ALGORITMOS DIVIDE Y VENCERÁS

GRUPO: HIBISCO
Integrantes:

- Noura Lachhab Bouhmadi
- Quintín Mesa Romero

ÍNDICE

EJERCICIO I

ALGORITMO DE FUERZA BRUTA

Va **comparando** cada elemento del vector con su posición. Si el elemento en la posición i, v[i], coincide con i, se fuerza la salida del bucle y se devuelve dicho elemento.

```
int buscaIgualIndice (int v[], int n)
{
    bool continua = true;
    int index = -1;

    for (int i = 0; i < n && continua; i++){
      if (v[i] == i){
          index = i;
          continua = false;
        }
    }

    return index;
}
```

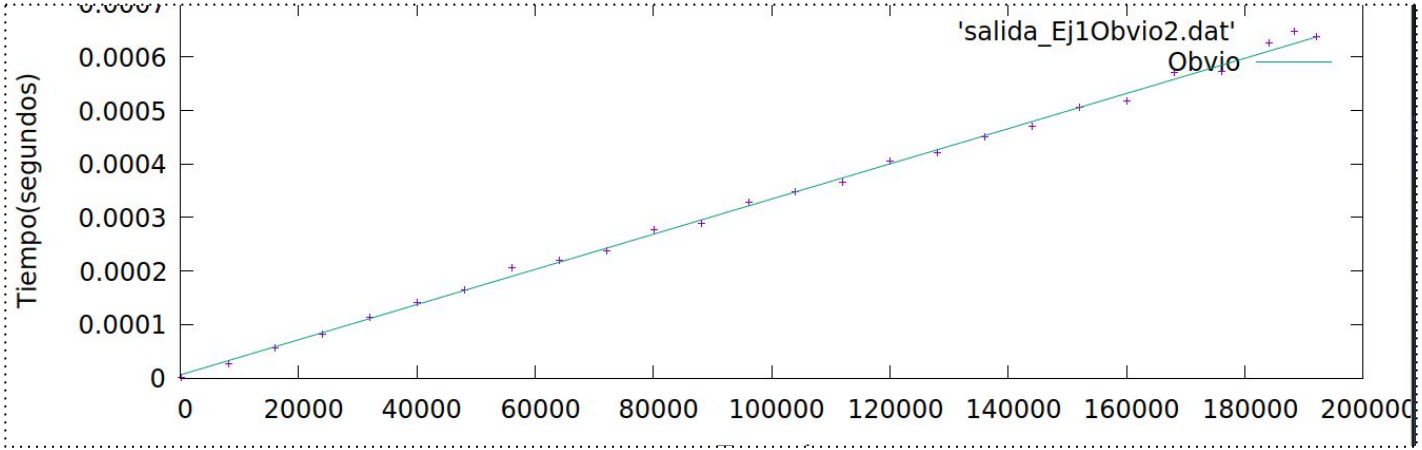
ALGORITMO DE FUERZA BRUTA

ANÁLISIS EMPÍRICO

FUERZA BRUTA	
TAMAÑO	TIEMPO DE EJECUCIÓN EN SEGUNDOS
100	5,33E-04
8096	2.77505e-05
16092	5.56081e-05
24088	8.21521e-05
32084	0.000113871
40080	0.000140849
48076	0.000165736
56072	0.000206813
64068	0.000220565
72064	0.000237331
80060	0.000276449
88056	0.000289482
96052	0.000329335
104048	0.000347544
112044	0.000366818
120040	0.000405891
128036	0.000421047
136032	0.000451158
144028	0.000470098
152024	0.000504965
160020	0.000518047
168016	0.00057008
176012	0.000572705
184008	0.000625479
192004	0.000638166

ALGORITMO DE FUERZA BRUTA

ANÁLISIS HÍBRIDO



$$f(x) = 5.83677 \cdot 10^{-6} + 3.28678 \cdot 10^{-9} \cdot x$$

ALGORITMO DIVIDE Y VENCERÁS
Versión en la que se pueden repetir elementos

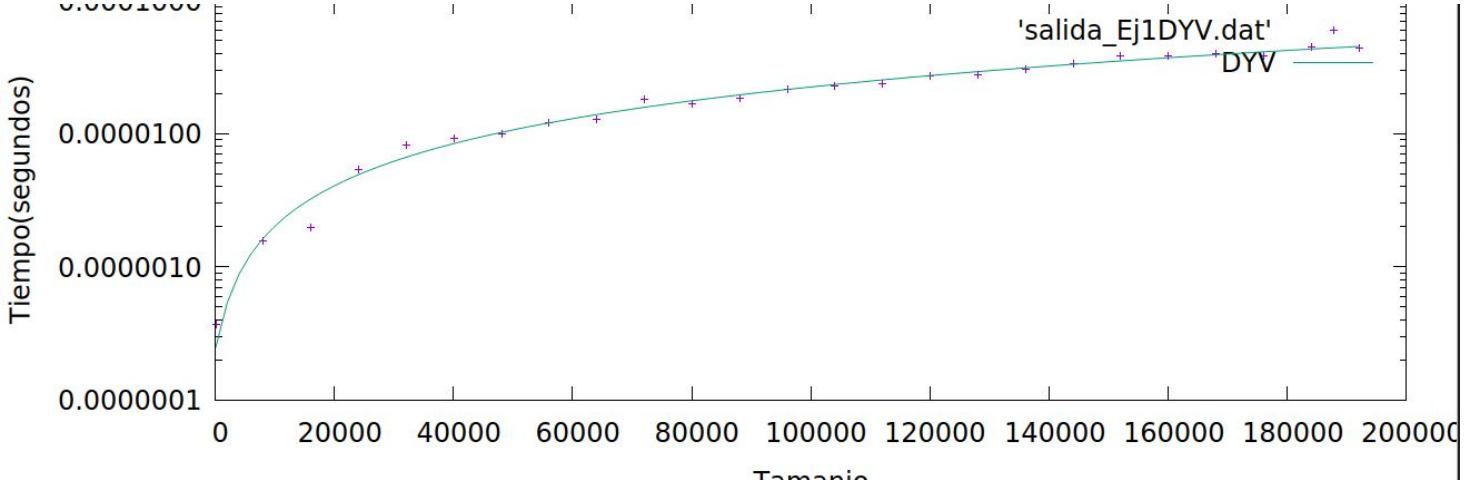
Localiza el elemento de la mitad del vector y si coincide con la posición mitad, lo devuelve. En caso contrario, si dicho elemento es mayor que la posición mitad, se descarta la mitad a la derecha de dicho elemento. Si es menor, se descarte la izquierda. Así sucesivamente hasta encontrar un elemento que cumpla lo requerido.

```
int buscaIgualIndiceDYV (int v[], int n) {  // Versión en la que no puede hab
                                            // elementos repetidos
    int izda = 0;
    int dcha = n-1;
    int half = 0;


    while (izda <= dcha){
      half = (izda+dcha)/2;
      if (v[half] > half) dcha = half-1;
      else if (v[half] < half) izda = half+1;
      else return half;
    }

    return -1;
}
```

