DIVIDE Y VENCERÁS
RELACIÓN DE EJERCICIOS Y PROBLEMAS

1. Ordene el siguiente vector de números enteros (9 1 3 5 0 4 2 6 8 7)

    a. ... utilizando el algoritmo Mergesort.

    b. ... utilizando el algoritmo Quicksort.

    c. ... de mayor a menor adaptando el algoritmo Mergesort.

    d. ... de mayor a menor adaptando el algoritmo Quicksort.

2. Modifique el algoritmo "divide y vencerás" para la multiplicación de grandes enteros de forma que permita multiplicar números en binario. Ilustre el funcionamiento del algoritmo a la hora de multiplicar los siguientes enteros: A = 10011011 y B = 10111010.

3. Supongamos que tenemos 3 algoritmos alternativos para resolver un problema:

    a. El algoritmo A resuelve el problema dividiéndolo en dos subproblemas de tamaño n/2, resuelve recursivamente cada subproblema y combina la solución en tiempo lineal.

    b. El algoritmo B resuelve el problema de tamaño n resolviendo dos subproblemas de tamaño n−1 y combinando la solución en tiempo constante.

    c. El algoritmo C resuelve el problema de tamaño n dividiéndolo en 9 subproblemas de tamaño n/3, resuelve recursivamente cada uno de esos subproblemas y combina las soluciones en tiempo $O(n^3)$.

    ¿Cuál es el tiempo de ejecución de cada uno de esos algoritmos (en notación O) y cuál escogería para resolver el problema?.

4. Sea T(n) la función que describe el tiempo de ejecución de un algoritmo "divide y vencerás". Si T(n) es de la forma $T(n) = aT(n/b) + n^k$ con $a \geq 1$, $b > 1$, analice el tiempo de ejecución del algoritmo dependiendo de los valores de a y b.

5. ¿Cuántas líneas, en función de n, imprime el siguiente programa?

```
void f (int n)
{
  if (n > 1) {
     printf("Línea...\n");
     f(n/2);
     f(n/2);
  }
}
```

6. Sea A un array bidimensional, de tamaño n×n, parcialmente ordenado. El criterio de ordenación es el siguiente: Los elementos en cada fila y columna se encuentran en orden ascendente. Esto es

   $A[i, j] \leq A[i, j + 1]$ con $i = \{1..n\}$ y $j = \{1..n-1\}$

   $A[i, j] \leq A[i + 1, j]$ con $i = \{1..n-1\}$ y $j = \{1..n\}$

   El siguiente algoritmo nos permite determinar si un determinado elemento x se encuentra o no en el array:

```
int inArray (int x, int **A, int n)
{
  int f,c;

  f = 1;
  c = n-1;

  while ( (f<=n-1) && (c>=0) ) {

        if ( x < A[f][c] )
           c = c-1;
        else if ( x > A[f][c] )
           f = f+1;
        else
           return 1;
  }

  return 0;
}
```

   a. Calcule el tiempo de ejecución del algoritmo `inArray`.

   b. Resuelva el problema anterior utilizando la técnica "divide y vencerás". Calcule la eficiencia del algoritmo "divide y vencerás" que diseñe y compárelo, en términos de eficiencia, con el algoritmo `inArray`.

7. Diseñe un algoritmo "divide y vencerás" que permita encontrar la mediana de un vector sin tener que ordenar el vector previamente.

8. Diseñe un algoritmo "divide y vencerás" que permita calcular el k−ésimo menor elemento de un vector.

9. Dado un vector de n elementos, de los cuales algunos están duplicados, diseñe un algoritmo O(n log n) que permita eliminar todos los elementos duplicados.

10. Dado un vector ordenado de números enteros X, diseñe un algoritmo "divide y vencerás" que permita determinar si existe un índice i tal que X[i] = i.

11. Supongamos que tenemos k arrays ordenados, cada uno con n elementos, y queremos combinarlos en un único array ordenado con kn elementos. Una posible alternativa consiste en, utilizando un algoritmo clásico, mezclar los dos primeros arrays, posteriormente mezclar el resultado con el tercero, y así sucesivamente.

    a. ¿Cuál sería el tiempo de ejecución de este algoritmo?

    b. Diseñe un algoritmo de mezcla más eficiente.

