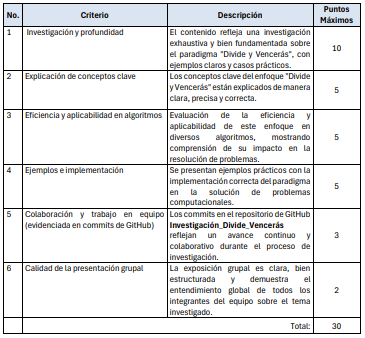
Consiste en una investigación profunda sobre el paradigma de resolución de problemas conocido como "Divide y Vencerás". Los estudiantes deberán explorar cómo este enfoque se utiliza en diversos algoritmos y casos prácticos, evaluando su eficiencia y aplicabilidad. Luego, presentarán sus hallazgos a través de una exposición grupal, donde explicarán los conceptos clave, ejemplos y su implementación en la solución de problemas computacionales. Además, deberán evidenciar el avance de su investigación realizando diversos commits en un repositorio de GitHub llamado Investigación\_Divide\_Vencerás, fomentando así la práctica de control de versiones y trabajo colaborativo en entornos de desarrollo.



INTRODUCCIÓN (Josué)

CONCEPTOS CLAVES:

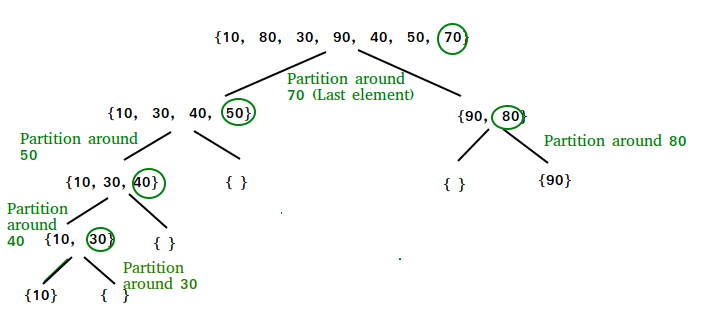
* Explicación General: (Enoc)
* DIVIDIR: (Dennis)

Partir el problema en subproblemas del mismo tipo. Este paso implica dividir el problema original en problemas más pequeños. Cada subproblema representará una parte del problema original. Normalmente, este paso utiliza un método recursivo para dividir el problema hasta que no se puedan crear subproblemas adicionales (Frías, 2021).

* VENCER: (Josué)
* COMBINAR: (Enoc)

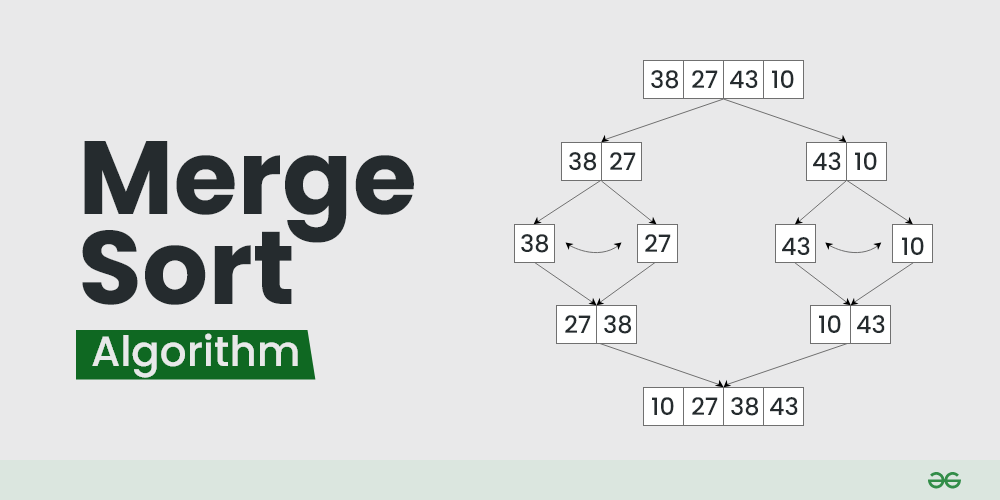
EJEMPLOS (Entre todos)

**Quicksort:** Es un algoritmo de ordenamiento. El algoritmo selecciona un elemento pivote, reorganiza los elementos de la matriz para que todos los elementos con valores inferiores al elemento pivote se muevan hacia la izquierda y los elementos con valores más altos vayan hacia la derecha. Finalmente, el algoritmo ordena recursivamente los subarreglos a la izquierda y a la derecha del elemento pivote (Frías, 2021).



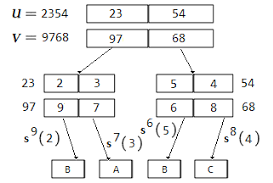
Este método sirve especialmente a la hora de buscar un valor en específico dentro de un array. Aunque en un array pequeño no sea necesario, en arrays de mayor tamaño, esta subdivisión ayuda a identificar el valor buscado de manera más rápida y eficiente.