ALGORITMO DIVIDE Y VENCERÁS
ANÁLISIS EMPÍRICO

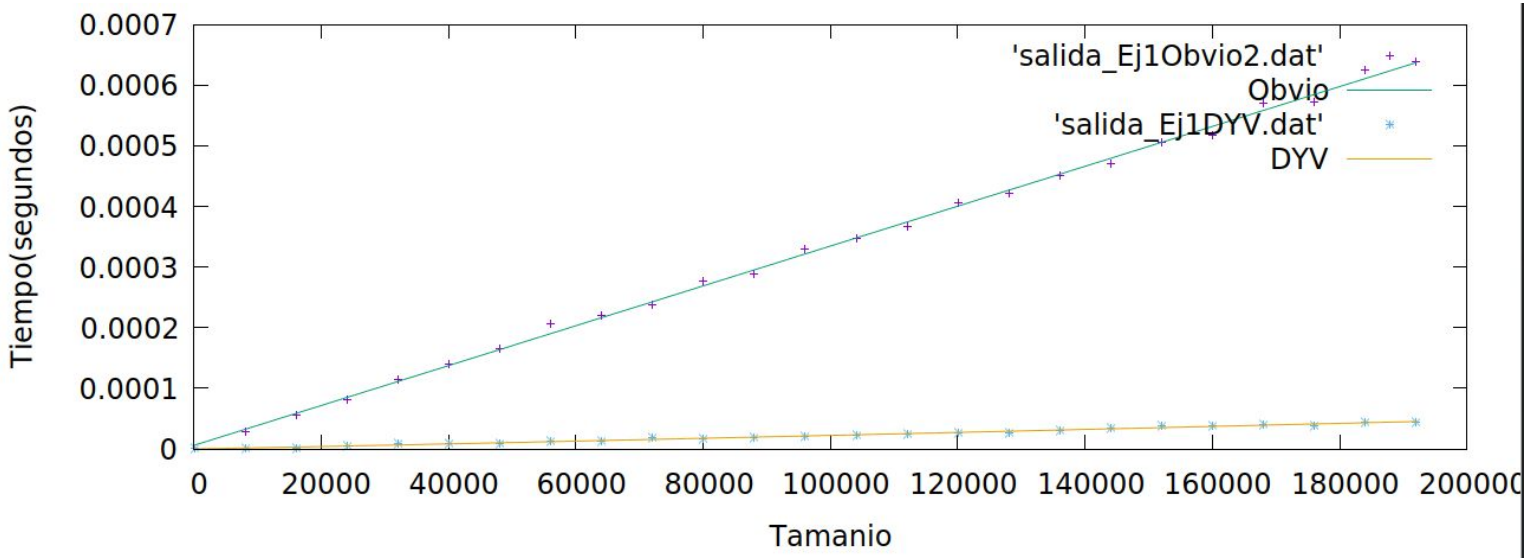
DIVIDE Y VENCERÁS (SIN REPETICIONES)	
TAMAÑO	TIEMPO DE EJECUCIÓN EN SEGUNDOS
100	3.69467e-07
8096	1.55907e-06
16092	1.9632e-06
24088	5.34987e-06
32084	8.1754e-06
40080	9.29387e-06
48076	1.00005e-05
56072	1.21519e-05
64068	1.27416e-05
72064	1.82634e-05
80060	1.6684e-05
88056	1.86288e-05
96052	2.14082e-05
104048	2.30803e-05
112044	2.38357e-05
120040	2.73881e-05
128036	2.75237e-05
136032	3.06091e-05
144028	3.39657e-05
152024	3.82245e-05
160020	3.88711e-05
168016	3.99393e-05
176012	3.81673e-05
184008	4.51063e-05
192004	4.44051e-05

ALGORITMO DIVIDE Y VENCERÁS
ANÁLISIS HÍBRIDO



$$f(x) = 4.4451 \cdot 10\text{-}11 \cdot x \cdot \log(x) + 2.37879 \cdot 10\text{-}7$$

FUERZA BRUTA VS DIVIDE Y VENCERÁS (sin rep)



¿Y SI LOS ELEMENTOS DEL VECTOR PUDIERAN REPETIRSE?

DIVIDE Y VENCERÁS (con rep)

