Universidad de Granada

Algorítmica

Memoria de Prácticas

# Práctica II: Divide y Vencerás

*Autores:*
Antonio Gámiz Delgado

17 de Marzo

Universidad de Granada

# 1 Problema y Objetivo

Dado un vector de enteros, de tamaño n, ordenado y sin elementos repetidos. Queremos encontrar, si lo hay, una posición $i$ del vector tal que su valor sea igual a ella misma, es decir, que cumpla que $v[i] = i$. Primero vamos a abordar el problema a lo bruto, y luego aplicaremos la técnica de Divide y Vencerás para ver que este último es más eficiente. Por último adaptaremos el algoritmo para vectores con elementos repetidos.

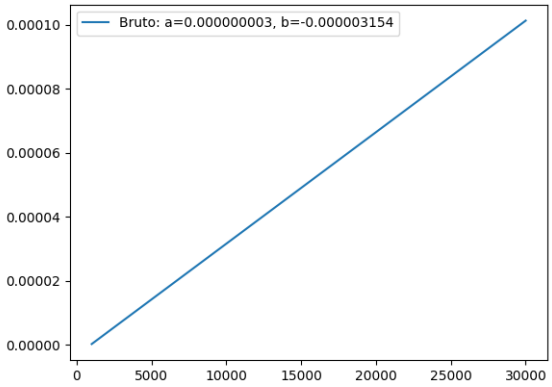
# 2 Elementos distintos

## 2.1 A lo bruto

La primera opción que abordamos es la primera que se nos ocurre: recorrer el vector $v$ desde 0 hasta $n-1$ y parar cuando encontremos el primer elemento que cumple la condición. Por lo que nos queda el siguiente código:

```
int algoritmo_bruto(vector<int> v)
{
  for(int i=0; i<v.size(); i++){
    if(v[i]==i) return i;
  }
  return -1;
}
```

Evidentemente el análisis de la eficiencia de este algoritmo nos da un orden lineal, es decir, $T(n) = n$. De forma empírica comprobamos que efectivamente tenemos:



## 2.2 Divide y vencerás(sin repeticiones)

Como el vector con el que estamos trabajando, $v$, está ordenado de menor a mayor (orden creciente), parece razonable pensar en un algoritmo parecido a la búsqueda binaria, fijándonos en el elemento que se encuentra en el medio del vector, $m = \frac{n}{2}$. Si este número coincide con el valor de $v[m]$, entonces hemos terminado.

En caso contrario, hay dos posibilidades: $v[m] > m$ o $v[m] < m$. Estudiamos el primer caso y el otro es análogo. Por lo que si tenemos que $v[m] > m$, al estar los elementos ordenados y sin repetición, sabemos que para todos los índices $i$ mayores que que $m$, los valores en esas posiciones, $v[i]$, serán siempre mayores que $v[m]$, es decir, $v[i] > i \quad \forall i > m$. Por tanto, sabemos que a la derecha de $m$ no se puede dar la situación que buscamos, luego nos la "saltamos" en el algoritmo, es decir, reducimos la búsqueda a la otra parte del vector, a la izquierda de $m$. El otro caso se razona de forma análoga.

Por lo que nos queda el siguiente algoritmo recursivo:

```
int divide_nr(vector<int>& v, int n1, int n2)
{
  if( n1==n2 )
    if( v[n1]==n1 ) return n1;
  else return -1;
  else {
    int middle=(n1+n2)/2;
    if( v[middle]==middle ) return middle;
    else if ( v[middle] > middle ) return divide_nr(v, n1, middle-1);
    else return divide_nr(v, middle+1, n2);
  }
}
```

El tiempo de ejecución del algoritmo viene dado por la expresión $T(n) = T(\frac{n}{2}) + a$. Con el cambio de variable $t_k = T(2k)$, tenemos que $t_k - t_{k-1} - a = 0$. Es una ecuación en recurrencia no homogénea con ecuación característica $(x-1)^2 = 0$. Resolviéndola obtenemos que $t_k = c_1 k + c_2$ y deshaciendo el cambio de variable anterior $(n = 2k)$ nos queda que $T(n) = c_1 \log n + c_2 \in O(\log n)$

De forma empírica comprobamos que efectivamente tenemos:

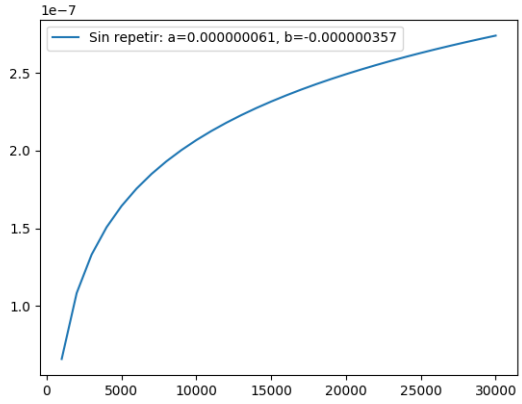



Figure 1: Eficiencia empirica

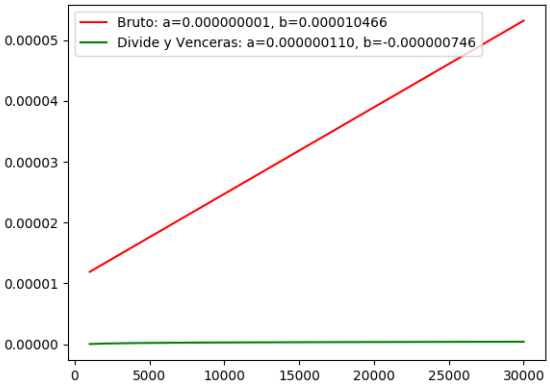



Figure 2: Empiciencia empírica de Fuerza Bruta y Divide y vencerás

### 2.3 Divide y vencerás (con repeticiones)

En este caso el algoritmo anterior no funcionará en algunos casos ya que el razonamiento usado anteriormente no es válido (necesitamos la desigualdad estricta, y en este caso no contamos con ella). Por lo que en este caso no podemos descartar de forma segura una de las partes del vector, pero lo que sí podemos asegurar en este caso es que, si $v[m] > m$, entonces para ningún índice $j$ situado entre los índices $m$ y $v[m]$ se cumple $v[j] = j$. Esto es así porque para cualquier $j$ en ese rango se tiene que $v[j] \geq v[m] > j$.

Por tanto nos centramos en dos rangos, $(1, m)$ y $(v[m], n)$. Ahora, el razonamiento es análogo al anterior pero con estos nuevos rangos. Quedando el siguiente algoritmo recursivo un poco más complejo:

```
int divide_r(vector<int>& v, int n1, int n2){
  if( n1==n2 ){
    if( v[n1]==n1 )
      return n1;
    else
      return -1;
  }else{
    int middle=(n1+n2)/2;
    if( v[middle]==middle )
      return middle;
    else if( v[middle] > middle ){

      int left=divide_r(v, n1, middle-1);
      int right;

      if( v[middle] <= n2 )
        right=divide_r(v, v[middle], n2);
      else right=0;

      if( left!=0 ) return left;
      if( right!=0 ) return right;

      return -1;
    }else{
      int right=divide_r(v, middle+1, n2);
      int left;

      if( n1<=v[middle] ) left=divide_r(v, n1, v[middle]);
      else left=0;

      if( right!=0 ) return right;
      if( left!=0 ) return left;

      return -1;
    }
  }
}
```

# 3 Capturas

Adjunto algunas capturas para ver el correcto funcionamiento del algoritmo:

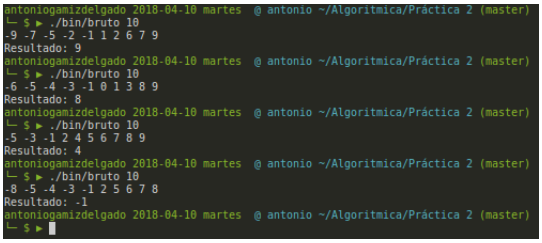



Figure 3: Algoritmo Bruto

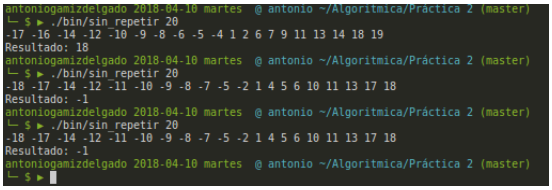



Figure 4: Divide y vencerás (sin repetición)

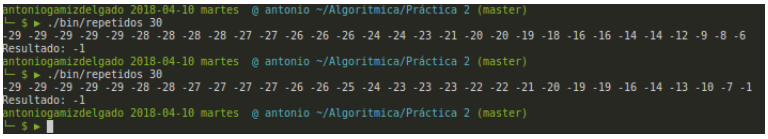



Figure 5: Divide y vencerás (con repetición)