1. Scrivere un metodo ricorsivo che calcola il fattoriale di un numero naturale `n`. Si ricorda la definizione del fattoriale: $0! = 1$ ed $n! = n \times (n-1)!$ per $n \geq 1$. Se al metodo viene passato un numero intero negativo, il metodo restituisce `-1`.

```
public static int fattR (int n) {
  if (n < 0) return -1;
  if (n == 0) return 1;                // caso base
  return n*fattR(n-1);                 // clausola ricorsiva
}
```

   Questo metodo, che corrisponde all'usuale definizione matematica ricorsiva del fattoriale, è detto *metodo con ricorsione annidata (nested recursion)*. Un metodo *con ricorsione in testa (tail-recursive)* per il fattoriale è il seguente:[1]

```
public static int fattTR (int n) {
  if (n < 0) return -1;
  return fattTR(n,1);
}

public static int fattTR (int n, int r) {
  if (n == 0) return r;                  // caso base
  return fattTR(n-1,n*r);                // clausola ricorsiva
}
```

2. Scrivere un metodo ricorsivo che calcola l'`n`-esimo numero di Fibonacci. Si ricorda la definizione dei numeri di Fibonacci: $fib(0) = 1$, $fib(1) = 1$, $fib(n) = fib(n-1) + fib(n-2)$ per $n \geq 2$. Se al metodo viene passato un numero intero negativo, il metodo restituisce `-1`.

```
public static int fibR (int n) {
  if (n < 0) return -1;
  if (n <= 1) return 1;                // casi base
  return fibR(n-1) + fibR(n-2);        // clausola ricorsiva
}
```

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

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

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

[1] Notare come il metodo venga sovraccaricato (overloaded) e come sia necessario introdurre ulteriori parametri nel metodo ricorsivo.

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

   Soluzione con ricorsione annidata

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

public static int occorrenzeRic (int[] a, int n, int i) {
  if (i == a.length)
    return 0;
  if (a[i] == n)
    return 1 + occorrenzeRic(a,n,i+1);
  return occorrenzeRic(a,n,i+1);
}
```

   Soluzione con ricorsione in testa

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

public static int occorrenzeRic (int[] a, int n, int i, int c) {
  if (i == a.length)
    return c;
  if (a[i] == n)
    return occorrenzeRic(a,n,i+1,c+1);
  return occorrenzeRic(a,n,i+1,c);
}
```

5. Scrivere un metodo ricorsivo 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 posizioneRic (int[] a, int n) {
  return posizioneRic(a,n,0);
}

public static int posizioneRic (int[] a, int n, int i) {
  if (i == a.length)
    return -1;
  if (a[i] == n)
    return i;
  return posizioneRic(a,n,i+1);
}
```

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

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

```
public static boolean occorreRic (int[][] a, int n, int i, int j) {
  if (i == a.length)
    return false;
  if (j == a[i].length)
    return occorreRic(a,n,i+1,0);
  if (a[i][j] == n)
    return true;
  return occorreRic(a,n,i,j+1);
}
```

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

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

public static int occorrenzeRic (int[][] a, int n, int i, int j, int c) {
  if (i == a.length)
    return c;
  if (j == a[i].length)
    return occorrenzeRic(a,n,i+1,0,c);
  if (a[i][j] == n)
    return occorrenzeRic(a,n,i,j+1,c+1);
  return occorrenzeRic(a,n,i,j+1,c);
}
```