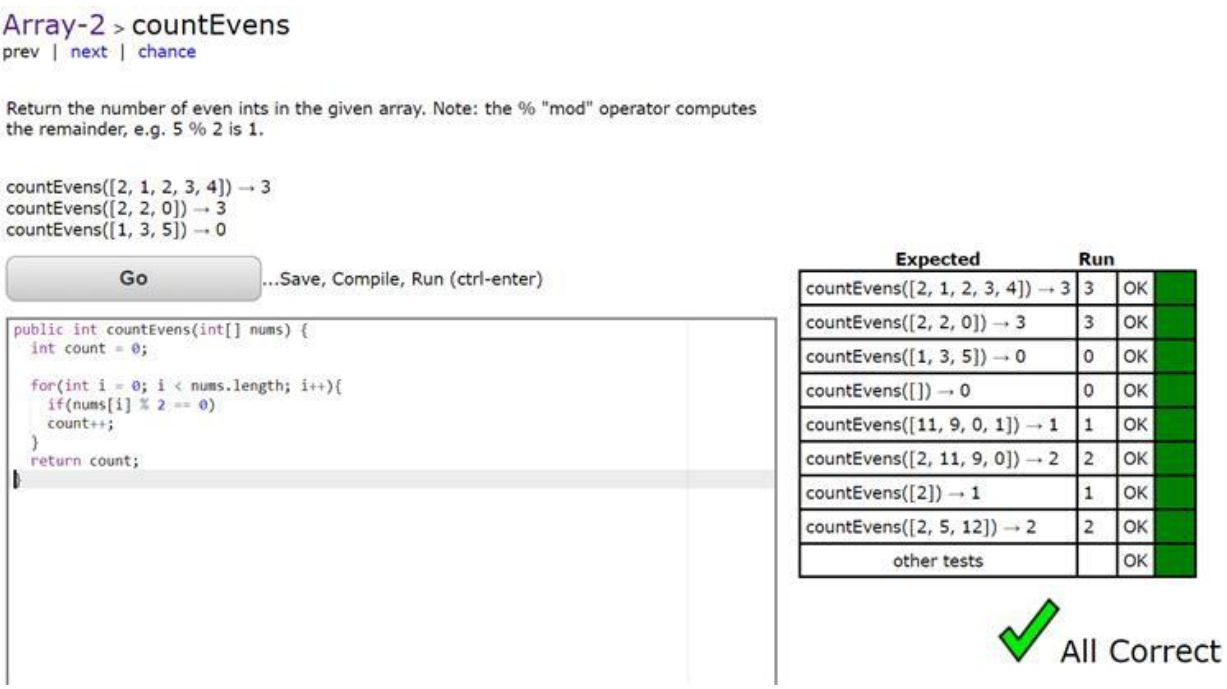
Laboratorio Nro. 2
Complejidad de algoritmos