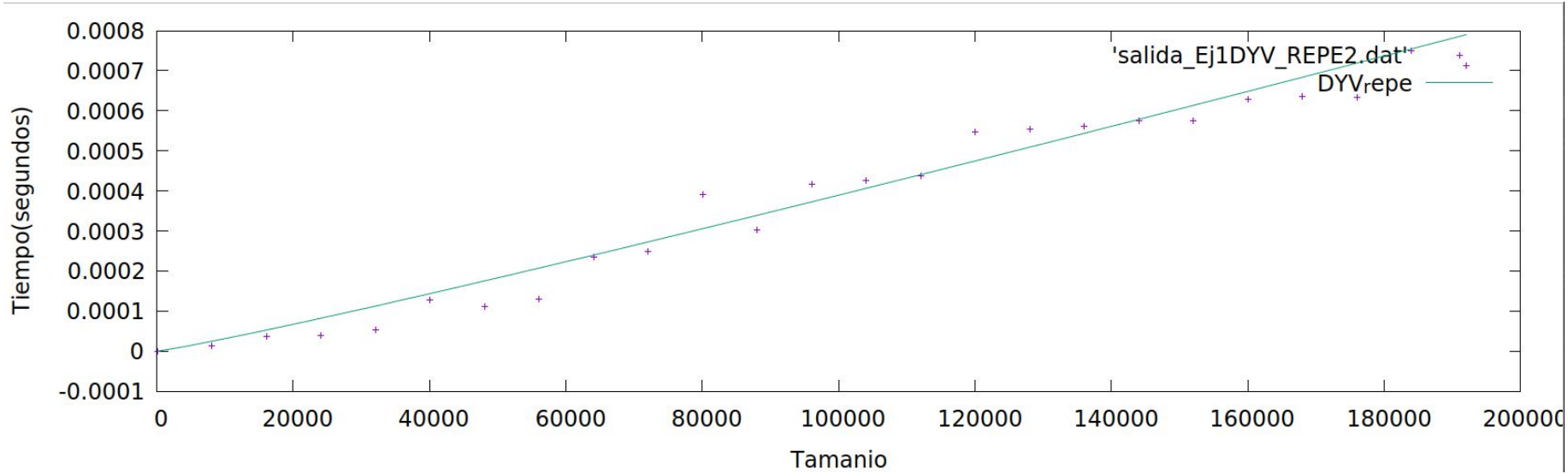
Se comprueba si el vector es de una casilla y si coincide con la posición 0, si no se devuelve -1, por convenio. En caso contrario se comprueba si los extremos verifican la condición del ejercicio y, en caso afirmativo se devuelve uno de los elementos extremos (se da preferencia al izquierdo por convenio en caso de verificar ambos la condición). Si no coinciden se elmininan los extremos antes comprobados, se calcula la mitad y si el elemento mitad cumple la condición se devuelve, en caso contrario, se divide el vector en dos mitades y, recursivamente se analizan los dos hasta encontrar un elemento que verifique la condición del ejercicio.

```
int buscaIgualIndiceDYV_REP (int v[], int l, int r)
{
  if (l <= r){

     if (l - r == 0){
         if (v[l] == l) return l;
         else return -1;
     }
     else if (v[l] == l) return l;
     else if (v[r] == r) return r;
     else{
          int half = (l+r)/2;
          if (v[half] == half) return half;
          else{
            l++;
            r--;
            int i= buscaIgualIndiceDYV_REP(v,l,half-1);
            if(i < 0){
                return buscaIgualIndiceDYV_REP(v,half+1,r);
            }else
              return i;
          }
     }
  }

  else return -1;
}
```

DIVIDE Y VENCERÁS (con rep)

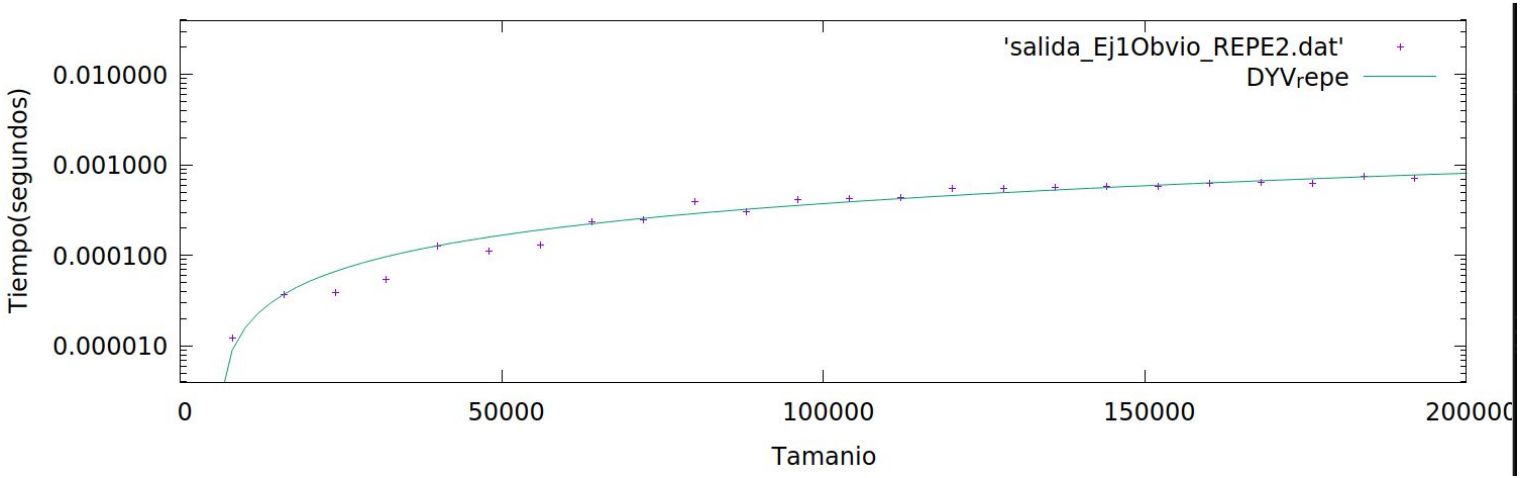
DIVIDE Y VENCERÁS CON REPETICIONES)	
TAMAÑO	TIEMPO DE EJECUCIÓN EN SEGUNDOS
100	4.14067e-07
8096	1.23172e-05
16092	3.706977e-05
24088	3.912655e-05
32084	5.3868e-05
40080	0.000128044
48076	0.000110946
56072	0.00012946
64068	0.000235981
72064	0.000248381
80060	0.000392104
88056	0.00030201
96052	0.000415648
104048	0.00042642
112044	0.000438016
120040	0.000546999
128036	0.000555258
136032	0.00056006
144028	0.000574978
152024	0.000574993
160020	0.000629421
168016	0.000634981
176012	0.00063243
184008	0.000750756
192004	0.00071303

DIVIDE Y VENCERÁS (con rep)



