boolean almenokPos (int[] a, int k) {
  int i = 0, cont = 0;
  while (i<a.length) {
    if (a[i] > 0) {
      cont++;
      if (cont >= k) return true;
    }
    i++;
  }
  return false;
}
```

In modo equivalente, mettendo la condizione `(cont < k)` nella guardia del `while`, si può scrivere:

```
public static boolean almenokPos (int[] a, int k) {
  int i = 0, cont = 0;
  while (i<a.length && cont < k) {
    if (a[i] > 0) cont++;
    i++;
  }
  return (cont >= k);
}
```

7. (*Esame scritto del 17/7/2002, Esercizio 3.*)
Scrivere un metodo in linguaggio Java che dato un array di interi restituisca `true` se tutti i suoi elementi sono identici, e `false` altrimenti.

```
public static boolean uguali (int[] a) {
  boolean equal = true;
  int i = 0;
  while(i<a.length-1 && equal) {
    if (a[i] != a[i+1])
      equal = false;
    else
      i++;
  }
  return equal;
}
```

8. (*Esame scritto del 18/9/2002, Esercizio 3.*)
Scrivere un metodo in linguaggio Java che, dato un array di interi `a`, restituisca `true` se i suoi elementi sono in ordine non decrescente (`a[0]` $\leq$ `a[1]` $\leq \ldots$), e `false` altrimenti.

```
public static boolean nondecr (int[] a) {
  boolean cond = true;
  int i = 0;
  while (i<a.length-1 && cond) {
    if (a[i] > a[i+1]) {
      cond = false;
    }
    i++;
  }
  return cond;
}
```

Un semplice metodo `main` per provare i metodi `uguali` e `nondecr` è il seguente (si assume che entrambi i metodi siano definiti in una classe `ArrayProp`):

```
public static void main (String[] args) {
  int[] a = new int[args.length];
  for (int i=0; i<a.length; i++) {
     a[i] = Integer.parseInt(args[i]);
  }
  System.out.println("Elementi identici? " + ArrayProp.uguali(a));
  System.out.println("Elementi non decrescenti? " + ArrayProp.nondecr(a));
}
```

Esercizio
Scrivere una versione diversa per i metodi `uguali` e `nondecr` in cui, invece di definire la variabile booleana, vengono utilizzate opportune istruzioni di `return`.

9. (*Esame scritto del 14/7/2004, Esercizio 3.*)
Scrivere un metodo che, dati due array a e b di interi, restituisce *true* se tutti gli elementi dell'array b compaiono nell'array a nello stesso ordine in cui compaiono in b, altrimenti il metodo restituisce *false*. Ad esempio, dati gli array `a = {-5,4,7,-1,10,21,9,-7}` e `b = {4,-1,9,-7}`, il metodo restituisce *true*.

```
public static boolean ordineArray (int[] a, int[] b) {
  int i = 0, j = 0;
  while (j < b.length && i < a.length) {
    if (b[j] == a[i]) {
      i++;
      j++;
    }
    else i++;
  }
  if (j == b.length) return true;
  else return false;
}
```

10. (*Esame scritto del 18/9/2002, Esercizio 1.*)
Si consideri il seguente metodo in linguaggio Java.

```
public static boolean metodo (int[] a) {
  boolean condizione = true;
  for (int i = 0; i < a.length-1; i++) {
    if (a[i] != 2*a[i+1]) {
      condizione = false;
    }
  }
  return condizione;
}
```

   (a) Determinare quale caratteristica deve soddisfare l'array `a` perché il metodo restituisca il valore `true`.

   (b) Riscrivere il metodo usando il ciclo `while`. Spiegare quale delle due soluzioni sia la piú efficiente e il perché.

Soluzione
1. Affinché il metodo restituisca il valore `true`, gli elementi dell'array `a` devono essere ciascuno il doppio dell'elemento successivo in `a`.
2. Soluzione con il `while`:

```
public static boolean metodo (int[] a) {
  boolean condizione = true;
  int i = 0;
  while (i < a.length-1 && condizione) {
    if (a[i] != 2*a[i+1]) {
      condizione = false;
    }
  i++;
  }
  return condizione;
}
```

La soluzione piú efficiente è quella con il `while`, in quanto nel caso in cui l'array non soddisfi la caratteristica detta al punto 1., l'iterazione termina non appena si trova un elemento dell'array che non è il doppio del successivo, senza scandire tutto l'array. Tale scansione di tutto l'array viene invece sempre eseguita dalla soluzione con il `for` indipendentemente dal valore finale di `condizione`.