Laboratorio di Algoritmi e Strutture Dati
(Laboratory of Algorithms and Data Structures)

Guido Fiorino
guido.fiorino@unimib.it

Exercises
(Last Modified: 09-04-2018)

A collection of exercises on divide-and-conquer method

1. Use the technique divide-and-conquer to write a Java method `int count10(int[] a, int left, int right)` that returns the frequency of the sequence $\langle 1, 0 \rangle$ in the array `a` spanning from `left` to `right` (solution on page 4);
2. use the technique divide-and-conquer to write a Java method `int findBBA(char[] a, int left, int right)` that returns the frequency of the string ``‘‘BBA’’`` in the array `a` spanning from `left` to `right`;
3. use the technique divide-and-conquer, write a Java method `int countOOE(int[] a, int left, int right)` that returns the number of times that in the array `a` spanning from `left` to `right` an even number is immediately preceded by two odd numbers (solution on page 14);
4. use the technique divide-and-conquer, write a Java method `int countTrueTrue(boolean[] a, int left, int right)` that returns the frequency of $\langle$`true`, `true`$\rangle$ in the array `a` spanning from `left` to `right`. Note that if `a = [ true, true, true ]`, then `countTrueTrue` returns 2.
5. use the technique divide-and-conquer, write a Java method `boolean evenVowels(char[] a, int left, int right)` that returns `true` if in the array `a` spanning from `left` to `right` the number of vowels is even, `false` otherwise;
6. use the technique divide-and-conquer to write a Java method `int count1001(int[] a, int left, int right)` that returns the frequency of the sequence $\langle 1, 0, 0, 1 \rangle$ in the array `a` spanning from `left` to `right` (solution on page 20);

A collection of exercises on divide-and-conquer method (2)

7. use the technique divide-and-conquer to write a Java method `int sumElementsBetween(int[] a, int Min, int Max, int left, int right)` that returns the sum of the elements of the array `a` whose value is included in the range between `Min` and `Max`. The array `a` spans from `left` to `right`;
8. use the technique divide-and-conquer to write a Java method `int freqTwo(int[] a, int X, int Y, int left, int right)` that returns the frequency of the sequence $\langle \mathtt{X}, \mathtt{Y} \rangle$ in the array `a` spanning from `left` to `right`;
9. use the technique divide-and-conquer to write a Java method `int countZeroZero(boolean[] a, int left, int right)` that returns the frequency of $\langle 0, 0 \rangle$ in the array `a` spanning from `left` to `right`. See comment to exercise 4;
10. use the technique divide-and-conquer to write a Java method `int sumOfMult(int[] a, int left, int right)` that returns the value of the expression $\sum_{i=left}^{right-1} \mathtt{a[i]} * \mathtt{a[i+1]}$;
11. use the technique divide-and-conquer to write a Java method `int countInc(int[] a, int left, int right)` that returns how many times in the array `a` spanning from `left` to `right` it is fulfilled the condition $a[i] > a[i+1]$, for $i = left, \ldots, right - 1$ (solution on page 31);
12. Use the technique divide-and-conquer to write a Java method `int count111(int[] a, int left, int right)` that returns the frequency of the sequence $\langle 1, 1, 1 \rangle$ in the array `a` spanning from `left` to `right` (solution on page 36);

Solution to Exercise 1

- If the array `a` has less than two elements, then there are no occurrences of the sequence $\langle 1, 0 \rangle$ in `a`, thus method `count10` must return 0;
- if `a` has exactly two elements, then:
  - if `a=[1,0]`, then `count10` must return 1,
  - otherwise `count10` must return 0.

This ends the analysis of the base of the recursion.

Solution to Exercise 1

If the size of the array a is greater than two,

- we apply the divide-and-conquer method by recursively splitting a in two halves: if we know the number of occurrences of $\langle 1, 0 \rangle$ in the two halves, then we can recombine the information and discover the frequency of $\langle 1, 0 \rangle$ in a.

