Programmazione
Si scriva un metoo "rollup" che riceve in input un array l di lunghezza n (con n pari) e restituisce un array di lunghezza n/2 il cui i-esimo elemento è pari a l[2*i]+l[2*i+1].

Soluzione:

```
public static int[] rollup(int[] l) {
    int[] ris = new int[l.length / 2];
    for (int i = 0; i < l.length / 2; i++)
        ris[i] = l[2 * i] + l[2 * i + 1];
    return ris;
}
```

Programmazione
Si scriva un metodo "alternati" che riceve in input un array l e restituisce true se l'array contiene valori alternati pari e dispari, cioè non ci sono due elementi consecutivi entrambi pari o entrambi dispari, e false altrimenti.

Soluzione:

```
public static boolean alternati(int[] l) {
    boolean precedente_pari = l[0] % 2 == 0;
    for (int i = 1; i < l.length; i++) {
        if ((l[i] % 2 == 0 && precedente_pari) ||
                                    (l[i] % 2 == 1 && !precedente_pari))
                            return false;
        precedente_pari = l[i] % 2 == 0;
       //alternativa: precedente_pari = !precedente_pari
    }
    return true;
}
```

Programmazione

Si scriva un metodo "costruisciArray" che riceve in input un array di interi V1 e restituisce un array di interi V2 di pari lunghezza, il cui generico elemento i è così ottenuto:

- Se la media degli elementi di V1 con indice maggiore o uguale a i è maggiore o uguale a V1[i], allora V2[i] è uguale a tale media.
- Altrimenti, V2[i] è uguale alla differenza tra la somma degli elementi alla sinistra di V1[i] e la somma degli elementi alla destra di V1[i] (ovviamente se non c'è nessun elemento alla destra o alla sinistra tale somma vale zero).

Soluzione:

```
public static int[] costruisciArray(int[] V1) {
    int[] V2 = new int[V1.length];
    for (int i = 0; i < V1.length; i++) {
        int somma = 0;
        for (int j = i; j < V1.length; j++)
            somma += V1[j];
        int media = somma/((V1.length) - i);
        if (media >= V1[i])
            V2[i] = media;
        else {
            int somma1 = 0;
            for (int j = 0; j < i; j++)
                somma1 += V1[j];
            int differenza = somma1 - (somma - V1[i]);
            V2[i] = differenza;
        }
    }
    return V2;
}
```

Programmazione

Si scrivano:

1. un metodo *costruisciMatrice* che riceve una matrice di interi *M* e restituisce una matrice di boolean *M'* della stessa dimensione tale che
   a. le celle di *M'* sulla cornice esterna contengono tutte il valore *false* e
   b. ogni altra cella *M'[i,j]* contiene il valore *true* se e solo se la somma delle celle adiacenti ad *M[i,j]* è uguale al valore di *M[i,j]*.
2. un metodo *verificaMatrice* che riceve in ingresso una matrice di interi *M* e restituisce *true* se e solo se le colonne pari sono ordinate in modo non crescente e quelle dispari in modo non decrescente;

Soluzione:

```
public static boolean[][] costruisciMatrice(int[][] M)
{       boolean[][] M1 = new boolean[M.length][M[0].length];
        for(int  j=0;j<M[0].length;j++)
        {       M1[0][j] = false;
                M1[M.length-1][j] = false;
        }
        for(int  i=1;i< M.length-1;i++)
        {       M1[i][0] = false;
                M1[i][M[0].length-1] = false;
        }

        for(int  i=1;i< M.length-1;i++)
                for(int  j=1;j<M[0].length-1;j++)
                {       int sommaAdiacenti=0;
                        sommaAdiacenti+= M[i-1][j-1] + M[i-1][j] + M[i-1][j+1];
                        sommaAdiacenti+= M[i][j-1] + M[i][j+1];
                        sommaAdiacenti+= M[i+1][j-1] + M[i+1][j] + M[i+1][j+1];
                        M1[i][j] = (M[i][j]==sommaAdiacenti);
                }
        return M1;
}

public static boolean verificaMatrice(int[][] M)
{       for(int  j=0;j<M[0].length;j++)
                if(j%2==0)
                        for(int i=0;i<M.length-1;i++)
                                if(M[i][j]<M[i+1][j])
                                        return false;
                else
                        for(int i=0;i<M.length-1;i++)
                                if(M[i][j]>M[i+1][j])
                                        return false;
        return true;
}
```