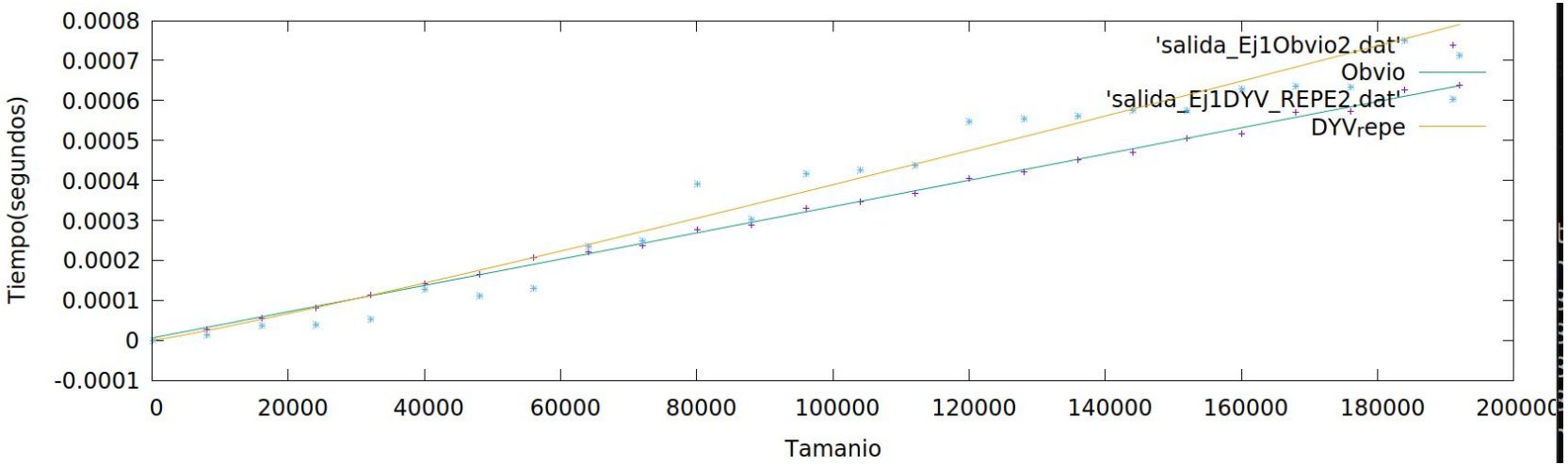
f(x) = 7.78941 · 10-10 · x · log(x) - 1.56589 · 10-7

DIVIDE Y VENCERÁS (con rep)

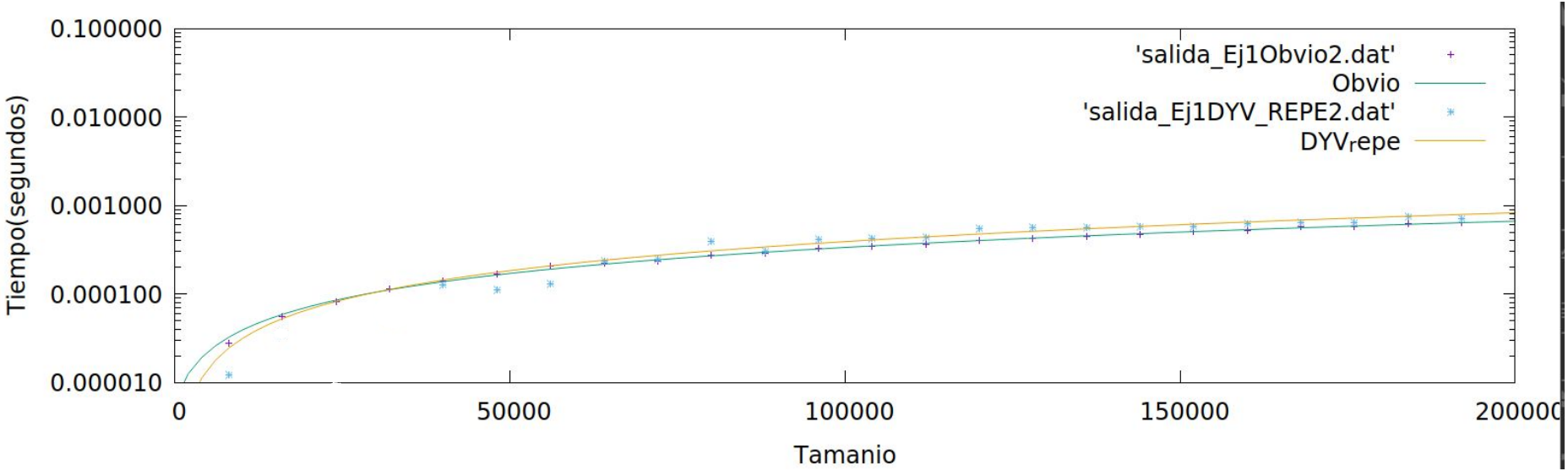


f(x) = 7.78941 · 10-10 · x · log(x) - 1.56589 · 10-7

FUERZA BRUTA VS DIVIDE Y VENCERÁS (con rep)



FUERZA BRUTA VS DIVIDE Y VENCERÁS (con rep)



CONCLUSIONES EJERCICIO 1

- Para el caso vector ordenado crecientemente con elementos distintos el algoritmo divide y vencerás es el más eficiente a la hora de resolver el problema, frente al de fuerza bruta.
- Para el caso vector ordenado crecientemente con elementos que admiten repetición, el algoritmo divide y vencerás del caso anterior es totalmente inválido.
- Para el caso vector ordenado crecientemente con elementos que admiten repetición, el algoritmo divide y vencerás es menos eficiente que el de fuerza bruta. Por lo tanto el preferible usar este último frente al divide y vencerás.

EJERCICIO 2

ALGORITMO DE FUERZA BRUTA

Este método crea un vector de tamaño K*N e introduce en él los elementos que se encuentran en la primera fila de la matriz. Luego con el uso de los 2 bucle for se van insertando los demás elementos de la matriz en el vector y a la vez se comparan los elementos que ya estaban insertados en el vector resultado.

```
int * mezcla_FB(int numVectores, int numElementos, int ** m){
  int * fusion = nullptr;
  fusion = new int [numVectores * numElementos];

  bool found;
  int k;
  for(int i = 0; i < numElementos; i++){
    fusion[i] = m[0][i];
  }
  for(int i = 1; i < numVectores; i++){ // Iteracion por vector
    for(int j = 0; j < numElementos; j++){ // Iteracion por cada elemento del nuevo
      found = false;
      k = 0;
      while(!found && k < numElementos * i + j){
        if(fusion[k] > m[i][j])
          found = true;
        else
          k++;
      }
      if(found){
        for(int p = numElementos*i+j-1; p >= k; p--){ // Traslacion (uno a la derec
          fusion[p+1] = fusion[p];
        }
        fusion[k] = m[i][j];
      }
      if(!found){
        fusion[numElementos * i + j ] = m[i][j];
      }
    }
  }

  return fusion;
}
```