The recombine step must be performed with some care:

$\cdots$	1	0	$\cdots$

we must take into account that there can be a sequence that starts in the orange region (the first half) and ends in the yellow region (the second half). The recombine step is:

the frequency of $\langle 1, 0 \rangle$ in the orange region $+$
the frequency of $\langle 1, 0 \rangle$ in the yellow region $+$

$$\begin{cases} 1 & \text{if the rightmost cell of the orange region contains 1 and} \\ & \text{the leftmost cell of the yellow region contains 0} \\ 0 & \textit{otherwise} \end{cases}$$

A small improvement

By the analysis in previous slide, we get that only one base case is sufficient:

- If the array `a` has less than two elements, then there are no occurrences of the sequence $\langle 1, 0 \rangle$ in `a`, thus method `count10` must return 0.

As a matter of fact, the case

- if `a` has exactly two elements, then:
  - if `a=[1,0]`, then `count10` must return 1,
  - otherwise `count10` must return 0.

is subsumed by

$$\begin{cases} 1 & \text{if the rightmost cell of the orange region contains 1 and} \\ & \text{the leftmost cell of the yellow region contains 0} \\ 0 & \textit{otherwise} \end{cases}$$

in the recombine step. Summarizing:

we can have a solution where the recursive step is performed also when the region has two elements.

Please note that our solutions do not consider this improvement.

Solution to Exercise 1

```java
static int count10(int[] a, int left, int right){
    int inc = 0, mid;

    if ( left == right ) return 0; // one cell

    if ( right - left == 1) // two cells
        if ( a[left] == 1 && a[right] == 0 ) return 1;
        else return 0;

    mid = (left+right)/2;
    if (a[mid] == 1 && a[mid + 1] == 0) inc = 1;
    return inc + count10(a, left, mid) + count10(a, mid+1, right);
}
```

Trees of the Recursive Calls of count10

- The tree of the recursive calls depends on the size of a, that is
- the tree of the recursive calls does not depend on the content of array a.

Tree of the recursive calls on array with seven elements

```
array a = [ 0, 0, 0, 0, 0, 0, 0 ]
count10(a, 0, 6)
              count10(a, 0, 3)
                            count10(a, 0, 1)
                            count10(a, 2, 3)
              count10(a, 4, 6)
                            count10(a, 4, 5)
                            count10(a, 6, 6)

array a = [ 1, 0, 0, 0, 0, 0, 0 ]
count10(a, 0, 6)
              count10(a, 0, 3)
                            count10(a, 0, 1)
                            count10(a, 2, 3)
              count10(a, 4, 6)
                            count10(a, 4, 5)
                            count10(a, 6, 6)
```

More Trees of the Recursive Calls of `count10`

Tree of the recursive calls on array with eight elements

```
array a = [ 0, 1, 1, 1, 1, 1, 1, 0]
count10(a, 0, 7)
              count10(a, 0, 3)
                            count10(a, 0, 1)
                            count10(a, 2, 3)
              count10(a, 4, 7)
                            count10(a, 4, 5)
                            count10(a, 6, 7)

array a = [ 1, 1, 1, 1, 1, 1, 1, 1 ]
count10(a, 0, 7)
              count10(a, 0, 3)
                            count10(a, 0, 1)
                            count10(a, 2, 3)
              count10(a, 4, 7)
                            count10(a, 4, 5)
                            count10(a, 6, 7)
```

Another Solution to Exercise 1

If we change the way we split the array a, then we get a new solution, with a different recombine step.
Consider the way we split the array a:

left	mid	mid+1	right
· · ·	1	0	· · ·

The recursive call

- **count10(a, left, mid)** counts all the sequences whose first element is between `left` and `mid-1`. Similarly
- **count10(a, mid+1, right)** counts all the sequences whose first element is between `mid+1` and `right-1`.

This implies that both the recursive calls disregard the sequence crossing the region. We can change the recursive calls as follows:

left	mid	mid+1	right
· · ·	1		
	1	0	· · ·

Another Solution to Exercise 1

```
static int count10V2(int[] a, int left, int right){
    int  mid;

    if ( left == right ) return 0; // one cell

    if ( right - left == 1) // two cells
        if ( a[left] == 1 && a[right] == 0 ) return 1;
        else return 0;

    mid = (left+right)/2;

    return  count10V2(a, left, mid) + count10V2(a, mid, right);
}
```

- the regions the two recursive are called on overlap on the cell of index `mid`;
- the recursive calls are on regions that are strictly shorter than the region spanning from `left` to `right`, this proves the termination of our method `count10V2`.

