Practica 2. Divide y vencerás I
Lunes 09 de Septiembre de 2019

Integrantes:

- Garcia Gonzalez Aaron Antonio
- Moreno Ramírez Daniel
- Pacheco Castillo Isaías

Ejercicio 1.

a) Fuerza bruta

a) Pseudocódigo

```
# Ejercicio 1. Por fuerza bruta, dado un arreglo haya los índices tales que
A[i]=i
def coincidencias(A):
    auxiliar = []
    for i in range(len(A)):
        if i == A[i]:
            auxiliar.append(i)
    return auxiliar
```

b) Complejidad
Se recorre una única vez el arreglo, para comparar el índice “i” con Arreglo[i], por lo que la complejidad del problema es el tamaño o número de elementos del arreglo, entonces, complejidad = O(n)

c) Entrada(s)
Un arreglo de enteros, el cual esta ordenado ascendentemente, en este ejemplo en particular, el arreglo que se introduce es [0,1,2,4,5,6,7,8,9,12]

d) Salida(s)
Un arreglo de enteros con las posiciones del arreglo original que cumplieron con las restricciones del programa, en este ejemplo en particular nos retorna [0,1,2]

e) Capturas de pantalla

```
MacBook-Pro-de-Aaron:p2 aarongarcia$ python index.py
('Arreglo: ', [0, 1, 2, 4, 5, 6, 7, 8, 9, 12])
('Coincidencias no eficiente', [0, 1, 2])
```

b) Mejorado

Pseudocódigo

```
indiceB( A )
        tam→A.length
        centro→0
        inferior→A[0]
        superior→A[tam-1]

        while inferior <= superior hacer
                centro→(inferior+superior) / 2
                si A[centro] == centro
                        return centro
                fin si
                si A[centro]>centro
                        superior→centro-1
                fin si
                otro caso
                        inferior→centro+1
                fin si
        fin del while
return -1;
```

Complejidad

Tamaño
n
n/2
n/4
n/2^j

Divide los problemas hasta llegar a 1. Para esto tenemos que 2^j = n.
j = log2 (n). **O(** $log2\ (n)$ **).** Porque va partiendo el arreglo a la mitad cada vez que no encuentra el índice correspondiente.

```java
public int indiceB(int[] arreglo){

    int n = arreglo.length;//tamaño del arreglo
    int centro,inf=0,sup=n-1;

    while(inf<=sup){
        centro=(sup+inf)/2;//Obtenemos el centro
        if(arreglo[centro]==centro) //Si esta en la mitad
            return centro;//retornamos el centro
        else if(centro < arreglo [centro] ){//si el indice es menor entonces esta a la izquierda
            sup=centro-1;//El tope seria uno antes del centro
        }
        else {//si es mayor el indice nos movemos a la derecha
            inf=centro+1;//El inicio del arreglo cambia a uno despues del centro
        }
    }
    return -1;

}
```

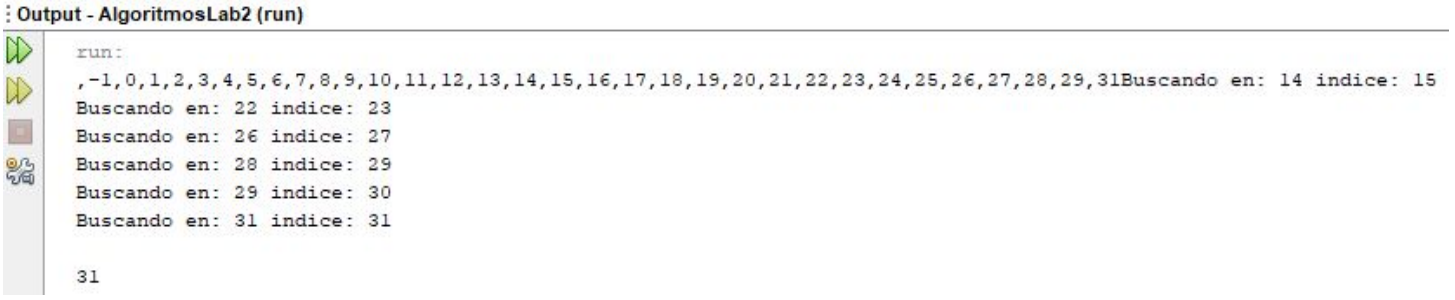
El problema nos pedía que el algoritmo tuviera una complejidad de log2(n). Además este algoritmo no es recursivo por lo que puede tardar un poco menos.