ALGORITMO DE FUERZA BRUTA

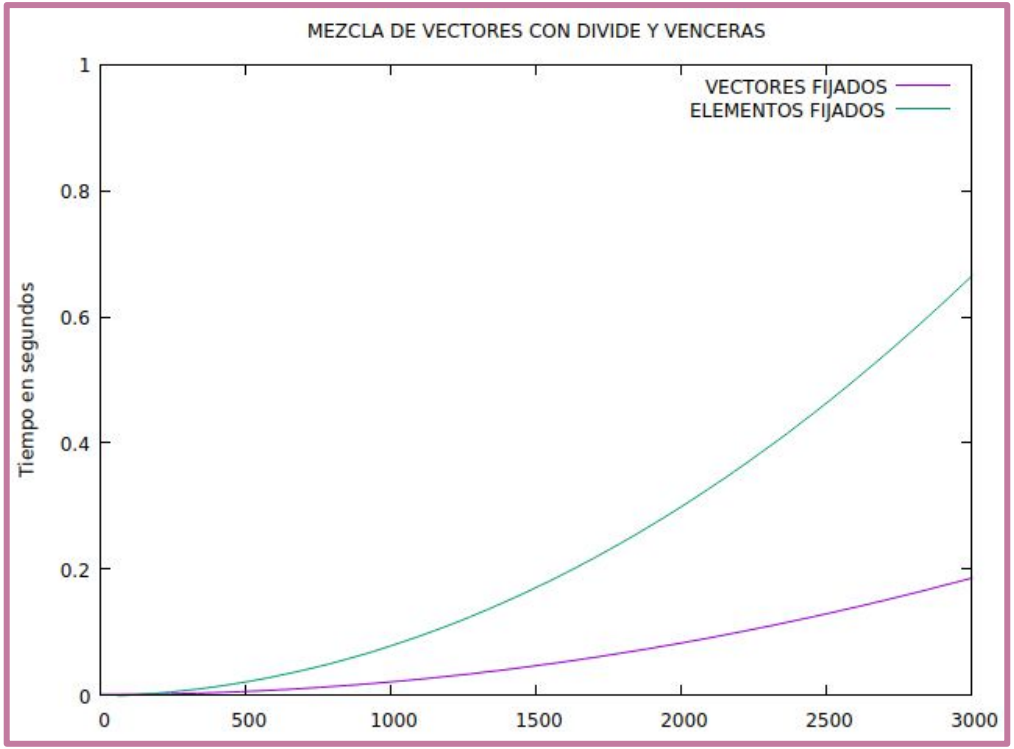
ANÁLISIS EMPÍRICO

NUMERO DE ELEMENTOS (VECTORES N=5)	TIEMPOS DE EJECUCIÓN EN SEGUNDOS
100	0.001172
496	0.006097
892	0.015509
1288	0.032932
1684	0.053301
2080	0.080623
2476	0.113845
2872	0.152556
3268	0.197894
3664	0.249331
4060	0.30896
4456	0.367212
4852	0.438958
5248	0.518436
5644	0.609944
6040	0.674232
6436	0.762268
6832	0.868134
7228	0.964117
7624	1.07277
8020	1.19122
8416	1.30782
8812	1.43861
9208	1.62673
9604	1.69641

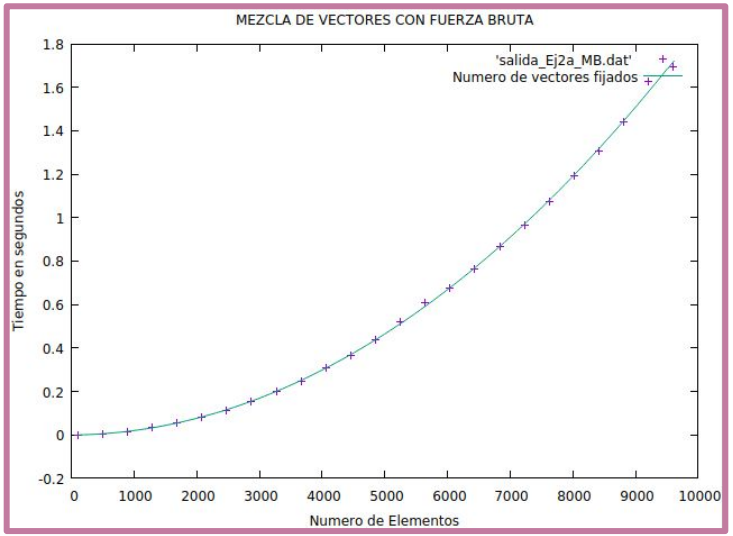
NUMERO DE VECTORES (ELEMENTOS N=8)	TIEMPOS DE EJECUCIÓN EN SEGUNDOS
100	0.00299
496	0.016775
892	0.040056
1288	0.08454
1684	0.151455
2080	0.206555
2476	0.292292
2872	0.394914
3268	0.510728
3664	0.641054
4060	0.785313
4456	0.94878
4852	1.14668
5248	1.3385
5644	1.53266
6040	1.76466
6436	2.00264
6832	2.26571
7228	2.53527
7624	2.82307
8020	3.12794
8416	3.45626
8812	3.84793
9208	4.12697
9604	4.47861

ALGORITMO DE FUERZA BRUTA

ANÁLISIS EMPÍRICO (gráfica comparativa)



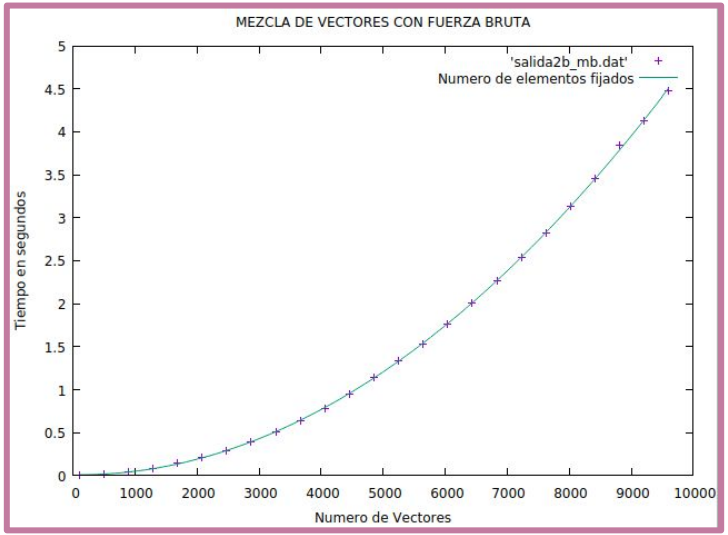
ALGORITMO DE FUERZA BRUTA
ANÁLISIS HÍBRIDO



$$f(x) = 1.76692 \cdot 10^{-8} \cdot x^3 - 2.94532 \cdot 10^{-6} \cdot x^2 - 0.000842 \cdot x + 7.06951 \cdot 10^{-14}$$

ALGORITMO DE FUERZA BRUTA

ANÁLISIS HÍBRIDO



$$f(x) = -8.44243 \cdot 10^{-15} x^3 + 4.9798 \cdot 10^{-8} x^2 - 9.50898 \cdot 10^{-8} x - 0.0105839$$