Trees of the Recursive Calls of count10V2

Trees of the recursive calls on array with seven elements

```
array a = [  0, 1, 0, 0, 0, 0, 0,  ]
count10V2(a, 0, 6)
              count10V2(a, 0, 3)
                            count10V2(a, 0, 1)
                            count10V2(a, 1, 3)
                                          count10V2(a, 1, 2)
                                          count10V2(a, 2, 3)
              count10V2(a, 3, 6)
                            count10V2(a, 3, 4)
                            count10V2(a, 4, 6)
                                          count10V2(a, 4, 5)
                                          count10V2(a, 5, 6)
array a = [  1, 0, 0, 0, 1, 0, 0,  ]
count10V2(a, 0, 6)
              count10V2(a, 0, 3)
                            count10V2(a, 0, 1)
                            count10V2(a, 1, 3)
                                          count10V2(a, 1, 2)
                                          count10V2(a, 2, 3)
              count10V2(a, 3, 6)
                            count10V2(a, 3, 4)
                            count10V2(a, 4, 6)
                                          count10V2(a, 4, 5)
                                          count10V2(a, 5, 6)
```

More Trees of the Recursive Calls of count10V2

Trees of the recursive calls on array with eight elements

```
array a = [  0, 1, 1, 1, 1, 1, 1, 0 ]
count10V2(a, 0, 7)
              count10V2(a, 0, 3)
                            count10V2(a, 0, 1)
                            count10V2(a, 1, 3)
                                          count10V2(a, 1, 2)
                                          count10V2(a, 2, 3)
              count10V2(a, 3, 7)
                            count10V2(a, 3, 5)
                                          count10V2(a, 3, 4)
                                          count10V2(a, 4, 5)
                            count10V2(a, 5, 7)
                                          count10V2(a, 5, 6)
                                          count10V2(a, 6, 7)
array a = [  1, 1, 1, 1, 1, 1, 1, 1  ]
count10V2(a, 0, 7)
              count10V2(a, 0, 3)
                            count10V2(a, 0, 1)
                            count10V2(a, 1, 3)
                                          count10V2(a, 1, 2)
                                          count10V2(a, 2, 3)
              count10V2(a, 3, 7)
                            count10V2(a, 3, 5)
                                          count10V2(a, 3, 4)
                                          count10V2(a, 4, 5)
                            count10V2(a, 5, 7)
                                          count10V2(a, 5, 6)
                                          count10V2(a, 6, 7)
```

Solution to Exercise 3

- If the array a has less than three elements, then in a there is no occurrence of a sequence of the kind $\langle O, O', E\rangle$ (where with $\langle O, O', E\rangle$ we mean a sequence of two odd numbers followed by an even number). Thus method `countOOE` must return 0;
- if a has exactly three elements, then if `a[0]` and `a[1]` contain an odd number and `a[2]` contains an even number, then `countOOE` must return 1, otherwise `countOOE` must return 0.

This ends the analysis of the base of the recursion.

Solution to Exercise 3

If the size of the array a is greater than three, then we apply the divide-and-conquer method by recursively splitting a in two halves:

- if we know the number of occurrences of sequences of the kind $\langle O, O', E\rangle$ in the two halves, then we can recombine the information and discover the frequency of sequences of the kind $\langle O, O', E\rangle$ in a.

The recombine step must be performed with some care. We have to take into account two cases:

$\cdots$	3	5	0	$\cdots$

$\cdots$	3	5	0	$\cdots$

because a sequence of the kind $\langle O, O', E\rangle$ can have one or two elements in the orange region.

The recombine step in this case is:

the frequency of $\langle O, O', E\rangle$ in the orange region +
the frequency of $\langle O, O', E\rangle$ in the yellow region +
the number of sequences of the kind $\langle O, O', E\rangle$ crossing the regions. This number is between zero and one.