12. Un array se dice que tiene un elemento mayoritario si más de la mitad de sus elementos tienen el mismo valor. Dado un array A, nos proponen que diseñemos un algoritmo eficiente que nos permita determinar sin un array tiene un elemento mayoritario y, en caso afirmativo, identifique dicho elemento. Ahora bien, los elementos del array no tienen por qué pertener a un dominio ordenado como el de los números enteros y, por tanto, NO se pueden realizar comparaciones del tipo A[i] > A[j]. En cualquier caso, sí que se pueden realizar comprobaciones del tipo A[i] == A[j].

13. Dada la forma exacta de un conjunto de objetos en un plano, encuentre el perfil de los mismos (en dos dimensiones) que se obtiene al eliminar las líneas ocultas utilizando un algoritmo "divide y vencerás" y analice el tiempo de ejecución del algoritmo resultante.

    NOTA: Los objetos son de forma rectangular, codificados mediante cuatro coordenadas, empezando por la esquina superior izquierda y continuando en el sentido de las agujas del reloj.

14. Diseñe y analice la eficiencia de un algoritmo "divide y vencerás" para encontrar el par de puntos más cercano dentro de un conjunto de puntos en el plano.

    Se podría diseñar un algoritmo "específico" para este problema calculando las distancias entre todos los pares de puntos $O(n^2)$, pero la técnica "divide y vencerás" nos permite obtener una solución más eficiente de la siguiente forma:

    a. Dividir: Se encuentra una línea vertical *l* que divide el conjunto *P* en dos conjuntos P_I y P_D de forma que cada uno contiene la mitad de los puntos en P. Todos los puntos en P_I están a la izquierda de la línea *l* y todos los puntos en P_D se encuentran a la derecha. El vector X se divide en dos subvectores X_I y X_D que contienen los puntos en P_I y P_D ordenados según el valor de su coordenada x. De igual forma, el vector Y se divide en dos arrays Y_I e Y_D que contienen los puntos de P_I y P_D, ordenados según su coordenada y.

    b. Resolver: Una vez divididos los vectores, se hacen las llamadas recursivas para encontrar los puntos más cercanos en P_I y P_D. La primera de ellas toma como argumentos los puntos P_I y los vectores X_I e Y_I. La segunda considera los puntos P_D y los vectores X_D e Y_D. Sean δ_I y δ_D las distancias más pequeñas en P_I y P_D.

    c. Combinar: Los puntos más cercanos son aquéllos que se encontraron a menor distancia en las llamadas recursivas ($\delta = \min\{\delta_I, \delta_D\}$) o bien es un par con un punto en P_I y otro en P_D. El algoritmo, por tanto, debe encontrar si existe un par a una distancia menor que δ. La clave está en saber que estos puntos, de existir, se encuentran dentro de una ventana con 2δ de ancho y δ de alto centrada alrededor de la línea *l*.

15. Diseñe y analice la eficiencia de un algoritmo "divide y vencerás" para medir la similitud entre dos rankings.

    Muchos sitios web intentan comparar las preferencias de dos usuarios para realizar sugerencias a partir de las preferencias de usuarios con gustos similares a los nuestros. Dado un ranking de n productos (p.ej. canciones) mediante el cual los usuarios indicamos nuestras preferencias, un algoritmo puede medir la similitud de nuestras preferencias contando el número de inversiones:

    Dos productos i y j están "invertidos" en las preferencias de A y B si el usuario A prefiere el producto i antes que el j mientras que el usuario B prefiere el producto j antes que el i.

    Esto es, cuantas menos inversiones existan entre dos rankings, más similares serán las preferencias de los usuarios representados por esos rankings.

    > NOTA: Un algoritmo de fuerza bruta compararía las preferencias de los usuarios probando todos los pares posibles de productos $\Theta(n^2)$.