ESTRUCTURA DE DATOS 1
Código ST0245

Resumen de ejercicios a resolver

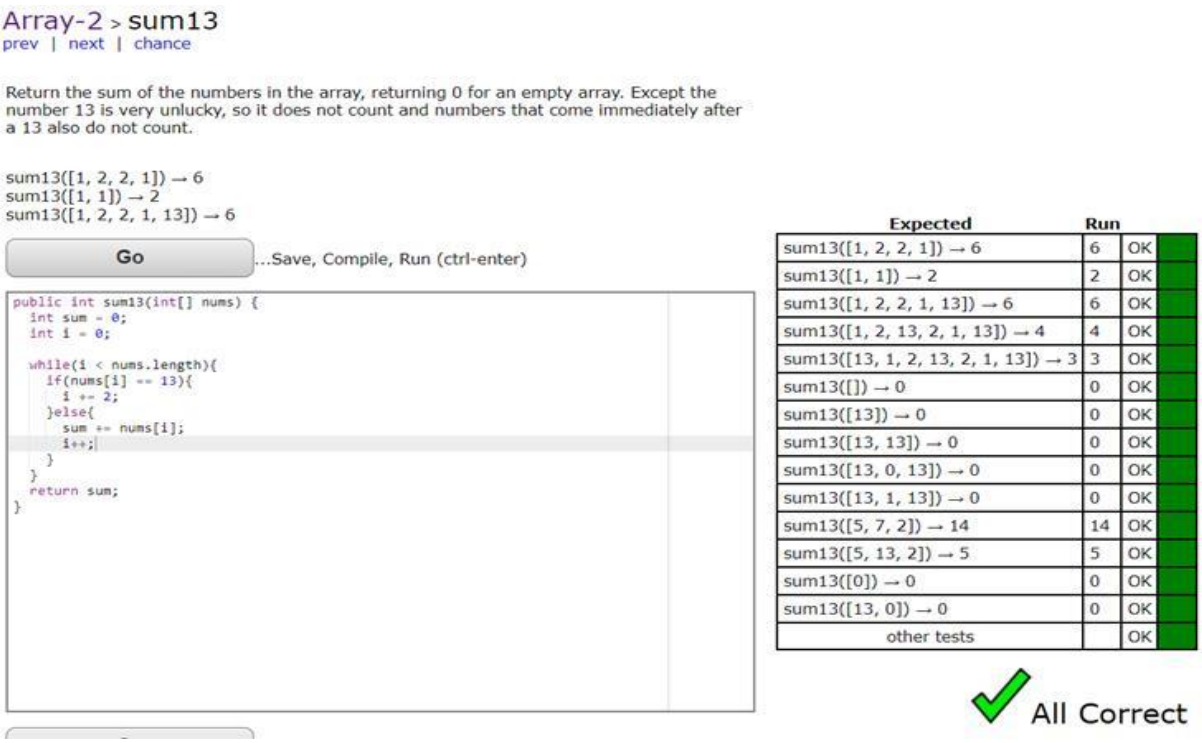
1.1 Midan los tiempos de ejecución de los algoritmos *Insertion Sort* y *Merge sort*
Estan en GitHub

2.1 Resuelvan –mínimo-- 5 ejercicios del nivel *Array 2* de la página *CodingBat*:
http://codingbat.com/java/Array-2



O(n)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas



Array-2 > sum13
prev | next | chance

Return the sum of the numbers in the array, returning 0 for an empty array. Except the number 13 is very unlucky, so it does not count and numbers that come immediately after a 13 also do not count.

sum13([1, 2, 2, 1]) → 6
sum13([1, 1]) → 2
sum13([1, 2, 2, 1, 13]) → 6

Go ...Save, Compile, Run (ctrl-enter)

```
public int sum13(int[] nums) {
  int sum = 0;
  int i = 0;

  while(i < nums.length){
    if(nums[i] == 13){
      i += 2;
    }else{
      sum += nums[i];
      i++;
    }
  }
  return sum;
}
```

	Expected	Run	
sum13([1, 2, 2, 1]) → 6	6	OK	
sum13([1, 1]) → 2	2	OK	
sum13([1, 2, 2, 1, 13]) → 6	6	OK	
sum13([1, 2, 13, 2, 1, 13]) → 4	4	OK	
sum13([13, 1, 2, 13, 2, 1, 13]) → 3	3	OK	
sum13([]) → 0	0	OK	
sum13([13]) → 0	0	OK	
sum13([13, 13]) → 0	0	OK	
sum13([13, 0, 13]) → 0	0	OK	
sum13([13, 1, 13]) → 0	0	OK	
sum13([5, 7, 2]) → 14	14	OK	
sum13([5, 13, 2]) → 5	5	OK	
sum13([0]) → 0	0	OK	
sum13([13, 0]) → 0	0	OK	
other tests		OK	

All Correct

O(Log n)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas

Array-2 > lucky13

prev | next | chance

Given an array of ints, return true if the array contains no 1's and no 3's.

lucky13([0, 2, 4]) → true
lucky13([1, 2, 3]) → false
lucky13([1, 2, 4]) → false

Go ...Save, Compile, Run (ctrl-enter)

```
public boolean lucky13(int[] nums) {
  for(int i = 0; i < nums.length; i++){
    if(nums[i] == 1 || nums[i] == 3)
    return false;
  }
  return true;
}
```