Solution to Exercise 3

```
static int countOOE(int[] a, int left, int right){
    int inc = 0, mid;
    if ( left == right || right-left == 1 ) return 0; // one or two cells

    if ( right - left == 2) // three cells
        if ( a[left]%2 == 1 && a[left+1]%2 == 1 && a[right]%2 == 0 ) return 1;
        else return 0;

    // if we are here a[left]...a[right] has four of more cells

    mid = (left+right)/2;

    // count the number of sequences traversing the middle of the array
    for(int i = mid - 1; i<= mid  ; i++)
        if (a[i] % 2 == 1 && a[i + 1] % 2 == 1 && a[i + 2] % 2 == 0) inc++;

    return inc + countOOE(a, left, mid) + countOOE(a, mid+1, right);
} // end function
```

Another Solution to Exercise 3

If we change the way we split the array `a`, then we get a new solution, with a different recombine step.
The elements of a sequence can be shared between the regions in two ways:

···	3	5	0	···

···	3	5	0	···

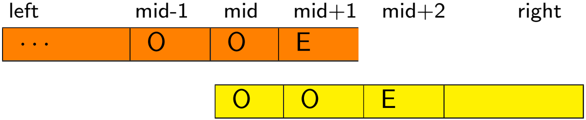
The recursive call

- **countOOE(a, left, mid)** counts all the sequences whose first element is between `left` and `mid-2`. Similarly
- **countOOE(a, mid+1, right)** counts all the sequences whose first element is between `mid+1` and `right-2`.

This implies both the recursive calls disregard the sequences crossing the region.

Another Solution to Exercise 3

We can change the recursive calls as follows:



The recursive calls have two cells of the region in common:

- the recursive call on the orange region takes into account a sequence starting in `mid-1` and ending in `mid+1`;
- the recursive call on the yellow region takes into account the sequence starting in `mid` and ending in `mid+2`

Since the region starting in `left` and ending in `right` has at least four cells, it follows that the orange and yellow region have less elements than the whole region. This proves that the method we are going to provide terminates.

Another Solution to Exercise 3

```
static int countOOEV2(int[] a, int left, int right){
    int  mid;

    if ( left == right || right-left == 1 ) return 0; // one or two cells

    if ( right - left == 2) // three cells
        if ( a[left]%2 == 1 && a[left+1]%2 == 1 && a[right]%2 == 0 ) return 1;
        else return 0;

    // if we are here a[left]...a[right] has four of more cells

    mid = (left+right)/2;

    return countOOEV2(a, left, mid+1) + countOOEV2(a, mid, right);
}
```

Solution to Exercise 6

- If the array a has less than four elements, then in a there is no occurrence of a sequence $\langle 1, 0, 0, 1 \rangle$. Thus method `count1001` must return 0;
- if a has exactly four elements, then if `a=[1,0,0,1]`, then `count1001` must return 1, otherwise `count1001` must return 0.

This ends the analysis of the base of the recursion.

Solution to Exercise 6

If the size of the array a is greater than four, then we apply the divide-and-conquer method by recursively splitting a in two halves:

- if we know the number of occurrences of the sequence $\langle 1, 0, 0, 1 \rangle$ in the two halves, then we can recombine the information and discover the frequency of sequences of the kind $\langle 1, 0, 0, 1 \rangle$ in a.

Solution to Exercise 6

The recombine step must be performed with some care. We have to take into account the following cases:

$\cdots$	1	0	0	1	$\cdots$

$\cdots$	1	0	0	1	$\cdots$

$\cdots$	1	0	0	1	$\cdots$

We see that the sequences $\langle 1, 0, 0, 1\rangle$ can start in the orange region (the first half) and end in the yellow region (the second half). Note that:

- if the array has more than five elements, then a sequence $\langle 1, 0, 0, 1\rangle$ can start in the orange region and end in the yellow region in three possible ways, correspondig to start at positons `mid - 2`, `mid-1` and `mid`, where `mid` is the index of the rightmost cell of the orange region;
- if the array has five elements, no sequence can start at index `mid`, and we have only two cases.

Solution to Exercise 6

The recombine step is:

the frequency of $\langle 1, 0, 0, 1 \rangle$ in the orange region $+$
the frequency of $\langle 1, 0, 0, 1 \rangle$ in the yellow region $+$
the number of sequences $\langle 1, 0, 0, 1 \rangle$ crossing the regions. This number is between zero and one.