**Merge Sort:** También es un algoritmo de ordenamiento. Este algoritmo divide la matriz en dos partes, cada parte se ordena de forma recursiva y finalmente une las dos partes ordenadas. La complejidad temporal de este algoritmo es , el mejor de los casos, el caso promedio y el peor de los casos. La complejidad del tiempo se puede entender fácilmente usando la ecuación: (Frías, 2021).



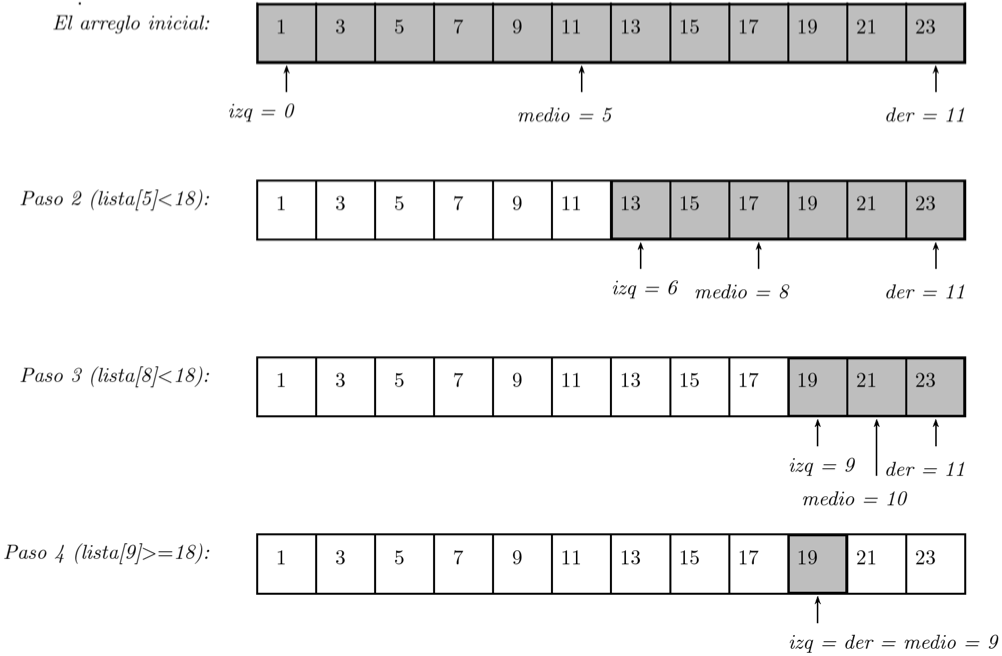
Este proceso sirve especialmente para la ordenación de arrays, como se puede observar en el ejemplo visual; en un array de mayor tamaño, la subdivisión ayudará a ordenar los valores con mayor rapidez. Este método posee un enfoque distinto al de Quicksort, ya que este último se centra en un solo valor, Merge Sort busca trabajar con todos los valores del array para ordenarlos.

**Algoritmo de Karatsuba:** Es el primer algoritmo de multiplicación que es asintóticamente más rápido que el algoritmo de multiplicación cuadrático clásico. Este algoritmo logra reducir la multiplicación de dos números de n dígitos a un máximo de , que es aproximadamente el logaritmo de 3 en base 2, es decir, , por lo tanto, , bajo la ecuación ; reduciendo esto a multiplicaciones de un solo dígito. Por lo tanto, es más rápido que el algoritmo clásico que requiere producto de números de un solo dígito (Frías, 2021).



CASOS PRÁCTICOS (Entre todos)

**Búsqueda Binaria:** Es un algoritmo de búsqueda. En cada paso, el algoritmo compara el parámetro de entrada (x) con el valor del elemento en el medio de la matriz. Si los valores coinciden, devuelve el índice del elemento del medio. Por el contrario, si x es mejor que el elemento del medio, el algoritmo devuelve la mitad izquierda del elemento del medio; de lo contrario, devuelve la mitad derecha del elemento del medio (Frías, 2021).



**Par de puntos más cercanos:** El problema es encontrar el par de puntos más cercano a otro par de puntos dado en el plano XY. Este problema se puede resolver en tiempo , calculando la distancia de cada par de puntos y comparando las distancias hasta encontrar el valor más pequeño. El algoritmo de tipo Dividir y Vencer resuelve este problema en un tiempo (Frías, 2021).