Expected	Run		
lucky13([0, 2, 4]) → true	true	OK	
lucky13([1, 2, 3]) → false	false	OK	
lucky13([1, 2, 4]) → false	false	OK	
lucky13([2, 7, 2, 8]) → true	true	OK	
lucky13([2, 7, 1, 8]) → false	false	OK	
lucky13([3, 7, 2, 8]) → false	false	OK	
lucky13([2, 7, 2, 1]) → false	false	OK	
lucky13([1, 2]) → false	false	OK	
lucky13([2, 2]) → true	true	OK	
lucky13([2]) → true	true	OK	
lucky13([3]) → false	false	OK	
lucky13([]) → true	true	OK	
other tests		OK	

All Correct

O(n)

Array-2 > fizzArray

prev | next | chance

Given a number n, create and return a new int array of length n, containing the numbers 0, 1, 2, ... n-1. The given n may be 0, in which case just return a length 0 array. You do not need a separate if-statement for the length-0 case; the for-loop should naturally execute 0 times in that case, so it just works. The syntax to make a new int array is: new int[desired_length] (See also: FizzBuzz Code)

fizzArray(4) → [0, 1, 2, 3]
fizzArray(1) → [0]
fizzArray(10) → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Go ...Save, Compile, Run (ctrl-enter)

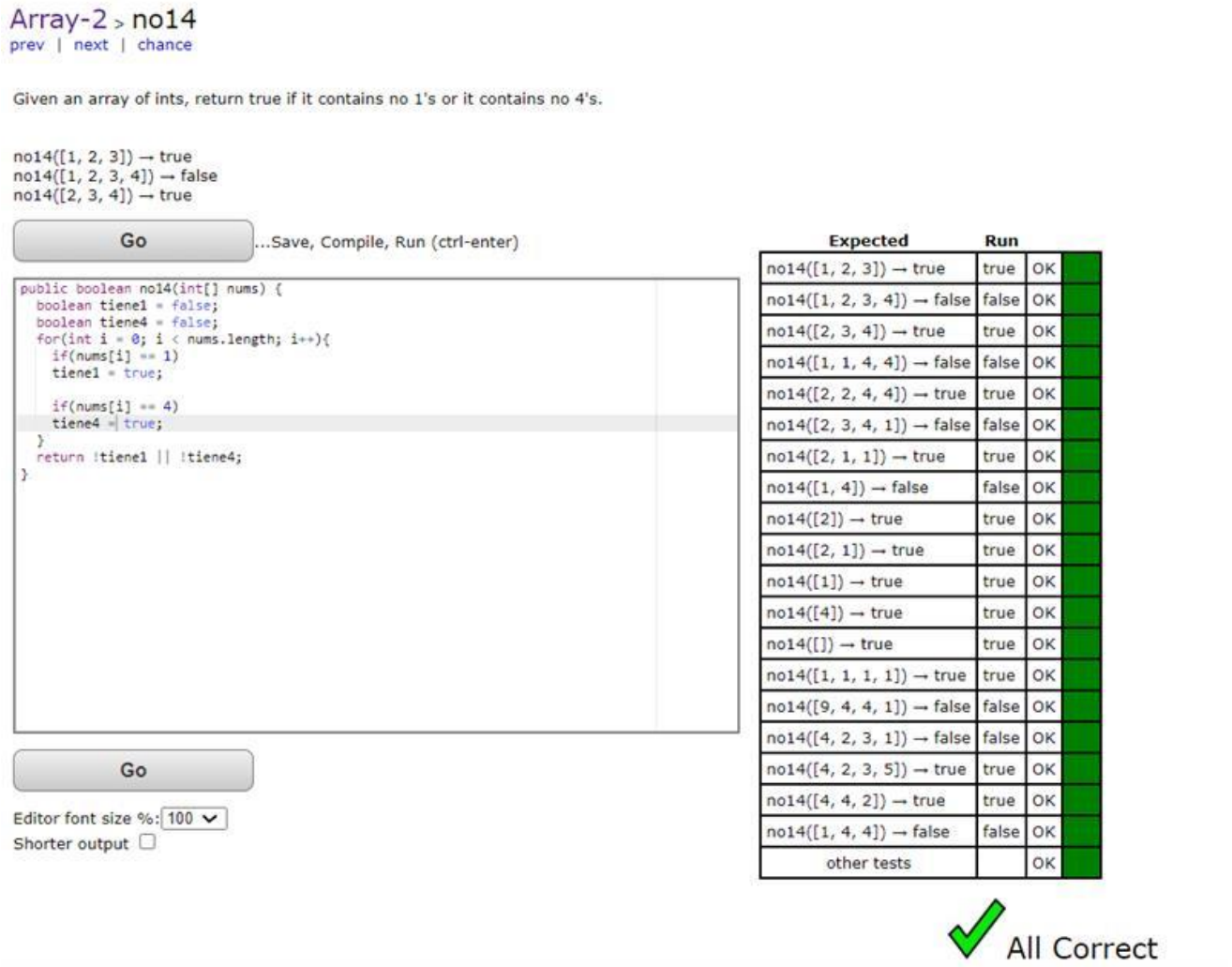
```
public int[] fizzArray(int n) {
  int[] arr = new int[n];

  for(int i = 0; i < n; i++)
  arr[i] = i;

  return arr;
}
```