Solution to Exercise 6

```
static int count1001(int[] a, int left, int right){
    int inc = 0, mid;
    if ( right - left < 3 ) return 0; // one, two or three cells
    if ( right - left == 3) // four cells
        if ( a[left]  == 1 && a[left + 1]  == 0 &&
             a[left + 2]  == 0 && a[left + 3] == 1 ) return 1;
        else return 0;

    // if we are here a[left]...a[right] has five of more cells

    mid = (left+right)/2;

    // count the number of sequences traversing the middle of the array
    // note the special control i + 3 <= right necessary in the case of array with 5 cells
    for(int i = mid - 2; i<= mid && i+3 <= right  ; i++)
        if ( a[i]  == 1 && a[i+1]  == 0 && a[i+2]  == 0 && a[i+3] == 1 ) inc++;

    return inc + count1001(a, left, mid) + count1001(a, mid+1, right);
} // end function
```

Another Solution to Exercise 6 (howto avoid the check `i + 3 <= right`)

If we change the way we split the array `a`, then we get a new solution, that avoids the special control `i + 3 <= right` necessary to take into account that in an array of five elements a sequence $\langle 1, 0, 0, 1 \rangle$ cannot start at the cell of index `mid`, that is the last of the following configurations is impossible:

$\cdots$	1	0	0	1	$\cdots$

$\cdots$	1	0	0	1	$\cdots$

$\cdots$	1	0	0	1	$\cdots$

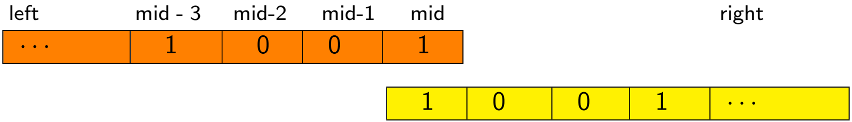
The recursive call

- `count1001(a, left, mid)` counts all the sequences whose first element is between `left` and `mid-3`. Similarly
- `count1001(a, mid+1, right)` counts all the sequences whose first element is between `mid+1` and `right-3`.

This implies both the recursive calls disregard the sequences $\langle 1, 0, 0, 1 \rangle$ crossing the regions. These sequences can start at `mid-2`, `mid-1` or `mid` (last case only for arrays with more than five elements).

Another Solution to Exercise 6 (howto avoid the check `i + 3 <= right`)

We can change the recursive calls as follows:



The recursive calls have the cell of index `mid` of the array in common:

- the recursive call on the yellow region takes into account the sequence starting in `mid` and ending in `mid+3` when the array has more than five elements

Since the region starting in `left` and ending in `right` has at least five cells, it follows that the orange and yellow region have less elements than the whole region. This proves that the method we are going to provide terminates.

Solution to Exercise 6

```
static int count1001V2(int[] a, int left, int right){
    int inc = 0, mid;
    if ( right - left < 3 ) return 0; // one, two or three cells
    if ( right - left == 3) // four cells
        if ( a[ left ]  == 1 && a[ left + 1 ]  == 0 &&
             a[ left + 2 ]  == 0 && a[ left + 3 ] == 1 ) return 1;
        else return 0;

    // if we are here a[left]...a[right] has five of more cells

    mid = (left+right )/2;

    // count the number of sequences traversing the middle of the array
    for(int i = mid - 2; i<= mid-1  ; i++)
        if ( a[i]  == 1 && a[i+1]  == 0 && a[i+2]  == 0 && a[i+3] == 1 ) inc++;

    return inc + count1001V2(a, left, mid) + count1001V2(a, mid, right);
} // end function
```

One More Solution to Exercise 6

If we change once more the way we split the array `a`, then we get a solution not containing the `for` iteration.

We recall that the elements of a sequence $\langle 1, 0, 0, 1 \rangle$ can be shared between the regions in three ways:

$\cdots$	1	0	0	1	$\cdots$

$\cdots$	1	0	0	1	$\cdots$

$\cdots$	1	0	0	1	$\cdots$

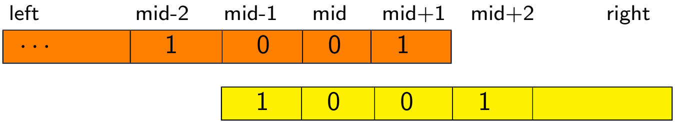
The recursive call

- `count1001(a, left, mid)` counts all the sequences whose first element is between `left` and `mid-3`. Similarly
- `count1001(a, mid, right)` counts all the sequences whose first element is between `mid` and `right-3`.