Entradas y salidas.
El arreglo que ocupamos fue:
[-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,31]
(el único que cumple con lo pedido es el 31).

Y la salida fue 31.

```
,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,31
31
```

Funcionamiento:



Ejercicio 2.

a) **Fuerza bruta**

Pseudocódigo:

```
Algoritmo( arr[ n ] , valor )
 sum ← 0

for i from 0 to n
 sum+= arr[n] * Math.pow( valor, i )

return sum;
```

Implementación:

```
public int evaluarPolinomio( int [] arr, int valor ){
      int suma=0;
      for( int i=0; i<arr.length; i++ ){
         suma+= ( arr[i]* Math.pow(valor, i ));
      }
      return suma;
   }
```

Complejidad: O(n^2) ya que se recorre el arreglo en su totalidad y dentro de la función se realizan n multiplicaciones del mismo número para devolver la potencia.

b) Menor complejidad

Pseudocódigo:

```
Algoritmo( arr[n] , valor )
aux ← valor
sum ← arr[0]

for i from 1 to n {
 sum += arr[i] + aux
 aux *= valor
}
return suma
```

Implementación:

```
public int evaluarPolinomiom( int [] arr, int valor ){
      int aux = valor;
      int suma = arr[0];

      for( int i=1; i<arr.length; i++){
         suma += arr[i] * aux;
         aux*=valor;
      }
       return suma;
```

}

Complejidad: **O(n)** ya que con una sola recorrida al arreglo se realiza la operación de evaluación del polinomio.

Entradas / Salidas:

```
Entrada: ([1, 2, 3, 7, 9] , 3 )
Salida: 952

Entrada: ([3, 4, 5, 7, 8, 10, 12, 45] , 1 )
Salida: 94

Entrada: ([1, 6, 8, 9, 11, 14, 15] , 4 )
Salida: 79321
```

Ejercicio 3. Fuerza bruta

a) pseudocódigo

```
#Ejercicio 3. Fuerza bruta, haya las inversiones dados los índices i,j donde
i < j, y también ocurre que A[i]>A[j]
def listarInversiones(A):
    auxiliar = {}

    for i in range(len(A)):
        for j in range(len(A)):
            if (i<j and A[i]>A[j]):
                auxiliar[A[i]] = A[j]
    return auxiliar
```

b) Complejidad
Dado que para recorrer el array y a la vez comparar todos con todos, se hace uso de dos ciclos for, uno anidado dentro del otro, por lo que la complejidad es $O(n^2)$

c) Entrada(s)
Recibe un arreglo de tamaño N, el cual no está ordenado, en este ejemplo en particular, el arreglo que se introduce es [2,4,1,3,5]

d) Salida(s)
Las inversiones A[i], A[j] que cumplieron con el programa, que en este diseño particular, se retornan diccionarios que almacenan dichos índices, y es este ejemplo en particular el diccionario de salida es {2: 1, 4:3}, que la interpretación a este es que 2 es mayor que 1, y 4 es mayor que 3, y en el arreglo al recorrerlo se encuentran en ese orden

e) Capturas de pantalla

```
MacBook-Pro-de-Aaron:p2 aarongarcia$ python index.py
('Arreglo: ', [2, 4, 1, 3, 5])
('Inversiones: ', {2: 1, 4: 3})
```