Expected	Run		
fizzArray(4) → [0, 1, 2, 3]	[0, 1, 2, 3]	OK	
fizzArray(1) → [0]	[0]	OK	
fizzArray(10) → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	OK	
fizzArray(0) → []	[]	OK	
fizzArray(2) → [0, 1]	[0, 1]	OK	
fizzArray(7) → [0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	OK	
other tests		OK	

All Correct

O(n)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas



Array-2 > no14
prev | next | chance

Given an array of ints, return true if it contains no 1's or it contains no 4's.

no14([1, 2, 3]) → true
no14([1, 2, 3, 4]) → false
no14([2, 3, 4]) → true

Go ...Save, Compile, Run (ctrl-enter)

```
public boolean no14(int[] nums) {
  boolean tiene1 = false;
  boolean tiene4 = false;
  for(int i = 0; i < nums.length; i++){
    if(nums[i] == 1)
    tiene1 = true;

    if(nums[i] == 4)
    tiene4 = true;
  }
  return !tiene1 || !tiene4;
}
```

Go

Editor font size %: 100
Shorter output

Expected	Run		
no14([1, 2, 3]) → true	true	OK	
no14([1, 2, 3, 4]) → false	false	OK	
no14([2, 3, 4]) → true	true	OK	
no14([1, 1, 4, 4]) → false	false	OK	
no14([2, 2, 4, 4]) → true	true	OK	
no14([2, 3, 4, 1]) → false	false	OK	
no14([2, 1, 1]) → true	true	OK	
no14([1, 4]) → false	false	OK	
no14([2]) → true	true	OK	
no14([2, 1]) → true	true	OK	
no14([1]) → true	true	OK	
no14([4]) → true	true	OK	
no14([]) → true	true	OK	
no14([1, 1, 1, 1]) → true	true	OK	
no14([9, 4, 4, 1]) → false	false	OK	
no14([4, 2, 3, 1]) → false	false	OK	
no14([4, 2, 3, 5]) → true	true	OK	
no14([4, 4, 2]) → true	true	OK	
no14([1, 4, 4]) → false	false	OK	
other tests		OK	

All Correct

O(n)

2.2 Resuelvan --mínimo-- 5 ejercicios del nivel *Array 3* de la página *CodingBat*: **http://codingbat.com/java/Array-3**

Array-3 > maxSpan

prev | next | chance

Consider the leftmost and righmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority.)

maxSpan([1, 2, 1, 1, 3]) → 4
maxSpan([1, 4, 2, 1, 4, 1, 4]) → 6
maxSpan([1, 4, 2, 1, 4, 4, 4]) → 6

Go ...Save, Compile, Run (ctrl-enter)

```
public int maxSpan(int[] nums) {
  int max = 0;

  for(int i = 0; i < nums.length; i++){
    int j = nums.length - 1;

    while(nums[i] != nums[j])
    j--;

    int span = j - i + 1;

    if(span > max)
    max = span;
  }
  return max;
}
```