This implies both the recursive calls disregard the sequences $\langle 1, 0, 0, 1 \rangle$ that start at `mid-2` or `mid-1`.

One More Solution to Exercise 6

We can change the recursive calls as follows:



The recursive calls have three cells of the region in common:

- the recursive call on the orange region takes into account a sequence starting in `mid-2` and ending in `mid+1`;
- the recursive call on the yellow region takes into account the sequence starting in `mid-1` and ending in `mid+2`

Since the region starting in `left` and ending in `right` has at least five cells, it follows that the orange and yellow region have less elements than the whole region. This proves that the method we are going to provide terminates.

One More Solution to Exercise 6

```
static int count1001V3(int[] a, int left, int right){
    int  mid;

    if ( right-left<3 ) return 0; // one, two or three cells

    if ( right - left == 3) // four cells
        if ( a[left]  == 1 && a[left + 1]  == 0 &&
             a[left + 2]  == 0 && a[left + 3] == 1 ) return 1;
        else return 0;

    // if we are here a[left]...a[right] has four of more cells

    mid = (left + right)/2;

    return count1001V3(a, left, mid + 1) + count1001V3(a, mid-1, right);
}
```

Solution to Exercise 11

This exercise is analogous to exercise 1.

- If the array a has less than two elements, then there are no occurrences of the sequences of the kind $\langle x, y \rangle$, with $x > y$, in a, thus method `countinc` must return 0;
- if a has exactly two elements, then:
  - if a is of the kind $\langle x, y \rangle$, with $x > y$, then `countinc` must return 1,
  - otherwise `countinc` must return 0.

This ends the analysis of the base of the recursion.

Solution to Exercise 11

If the size of the array a is greater than two,

- we apply the divide-and-conquer method by recursively splitting a in two halves: if we know the number of occurrences of sequences of the kind $\langle x, y \rangle$, with $x > y$, in the two halves, then we can recombine the information and discover the frequency of sequences of the kind $\langle x, y \rangle$ in a.

The recombine step must be performed with some care:

$\cdots$	x	y	$\cdots$

we must take into account that there can be a sequence that starts in the orange region (the first half) and ends in the yellow region (the second half). The recombine step is:

the frequency of sequences of the kind $\langle x, y \rangle$, with $x > y$, in the orange region +
the frequency of sequences of the kind $\langle x, y \rangle$, with $x > y$, in the yellow region +

$$\begin{cases} 1 & \text{if the rightmost cell of the orange region contains } x \text{ and} \\ & \text{the leftmost cell of the yellow region contains } y, \text{ with } x > y \\ 0 & \textit{otherwise} \end{cases}$$

Solution to Exercise 11

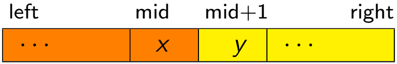
```
static int countInc(int[] a, int left, int right){
    int inc = 0, mid;
    if ( left == right ) return 0; // one cell
    if ( right - left == 1) // two cells
        if ( a[left] > a[right]  ) return 1;
        else return 0;

    mid = (left+right)/2;
    if (a[mid] > a[mid + 1] ) inc = 1;

    return inc + countInc(a, left, mid) + countInc(a, mid+1, right);
}
```

Another Solution to Exercise 11

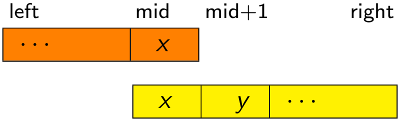
We can proceed as in the case of Exercise 1: if we change the way we split the array a, then we get a new solution, with a different recombine step.
Consider the way we split the array a:



The recursive call

- `countInc(a, left, mid)` counts all the sequences whose first element is between `left` and `mid-1`. Similarly
- `countInc(a, mid+1, right)` counts all the sequences whose first element is between `mid+1` and `right-1`.

This implies that both the recursive calls disregard the sequence crossing the region. We can change the recursive calls as follows:



Another Solution to Exercise 11

```
    static int countIncV2(int[] a, int left, int right){
    int  mid;
    if ( left == right ) return 0; // one cell
    if ( right - left == 1) // two cells
        if ( a[left] > a[right] ) return 1;
        else return 0;

    mid = (left+right)/2;


    return  countIncV2(a, left, mid) + countIncV2(a, mid, right);
}
```

- the regions the two recursive are called on overlap on the cell of index `mid`;
- the recursive calls are on regions that are strictly shorter than the region spanning from `left` to `right`, this proves the termination of our method `countIncV2`.

Solution to Exercise 12

This exercise is analogous to 3.

- If the array a has less than three elements, then in a there is no occurrence of a sequence of the kind $\langle 1, 1, 1 \rangle$. Thus method `count111` must return 0;
- if a has exactly three elements, then:
  - if `a=[1,1,1]`, then `count111` must return 1,
  - otherwise `count111` must return 0.

This ends the analysis of the base of the recursion.

Solution to Exercise 12

If the size of the array a is greater than three, then we apply the divide-and-conquer method by recursively splitting a in two halves:

- if we know the number of occurrences of sequences of the kind $\langle 1, 1, 1 \rangle$ in the two halves, then we can recombine the information and discover the frequency of sequences of the kind $\langle 1, 1, 1 \rangle$ in a.

The recombine step must be performed with some care. We have to take into account two cases:

$\cdots$	1	1	1	$\cdots$

$\cdots$	1	1	1	$\cdots$

because a sequence of the kind $\langle 1, 1, 1 \rangle$ can have one or two elements in the orange region.

The recombine step in this case is:

the frequency of $\langle 1, 1, 1 \rangle$ in the orange region $+$
the frequency of $\langle 1, 1, 1 \rangle$ in the yellow region $+$
the number of sequences of the kind $\langle 1, 1, 1 \rangle$ crossing the regions.

Solution to Exercise 12

```
static int count111(int[] a, int left, int right){
    int inc = 0, mid;
    if ( left == right || right-left == 1 ) return 0; // one or two cells
    if ( right - left == 2) // three cells
        if ( a[left] == 1 && a[left+1] == 1 && a[right] == 1 ) return 1;
        else return 0;

    // if we are here a[left]...a[right] has four of more cells

    mid = (left+right)/2;

    // count the number of sequences traversing the middle of the array
    for(int i = mid - 1; i<= mid  ; i++)
        if (a[i]  == 1 && a[i + 1]  == 1 && a[i + 2]  == 1) inc++;

    return inc + count111(a, left, mid) + count111(a, mid+1, right);
} // end function
```

Another Solution to Exercise 12

If we change the way we split the array a, then we get a new solution, with a different recombine step.
The elements of a sequence can be shared between the regions in two ways:

$\cdots$	1	1	1	$\cdots$

$\cdots$	1	1	1	$\cdots$

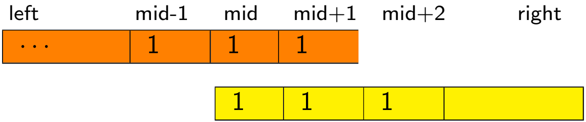
The recursive call

- `count10(a, left, mid)` counts all the sequences whose first element is between `left` and `mid-2`. Similarly
- `count10(a, mid+1, right)` counts all the sequences whose first element is between `mid+1` and `right-2`.

This implies both the recursive calls disregard the sequences crossing the region.

Another Solution to Exercise 12

We can change the recursive calls as follows:



The recursive calls have two cells of the region in common:

- the recursive call on the orange region takes into account a sequence starting in `mid-1` and ending in `mid+1`;
- the recursive call on the yellow region takes into account the sequence starting in `mid` and ending in `mid+2`

Since the region starting in `left` and ending in `right` has at least four cells, it follows that the orange and yellow region have less elements than the whole region. This proves that the method we are going to provide terminates.

Another Solution to Exercise 12

```
static int count111V2(int[] a, int left, int right){
    int  mid;

    if ( left == right || right-left == 1 ) return 0; // one or two cells

    if ( right - left == 2) // three cells
        if ( a[left] == 1 && a[left+1] == 1 && a[right] == 1 ) return 1;
        else return 0;

    // if we are here a[left]...a[right] has four of more cells

    mid = (left+right)/2;

    return count111V2(a, left, mid+1) + count111V2(a, mid, right);
}
```