**Algoritmo de Delaunay:** En gráficos y geometría computacional, la triangulación de Delaunay se utiliza para dividir una superficie o un conjunto de puntos en triángulos. Este algoritmo utiliza Divide y Vencerás para crear la triangulación de manera eficiente. Es útil en gráficos 3D, modelado de terrenos y simulaciones físicas. Esto es especialmente así ya que una de las figuras 2D primarias es el triángulo, que puede poder formar otras figuras más complejas, como rectángulos y cuadrados; y en conjunto, todos estos dan a figuras geométricas para la creación de figuras 3D (Rodríguez, 2013).

|  | |
| --- | --- |
| **Sin Triangulizar** | **Triangulado** |

Además, la división de los cuadrados a triángulos lleva a complejizar la forma que se está intentando formar. No obstante, se evita un alto consumo de recursos de procesamiento, ya sea por ser la manera más básica en que un plano 2D se puede subdividir, o por ser la forma en que se puede “reducir” los planos 2D que conforman el objeto 3D.

|  |
| --- |
| **Más Optimización | Más Complejidad** |

En este caso práctico, se puede ver como el modelo de la roca sin triangulizar tiene una cantidad excesiva de caras, que puede consumir arduamente el procesamiento del dispositivo que intente cargar el modelo. No obstante, cuando la roca es triangulada, se puede ver como la cantidad de caras se reduce drásticamente, ya que se el Algoritmo de Delaunay ha encontrando la forma más óptima de disminuir las caras a través de triángulos.

|  | |
| --- | --- |
| **Sin Triangulizar** | **Triangulado** |

El Algoritmo hace que se encuentre las superficies “comunes” entre varias de las caras, para poderlas disolver (es decir, volverlas en una sola cara), y en el proceso se busca mantener la forma original del modelo. Para ello, también se usa como base otros métodos de Dividir y Vencer, como el de *Par de puntos más cercanos* de la aplicación anterior.

CASO PRÁCTICO APLICADO A UN SISTEMA EN C# (Entre todos)

**Búsqueda Binaria**

using System;

class BinarySearchExample

{

public static void Main(string[] args)

{

int[] array = { 3, 9, 10, 27, 38, 43, 82 };

int objetivo = 27;

int result = BusquedaBinaria(array, objetivo); //Llamada de la búsqueda binaria

if (result != -1)

Console.WriteLine("Elemento encontrado en la posición: " + result); //Devolución de elemento encontrado

else

Console.WriteLine("Elemento no encontrado."); //Devolución de elemento no encontrado

}

static int BusquedaBinaria(int[] array, int objetivo)

{

int izquierda = 0, derecha = array.Length - 1;

while (izquierda <= derecha)

{

// Encuentra el punto medio

int medio = izquierda + (derecha - izquierda) / 2;

// Compara el elemento del medio con el objetivo

if (array[medio] == objetivo)

return medio;

// Si el objetivo es menor, ignora la mitad derecha

if (array[medio] > objetivo)

derecha = medio - 1;

// Si el objetivo es mayor, ignora la mitad izquierda

else

izquierda = medio + 1;

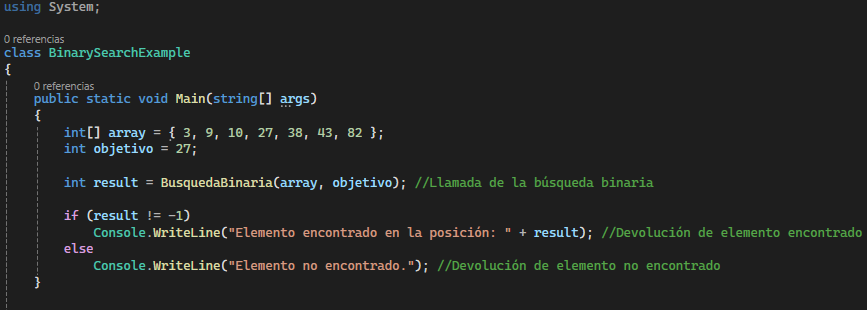
}

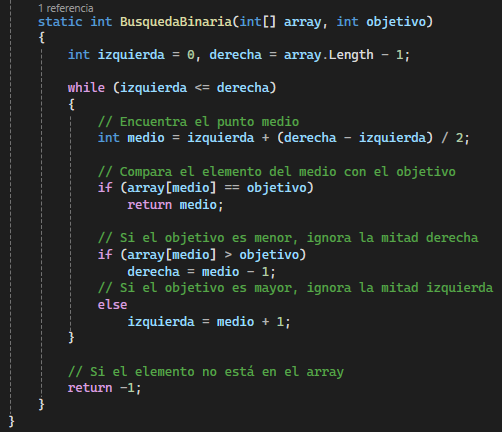
// Si el elemento no está en el array

return -1;

}

}





**Resultado:**





27 se encuentra en la posición 3 del array; visualmente, aparece en la posición 4, pero los arrays empiezan desde 0; entonces la posición 3 es técnicamente la posición 4, así que efectivamente, la búsqueda binaria cumplió su propósito.

REFERENCIAS (Dennis)

Frías, S. (2021, 4 de abril). *Significado del algoritmo divide y vencerás: Explicado con ejemplos*. freeCodeCamp. <https://www.freecodecamp.org/espanol/news/significado-del-algoritmo-divide-y-venceras/>

Rodríguez, E. (2013, 22 de marzo). *Triangulaciones de Delaunay*. Cinvestav-Tamaulipas. <https://www.tamps.cinvestav.mx/~ertello/gc/sesion18.pdf>