	Expected	Run	
maxSpan([1, 2, 1, 1, 3]) → 4	4	OK	
maxSpan([1, 4, 2, 1, 4, 1, 4]) → 6	6	OK	
maxSpan([1, 4, 2, 1, 4, 4, 4]) → 6	6	OK	
maxSpan([3, 3, 3]) → 3	3	OK	
maxSpan([3, 9, 3]) → 3	3	OK	
maxSpan([3, 9, 9]) → 2	2	OK	
maxSpan([3, 9]) → 1	1	OK	
maxSpan([3, 3]) → 2	2	OK	
maxSpan([]) → 0	0	OK	
maxSpan([1]) → 1	1	OK	
other tests		OK	

All Correct

O(n)

Array-3 > seriesUp

prev | next | chance

Given n>=0, create an array with the pattern {1, 1, 2, 1, 2, 3, ... 1, 2, 3 .. n} (spaces added to show the grouping). Note that the length of the array will be 1 + 2 + 3 ... + n, which is known to sum to exactly n*(n + 1)/2.

seriesUp(3) → [1, 1, 2, 1, 2, 3]
seriesUp(4) → [1, 1, 2, 1, 2, 3, 1, 2, 3, 4]
seriesUp(2) → [1, 1, 2]

Go ...Save, Compile, Run (ctrl-enter)

```
public int[] seriesUp(int n) {
  int [] arr = new int[n*(n+1)/2];

  int indice = 0;

  for(int i = 1; i <= n; i++){
    for(int j = 0; j < i; j++){
      arr[indice + j] = j + 1;
    }
    indice += i;
  }
  return arr;
}
```

	Expected	Run	
seriesUp(3) → [1, 1, 2, 1, 2, 3]	[1, 1, 2, 1, 2, 3]	OK	
seriesUp(4) → [1, 1, 2, 1, 2, 3, 1, 2, 3, 4]	[1, 1, 2, 1, 2, 3, 1, 2, 3, 4]	OK	
seriesUp(2) → [1, 1, 2]	[1, 1, 2]	OK	
seriesUp(1) → [1]	[1]	OK	
seriesUp(0) → []	[]	OK	
seriesUp(6) → [1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6]	[1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6]	OK	

All Correct

O(n^2)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas

Array-3 > fix34
prev | next | chance

Return an array that contains exactly the same numbers as the given array, but rearranged so that every 3 is immediately followed by a 4. Do not move the 3's, but every other number may move. The array contains the same number of 3's and 4's, every 3 has a number after it that is not a 3, and a 3 appears in the array before any 4.

fix34([1, 3, 1, 4]) → [1, 3, 4, 1]
fix34([1, 3, 1, 4, 4, 3, 1]) → [1, 3, 4, 1, 1, 3, 4]
fix34([3, 2, 2, 4]) → [3, 4, 2, 2]

Go ...Save, Compile, Run (ctrl-enter)

```
public int[] fix34(int[] nums) {
  for(int i = 0; i < nums.length; i++){
    if(nums[i] == 3){
      for(int j = 0; j < nums.length; j++){
        if(nums[j] == 4 && nums[j - 1] != 3){
          int temp = nums[i+1];
          nums[i+1] = nums[j];
          nums[j] = temp;
        }
      }
    }
  }
  return nums;
}
```