ALGORITMO DE DIVIDE Y VENCERÁS

```
void mezcla_DYV_recursivo(int vec_inicial, int vec_final, int numElementos, int **
* &mezcla){

  //PASO1: COMPROBAMOS QUE SIEMPRE VEC_FINAL > VEC_INICIAL
  if(vec_inicial < vec_final){
    //PASO2: OBTENEMOS EL NUMERO DE VECTORES QUE TENEMOS, ASI COMO EL VECTOR MEDIO
    int numVectores = vec_final-vec_inicial+1;
    int vec_medio = (vec_final+vec_inicial)/2;
    //OBTENEMOS DIMENSION PARA VECTORES AUXILIARES DE MEZCLA
    int dim2 = numVectores/2;
    int dim1 = numVectores - dim2;
    //PASO3: DECLARAMOS VECTORES AUXILIARES DONDE GUARDAREMOS RESULTADOS PREVIOS
    int * mezcla1 = nullptr; int * mezcla2 = nullptr;
    mezcla1 = new int [numElementos*(dim1)];
    mezcla2 = new int [numElementos*(dim2)];

    //LLAMADA RECURSIVA A FUNCION: CALCULA LA MEZCLA PARA K/2 VECTORES (DESDE 0 HAS
    mezcla_DYV_recursivo(vec_inicial,vec_medio,numElementos,m,mezcla1);
    //LLAMADA RECURSIVA A FUNCION: CALCULA LA MEZCLA PARA K/2 VECTORES (DESDE VEC_M
    mezcla_DYV_recursivo(vec_medio+1,vec_final,numElementos,m,mezcla2);
    //PASO 4: MEZCLAMOS LOS VECTORES OBTENIDOS ANTERIORMENTE
    mezclaVectores(mezcla1,mezcla2,dim1*numElementos,dim2*numElementos,mezcla);
    //PASO 5: LIBERAMOS MEMORIA

    delete [] mezcla1;
    mezcla1 = nullptr;

    delete [] mezcla2;
    mezcla2 = nullptr;

  }
  //EN CASO CONTRARIO AL PASO1: CASO BASE (SOLO MEZCLAMOS 1 VECTOR)
  else{
    for(int i=0; i<numElementos;i++)
    mezcla[i] = m[vec_inicial][i];
    }
}
```

```
void mezclaVectores(int * vec1, int * vec2, int dim1, int dim2, int * &mezcla){

  for(int i=0; i<dim1; i++)
    mezcla[i] = vec1[i];

  bool found;
  int k;

  for(int j=0; j<dim2; j++){
    k=0; found = false;
    while(!found && k<(dim1+dim2)){
      if(mezcla[k] >= vec2[j])
        found = true;
      else
        k++;
    }

    if(found){
      for(int p = dim1+j-1; p >= k; p--){
        mezcla[p+1] = mezcla[p];
      }
      mezcla[k] = vec2[j];
    }
    if(!found){
      mezcla[dim1+j] = vec2[j];
    }
  }
}
```

En el caso de divide y vencerás hemos definido 2 métodos. El primer método nos mezcla 2 vectores y los almacena en un vector resultado.El segundo método se encarga de hacer los cálculos recursivos. El método mezcla_DYV_recursivo lo que que hace es dividir la matriz dónde están almacenados los vectores en 2 vectores, y mediante recursividad esos 2 vectores a su vez se dividen en 2 vectores y así sucesivamente, hasta que llega a vectores de una sola componentes y vuelve hacia atrás mezclando cada uno de los subvectores obtenidos hasta llegar a mezclar los K vectores.

ALGORITMO DE DIVIDE Y VENCERÁS

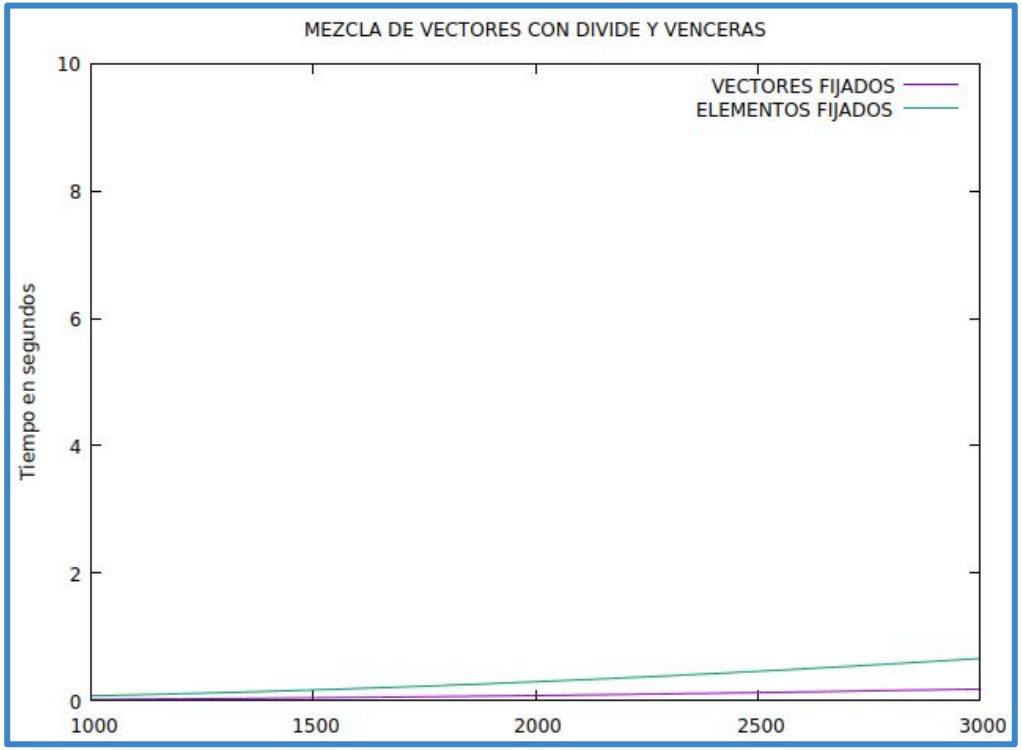
ANÁLISIS EMPÍRICO

NUMERO DE ELEMENTOS (VECTORES N=5)	TIEMPOS DE EJECUCIÓN EN SEGUNDOS
100	0.001329
496	0.006933
892	0.017853
1288	0.035513
1684	0.060093
2080	0.091773
2476	0.125806
2872	0.169719
3268	0.220153
3664	0.275771
4060	0.339713
4456	0.406758
4852	0.485479
5248	0.574904
5644	0.657258
6040	0.750443
6436	0.853993
6832	0.962448
7228	1.07375
7624	1.18986
8020	1.31601
8416	1.45522
8812	1.59453
9208	1.73925
9604	1.88978

NUMERO DE VECTORES (ELEMENTOS N=8)	TIEMPOS DE EJECUCIÓN EN SEGUNDOS
100	0.000883
496	0.019304
892	0.06202
1288	0.128148
1684	0.220234
2080	0.335605
2476	0.449421
2872	0.603817
3268	0.782334
3664	0.980963
4060	1.21511
4456	1.4598
4852	1.7286
5248	2.08436
5644	2.3484
6040	2.69669
6436	3.1304
6832	3.48079
7228	3.89226
7624	4.32137
8020	4.86858
8416	5.36424
8812	5.87993
9208	6.43104
9604	7.03543

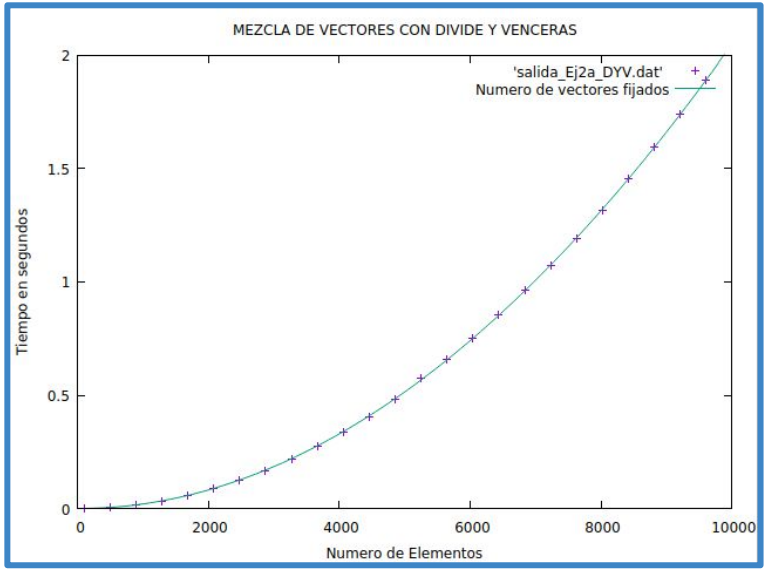
ALGORITMO DE DIVIDE Y VENCERÁS

ANÁLISIS EMPÍRICO (gráfica comparativa)



ALGORITMO DE DIVIDE Y VENCERÁS

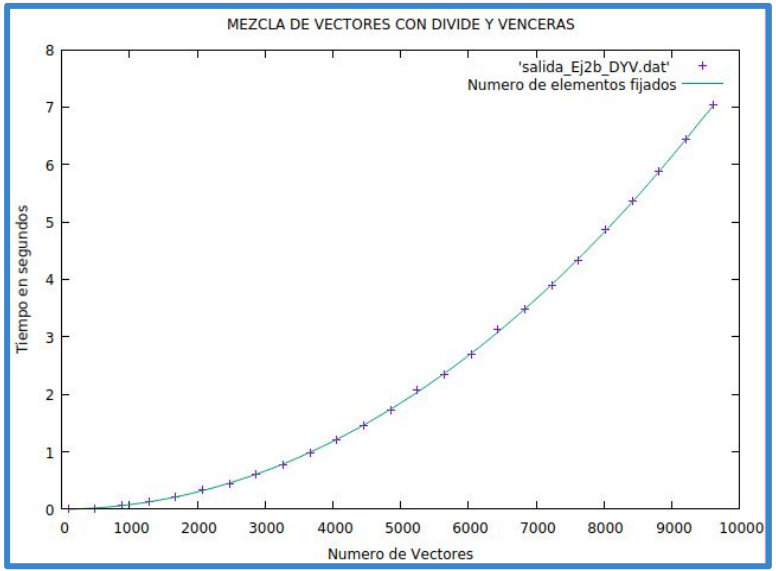
ANÁLISIS HÍBRIDO



$$f(x) = -3.61135 \cdot 10^{-14} \cdot x^3 + 2.091 \cdot 10^{-8} \cdot x^2 - 9.01735 \cdot 10^{-7} x + 0.00178979$$

ALGORITMO DE DIVIDE Y VENCERÁS

ANÁLISIS HÍBRIDO



$$f(x) = 7.11628 \cdot 10^{-13}\, x^3 + 6.82646 \cdot 10^{-8}\, x^2 + 1.07696 \cdot 10^{-5}\, x - 0.000955068$$

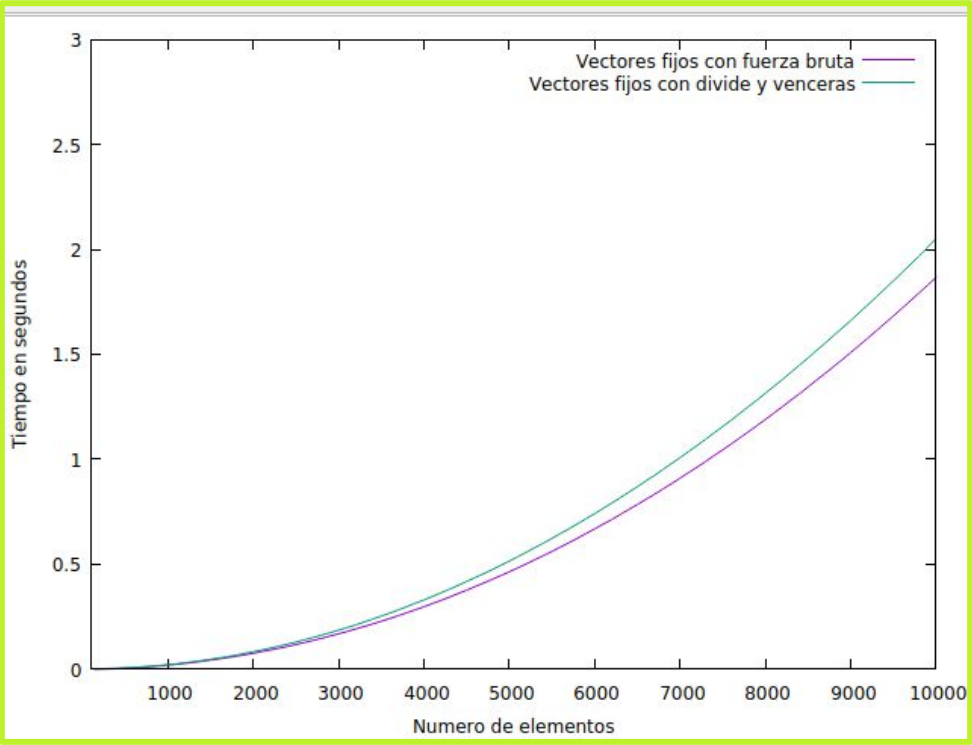
ALGORITMO DE DIVIDE Y VENCERÁS

Comparativa entre ambos algoritmos cuando los vectores son fijos

NUMERO DE ELEMENTOS (VECTORES N=5)	TIEMPOS DE EJECUCIÓN EN SEGUNDOS CON FB	TIEMPOS DE EJECUCIÓN EN SEGUNDOS CON DYV
100	0.001172	0.001329
496	0.006097	0.006933
892	0.015509	0.017853
1288	0.032932	0.035513
1684	0.053301	0.060093
2080	0.080623	0.091773
2476	0.113845	0.125806
2872	0.152556	0.169719
3268	0.197894	0.220153
3664	0.249331	0.275771
4060	0.30896	0.339713
4456	0.367212	0.406758
4852	0.438958	0.485479
5248	0.518436	0.574904
5644	0.609944	0.657258
6040	0.674232	0.750443
6436	0.762268	0.853993
6832	0.868134	0.962448
7228	0.964117	1.07375
7624	1.07277	1.18986
8020	1.19122	1.31601
8416	1.30782	1.45522
8812	1.43861	1.59453
9208	1.62673	1.73925
9604	1.69641	1.88978

ALGORITMO DE DIVIDE Y VENCERÁS

Comparativa entre ambos algoritmos cuando los vectores son fijos



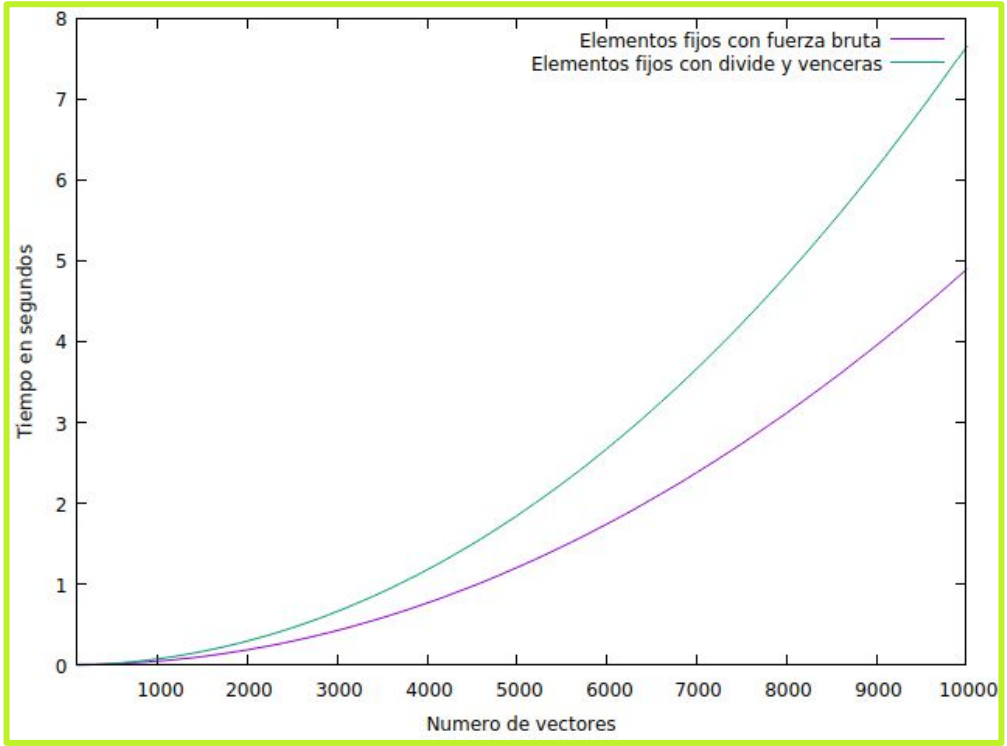
ALGORITMO DE DIVIDE Y VENCERÁS

Comparativa entre ambos algoritmos cuando los elementos son fijos

NUMERO DE VECTORES (ELEMENTOS N=8)	TIEMPOS DE EJECUCIÓN EN SEGUNDOS CON FB	TIEMPOS DE EJECUCIÓN EN SEGUNDOS CON DYV
100	0.00299	0.000883
496	0.016775	0.019304
892	0.040056	0.06202
1288	0.08454	0.128148
1684	0.151455	0.220234
2080	0.206555	0.335605
2476	0.292292	0.449421
2872	0.394914	0.603817
3268	0.510728	0.782334
3664	0.641054	0.980963
4060	0.785313	1.21511
4456	0.94878	1.4598
4852	1.14668	1.7286
5248	1.3385	2.08436
5644	1.53266	2.3484
6040	1.76466	2.69669
6436	2.00264	3.1304
6832	2.26571	3.48079
7228	2.53527	3.89226
7624	2.82307	4.32137
8020	3.12794	4.86858
8416	3.45626	5.36424
8812	3.84793	5.87993
9208	4.12697	6.43104
9604	4.47861	7.03543

ALGORITMO DE DIVIDE Y VENCERÁS

Comparativa entre ambos algoritmos cuando los elementos son fijos



Conclusiones del ejercicio 2

- El algoritmo implementado por fuerza bruta tarda menos que algoritmo implementado por divide y vencerás y esto se debe a que en el algoritmo de divide y vencerás usamos la recursividad.

- Vemos también que cada vez la gráfica de divide y vencerás se va alejando cada vez más de la gráfica de fuerza bruta, esto quiere decir que en ningún momento el algoritmo de divide y vencerás será mejor que el de fuerza bruta para resolver este ejercicio.

- No es buena idea usar la recursividad para ejercicios sencillos como este ya que no nos va reducir el tiempo de ejecución del algoritmo.

- Deducimos una importante conclusión y es hacer un análisis previo del problema ya que eso nos ayuda mucho a la hora de elegir el algoritmo más adecuado y eficiente para nuestro problema.


THE END