Go

Expected	Run	
fix34([1, 3, 1, 4]) → [1, 3, 4, 1]	[1, 3, 4, 1]	OK
fix34([1, 3, 1, 4, 4, 3, 1]) → [1, 3, 4, 1, 1, 3, 4]	[1, 3, 4, 1, 1, 3, 4]	OK
fix34([3, 2, 2, 4]) → [3, 4, 2, 2]	[3, 4, 2, 2]	OK
fix34([3, 2, 3, 2, 4, 4]) → [3, 4, 3, 4, 2, 2]	[3, 4, 3, 4, 2, 2]	OK
fix34([2, 3, 2, 3, 2, 4, 4]) → [2, 3, 4, 3, 4, 2, 2]	[2, 3, 4, 3, 4, 2, 2]	OK
fix34([5, 3, 5, 4, 5, 4, 5, 4, 3, 5, 3, 5]) → [5, 3, 4, 5, 5, 5, 5, 5, 3, 4, 3, 4]	[5, 3, 4, 5, 5, 5, 5, 5, 3, 4, 3, 4]	OK
fix34([3, 1, 4]) → [3, 4, 1]	[3, 4, 1]	OK
fix34([3, 4, 1]) → [3, 4, 1]	[3, 4, 1]	OK
fix34([1, 1, 1]) → [1, 1, 1]	[1, 1, 1]	OK
fix34([1]) → [1]	[1]	OK
fix34([]) → []	[]	OK
fix34([7, 3, 7, 7, 4]) → [7, 3, 4, 7, 7]	[7, 3, 4, 7, 7]	OK
fix34([3, 1, 4, 3, 1, 4]) → [3, 4, 1, 3, 4, 1]	[3, 4, 1, 3, 4, 1]	OK
fix34([3, 1, 1, 3, 4, 4]) → [3, 4, 1, 3, 4, 1]	[3, 4, 1, 3, 4, 1]	OK
other tests		OK

All Correct

$O(n^2)$

Array-3 > linearIn

prev | next | chance

Given two arrays of ints sorted in increasing order, **outer** and **inner**, return true if all of the numbers in inner appear in outer. The best solution makes only a single "linear" pass of both arrays, taking advantage of the fact that both arrays are already in sorted order.

linearIn([1, 2, 4, 6], [2, 4]) → true
linearIn([1, 2, 4, 6], [2, 3, 4]) → false
linearIn([1, 2, 4, 4, 6], [2, 4]) → true

Go ...Save, Compile, Run (ctrl-enter)

```
public boolean linearIn(int[] outer, int[] inner) {
  int i = 0;
  int j = 0;

  while(i < inner.length && j < outer.length){
    if(inner[i] > outer[j]){
      j++;
    }else if(inner[i] < outer[j]){
      return false;
    }else{
      i++;
    }
  }
  if(i != inner.length)
  return false;

  return true;
}
```

Expected		Run	
linearIn([1, 2, 4, 6], [2, 4]) → true	true	OK	
linearIn([1, 2, 4, 6], [2, 3, 4]) → false	false	OK	
linearIn([1, 2, 4, 4, 6], [2, 4]) → true	true	OK	
linearIn([2, 2, 4, 4, 6, 6], [2, 4]) → true	true	OK	
linearIn([2, 2, 2, 2, 2], [2, 2]) → true	true	OK	
linearIn([2, 2, 2, 2, 2], [2, 4]) → false	false	OK	
linearIn([2, 2, 2, 2, 4], [2, 4]) → true	true	OK	
linearIn([1, 2, 3], [2]) → true	true	OK	
linearIn([1, 2, 3], [-1]) → false	false	OK	
linearIn([1, 2, 3], []) → true	true	OK	
linearIn([-1, 0, 3, 3, 3, 10, 12], [-1, 0, 3, 12]) → true	true	OK	
linearIn([-1, 0, 3, 3, 3, 10, 12], [0, 3, 12, 14]) → false	false	OK	
linearIn([-1, 0, 3, 3, 3, 10, 12], [-1, 10, 11]) → false	false	OK	
other tests		OK	

All Correct

O(Log n)

Array-3 > maxMirror

prev | next | chance

We'll say that a "mirror" section in an array is a group of contiguous elements such that somewhere in the array, the same group appears in reverse order. For example, the largest mirror section in {1, 2, 3, 8, 9, 3, 2, 1} is length 3 (the {1, 2, 3} part). Return the size of the largest mirror section found in the given array.

maxMirror([1, 2, 3, 8, 9, 3, 2, 1]) → 3
maxMirror([1, 2, 1, 4]) → 3
maxMirror([7, 1, 2, 9, 7, 2, 1]) → 2

Go ...Save, Compile, Run (ctrl-enter)

```
public int maxMirror(int[] nums) {
  int max = 0;
  for(int i = 0; i < nums.length; i++){
    int count = 0;
    for(int j = nums.length - 1; j >= 0 && i + count < nums.length; j--){
      if(nums[i + count] == nums[j]){
        count++;
      }else{
        max = Math.max(max, count);
        count = 0;
      }
    }
    max = Math.max(max, count);
  }
  return max;
}
```

Go

Expected		Run	
maxMirror([1, 2, 3, 8, 9, 3, 2, 1]) → 3	3	OK	
maxMirror([1, 2, 1, 4]) → 3	3	OK	
maxMirror([7, 1, 2, 9, 7, 2, 1]) → 2	2	OK	
maxMirror([21, 22, 9, 8, 7, 6, 23, 24, 6, 7, 8, 9, 25, 7, 8, 9]) → 4	4	OK	
maxMirror([1, 2, 1, 20, 21, 1, 2, 1, 2, 23, 24, 2, 1, 2, 1, 25]) → 4	4	OK	
maxMirror([1, 2, 3, 2, 1]) → 5	5	OK	
maxMirror([1, 2, 3, 3, 8]) → 2	2	OK	
maxMirror([1, 2, 7, 8, 1, 7, 2]) → 2	2	OK	
maxMirror([1, 1, 1]) → 3	3	OK	
maxMirror([1]) → 1	1	OK	
maxMirror([]) → 0	0	OK	
maxMirror([9, 1, 1, 4, 2, 1, 1, 1]) → 3	3	OK	
maxMirror([5, 9, 9, 4, 5, 4, 9, 9, 2]) → 7	7	OK	
maxMirror([5, 9, 9, 6, 5, 4, 9, 9, 2]) → 2	2	OK	
maxMirror([5, 9, 1, 6, 5, 4, 1, 9, 5]) → 3	3	OK	
other tests		OK	

All Correct

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas

3.2 Grafiquen los tiempos que tomó, cada algoritmo, para los 20 tamaños del problema.

Insertion Sort = 0.0000305000 ms
Merge Sort = 0.0000264000 ms

3.3 Según la complejidad en tiempo, ¿es apropiado usar *insertion sort* para un videojuego con millones de elementos en una escena y demandas de tiempo real en la renderización de escenas 3D?
No porque este método es eficiente para arreglos pequeños ya que su velocidad de procesamiento es inversamente proporcional al tamaño del arreglo, entre mayor es este la velocidad se hace cada vez más lenta.

3.4 ¿Por qué aparece un logaritmo en la complejidad asintótica, para el peor de los casos, de *merge sort* o *insertion sort*?
A medida que aumenta la cantidad de datos, el número de operaciones aumenta, pero no de forma proporcional a los datos. En la vida real la **complejidad** logarítmica se da en casos en los que no es necesario recorrer todos los elementos.

3.5 Calculen la complejidad de los ejercicios en línea, numerales 2.1 y 2.2.
La respuesta está arriba junto los pantallazos.

4. Simulacro de Parcial.

4.1 Juanito implementó un algoritmo para sumar dos matrices cuadradas de dimensión *NxN*. Su algoritmo tiene complejidad $T(n)=c\times n^2$ y toma $T(n)$ segundos para procesar n datos.

¿Cuánto tiempo tardará este algoritmo para para procesar 10000 datos, si sabemos que, para n=100, $T(n)=T(100)$=1ms? Recuerda que 1sg=1000ms. Así como en los parciales de Física 1, NO olvides indicar la unidad de medida del tiempo que calcules.

100n ---- 1ms
10000n ---- Xms?

10000n x 1ms / 100n = 100ms

4.2 Dayla sabe que la complejidad asintótica de la función P(n) es $O(\sqrt{n})$. Ayúdenle a Dayla a sacar la complejidad asintótica para la función mistery(n,m).

```
void mistery(int n, int m) {
  for(int i = 0; i < m; ++i){
    boolean can = P(n);
    if(can)
      for(int i = 1; i * i <= n; i++)
        //Hacer algo en O(1)
    else
      for(int i = 1; i <= n; i++)
        //Hacer algo en O(1)
} }
```

La complejidad de mistery(n,m) es:

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas

a) O(m+n)
b) O(m×n×√n)
c) O(m+n+√n)
d) O(m×n)

4.3 Considera el siguiente algoritmo:

```
void f(int n){
  for(int i=1; i*i <= n; i++) {
    for(int j=1; j*j<=n; j++) {
      for(int k=0; k<n; k++) {
        for(int h=0; h<=n; h++) {
          System.out.println("hola");
        }
      }
    }
  }
}
```

¿Cuál es la complejidad asintótica, para el peor de los casos, del algoritmo *f*(*n*) para *n*>1?

a) $O(n^3)$
b) $O(n^2)$
c) $O(n^3 \times n)$
d) $O(n^4 \times \sqrt{n})$

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas

4.4 La *lógistica de última milla* es el proceso de entregar un pedido de comercio electrónico (por ejemplo, en Amazon o Rappi) desde que sale de la tienda hasta que llega al cliente final. La logística de última milla representa hasta un 50% de los costos logísticos. Contar el número de caminos se utiliza en logística de última milla. Además, según el portal *Geeks for Geeks*, este es un problema –muy frecuente– en entrevistas de Amazon, Microsoft y y Adobe. El problema es contar todos los posibles caminos, desde la esquina superior izquierda (0,0) hasta la esquina inferior derecha ($n-1$,$m-1$), de una matriz de $n \times m$, con la restricción de que desde cada celda sólo nos podemos mover hacia la derecha (i,$j+1$) o hacia abajo ($i+1$,j). Este problema es similar al problema del conejo, ¿cierto? Como un ejemplo, si $n=3$ y $m=2$, la salida es 3 porque hay estos 3 caminos:

- (0,0)->(0,1)->(0,2)->(1,2) •
  (0,0)->(0,1)->(1,1)->(1,2)
- (0,0)->(1,0)->(1,1)->(1,2)

Si trabajas en Java, considera este código:

```
int numberOfPaths(int m, int n) {
 int count[][] = new int[m][n];

 for (int i = 0; i < m; i++)
      count[i][0] = 1;

 for (int j = 0; j < n; j++)
      count[0][j] = 1;

 for (int i = 1; i < m; i++) {
  for (int j = 1; j < n; j++)
   count[i][j]=count[i-1][j]+count[i][j-1];
 }
 return count[m - 1][n - 1];
}
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas

Si trabajas en Python, considera este código:

```
def numberOfPaths(m, n):
 count = [[0 for y in range(n)] for x in range
     (m)]

 for i in range(m):
  count[i][0] = 1

 for j in range(n):
  count[0][j] = 1

 for i in range(1, m):
  for j in range(1, n):
   count[i][j] = count[i-1][j] + count[i][j-1]
 return count[m-1][n-1]
```

1. ¿Cuál es la complejidad asintótica, en **tiempo**, en el peor de los casos, en términos de *n* y de *m*? O(m*n)

2. ¿Cuál es la complejidad asintótica, en **memoria**, en el peor de los casos, en términos de *n* y de *m*? O(m*n)

La complejidad en memoria quiere decir cuántos enteros nuevos crea el algoritmo, fuera de los que ya existían.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas

4.5 Considera el siguiente algoritmo:

```
static int count7(int n) {
    if (n == 0) return 0;
    if (n % 10 == 7) return 1 + count7(n / 10);
    return count7(n / 10);
}
```

¿Cuál es la ecuación de recurrencia que mejor define la complejidad, para el peor caso, del algoritmo anterior? Asume que c es la suma de todas las operaciones que toman un tiempo constante en el algoritmo.

(a) $T(n)=T(n-1)+c$, que es $O(n)$

(b) $T(n)=4T(n/2)+c$, que es $O(n^2)$

(c) $T(n)=T(n-1)+T(n-2)+c$, que es $O(2^n)$

(d) $T(n)=T(n/10)+c$, que es $O(\log_{10} n)$

¿El algoritmo anterior siempre termina para todo número entero $n \in Z$?

(a) Sí
(b) No

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas