

# Tecnológico Nacional de México

Instituto Tecnológico de Tijuana

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN SEMESTRE SEPTIEMBRE 2021 – ENERO 2022

Ing. Sistemas Computacionales

Estructura de Datos - 2SC3CD

Investigación

Unidad 5 – Métodos de Ordenamiento

Guillen Martinez Anthony - 20210575

## M.C.C. LUZ ELENA CORTEZ GALVAN

FECHA DE ENTREGA 24/11/2020

## CONTENIDO DEL TRABAJO:

1. **INDICE**

**5.1.1 - Algoritmos de ordenamiento internos………………………………………….……3**

**5.1.2 - Burbuja……………………………………………………………………………….……5**

**5.1.3 - Shellsort………………………………………………………………............................6**

**5.1.4 - Quicksort………………………………………………………………………………….7**

**5.1.5 - Radix…………………………………………………………………………………….…8**

**5.2.1 - Algoritmos de ordenamiento externos………………………………………………9**

**5.2.2 - Intercalación Simple…………………………………………………………………...11**

**5.2.3 - Intercalación Cuadrática……………………………………………………………...12**

**5.2.4- Merge………………………………………………………………….…………………..13**

**Tabla comparativa……………………………………………………………………………...15**

**Conclusiones……………………………………………………………………………………15**

**Bibliografias……………………………………………………………………………………..16**

1. **Material de cada tema investigado.**

**5.1.1 – Algoritmos de ordenamiento internos.**

**Qué es un método de ordenamiento**

* El ordenamiento consiste en ordenar los registros de un archivo homogéneo basándose en el valor de alguna llave y puede ser descendente o ascendente.
* El ordenamiento es una de las aplicaciones más utilizadas en la computación.
* La mayoría de los datos están ordenados de alguna manera.
* Muchos de los cálculos invocan internamente a un método de ordenamiento.

**Porqué son llamados de ordenamiento interno**

Ordenamiento interno: Se lleva a cabo completamente en memoria principal. Todos los objetos que se ordenan caben en la memoria principal de la computadora (RAM).

La ordenación interna o de arreglos, recibe este nombre ya que los elementos o componentes del arreglo se encuentran en la memoria principal de la computadora.

**Cuales métodos de ordenamiento interno existen**

**Burbuja**

El método de burbuja también se le puede llamar como Método de "intercambio directo". El algoritmo ordena los elementos del arreglo utilizando el método de la burbuja. Transporta en cada pasada el elemento más pequeño hacia la parte de izquierda del arreglo. Este ordenamiento es eficiente sólo en listas pequeñas (10 elementos).El método de burbuja va comparando cada elemento del arreglo con el siguiente; si un elemento es mayor que el que le sigue, entonces se intercambian; esto producirá que en el arreglo quede como su último elemento, el más grande. Este proceso deberá repetirse recorriendo todo el arreglo hasta que no ocurra ningún intercambio. Los elementos que van quedando ordenados ya no se comparan. “Baja el más pesado”. Esté método se basa en el principio de comparar pares de elementos adyacentes intercambiarlos entre sí hasta que estén todos ordenados.

**Quicksort (ORDENAMIENTO RAPIDO)**

Es un algoritmo basado en la técnica de “divide y vencerás” que permite en promedio ordenar n elementos en un tiempo proporcional a n log. n. el algoritmo original es recursivo, pero se utilizan versiones interactivas para mejorar su ordenamiento.

**ShellSort**

El método de Shell es una versión mejorada del método de inserción directa. Y recibe el nombre en honor a su autor, Donald Shell. El método también se conoce como de inserción con incrementos decrecientes. En el método de ordenación por inserción directa cada elemento se compara para su ubicación correcta en el arreglo con los elementos que se encuentren en la parte izquierda del sí mismo. Si el elemento a insertar es más pequeño que el grupo de elementos que se encuentran a su izquierda, es necesario efectuar entonces varias comparaciones antes de su ubicación.

**Radix**

Es un algoritmo que ordena números enteros de forma individual, es decir del menos significativo al más significativo. Este ordenamiento consta de 10 dígitos de (0-9), su ordenamiento es del más significativo al menos significativo de “menor a mayor”.

**Ordenamiento por Selección**

El algoritmo de ordenación por método de selección directa es relativamente sencillo y fácil de recordar e implementar. Se basa en realizar varias pasadas, intentando encontrar en cada una de ellas el elemento que según criterio de ordenación es mínimo y colocando posteriormente en su sitio.

**Ordenamiento HeapSort.**

Su desempeño es en promedio tan bueno como el Quicksort y se comporta mejor que este último en los peores casos. Aunque el Heapsort tiene un mejor desempeño general que cualquier otro método presentado de clasificación interna, es bastante complejo de programar.

**Programa ejemplo en C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace PruebaVector

{

    class PruebaVector

    {

        private int[] vector;

        public void Cargar()

        {

            Console.WriteLine("Metodo de insercion directa");

            Console.Write("Cuantos longitud del vector:");

            string linea;

            linea = Console.ReadLine();

            int cant;

            cant = int.Parse(linea);

            vector = new int[cant];

            for (int f = 0; f < vector.Length; f++)

            {

                Console.Write("Ingrese elemento "+(f+1)+": ");

                linea = Console.ReadLine();

                vector[f] = int.Parse(linea);

            }

        }

        public void InsercionDirecta()

        {

            int auxili;

            int j;

            for (int i = 0; i < vector.Length; i++)

            {

                auxili = vector[i];

                j = i - 1;

                while (j >= 0 && vector[j] > auxili)

                {

                    vector[j + 1] = vector[j];

                    j--;

                }

                vector[j + 1] = auxili;

            }

        }

        public void Imprimir()

        {

            Console.WriteLine("Vector ordenados en forma ascendente");

            for (int f = 0; f < vector.Length; f++)

            {

                Console.Write(vector[f]+"  ");

            }

            Console.ReadKey();

        }

        static void Main(string[] args)

        {

            PruebaVector pv = new PruebaVector();

            pv.Cargar();

            pv.InsercionDirecta();

            pv.Imprimir();

        }

    }

}

**Tarea 5.1.2 - Burbuja.**

**Concepto/Descripción**

Es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

El procedimiento de la burbuja es el siguiente: § Ir comparando desde la casilla 0 número tras número hasta encontrar uno mayor, si este es realmente el mayor de todo el vector se llevará hasta la última casilla, si no es así, será reemplazado por uno mayor que él. § Este procedimiento seguirá así hasta que haya ordenado todas las casillas del vector. § Una de las deficiencias del algoritmo es que ya cuando a ordenado parte del vector vuelve a compararlo cuando esto ya no es necesario.

**Funcionamiento**

Este método consiste en ir comparando cada par de elementos del array e ir moviendo el mayor elemento hasta la última posición, comenzando desde la posición cero. Una vez acomodado el mayor elemento, prosigue a encontrar y acomodar el segundo más grande comparando de nuevo los elementos desde el inicio de la lista, y así sigue hasta ordenar todos los elementos del arreglo.

**Algoritmo**

Public int[] OrdenarBurbuja(int[]x)

{

int t= x.Length, temp;

for(int i=1 ; i< t ; i++)

for(int j = t-1 ; j >= i; j--)

{

if(x[j] < x[j-1])

{

temp= x[j];

x[j]= x[j-1];

x[j-1]= temp;

}

}

return x;

}

**Ventajas y Desventajas**

La ventaja principal del ordenamiento de burbuja es que es muy popular y fácil de implementar. Además, en este tipo de ordenamiento, los elementos se intercambian sin utilizar almacenamiento temporal adicional, de modo que el espacio requerido es el mínimo. La principal desventaja del ordenamiento de burbuja es el hecho de que no se comporta adecuadamente con una lista que contenga un número grande de elementos. Esto se debe a que este ordenamiento requiere n al cuadrado de pasos de procesamiento para cada n número de elementos a ser ordenados. Como tal, este tipo de ordenamiento es más apropiado para la enseñanza académica pero no para aplicaciones de la vida real.

**5.1.3 - Shell**

**Concepto/Descripción**

Es un algoritmo de ordenación interna muy sencillo pero muy ingenioso, basado en comparaciones e intercambios, y con unos resultados radicalmente mejores que los que se pueden obtener con el método de la burbuja, el des elección directa o el de inserción directa.

**Funcionamiento**

El ShellSort es una mejora del método de inserción directa que se utiliza cuando el número de elementos a ordenar es grande. El método se denomina “shell” –en honor de su inventor Donald Shell – y también método de inserción con incrementos decrecientes.

**Algoritmo**

public void Shell()

        {

            int salto = 0;

            int sw = 0;

            int auxi = 0;

            int e = 0;

            salto = vector.Length / 2;

            while (salto > 0)

            {

                sw = 1;

                while (sw != 0)

                {

                    sw = 0;

                    e = 1;

                    while (e <= (vector.Length - salto))

                    {

                        if (vector[e - 1] > vector[(e - 1) + salto])

                        {

                            auxi = vector[(e - 1) + salto];

                            vector[(e - 1) + salto] = vector[e - 1];

                            vector[(e - 1)] = auxi;

                            sw = 1;

                        }

                        e++;

                    }

                }

                salto = salto / 2;

            }

        }

**Ventajas y Desventajas**

Ventajas

* Es un algoritmo muy simple teniendo un tiempo de ejecución aceptable.
* Es uno de los algoritmos más rápidos.
* No requiere memoria adicional.
* Fácil implementación.

Desventajas

* Su complejidad es difícil de calcular y depende mucho de la secuencia de incrementos que utilice.
* Shell Sort es un algoritmo no estable porque se puede perder el orden relativo inicial con facilidad.
* Es menos eficiente que los métodos Merge, Heap y Quick Sort.
* Realiza numerosas comparaciones e intercambios.

**5.1.4 - Quicksort**

**Concepto/Descripción.**

El método de ordenamiento Quick Sort es actualmente el más eficiente y veloz de los métodos de ordenación interna. Es también conocido con el nombre del método rápido y de ordenamiento por partición, en el mundo de habla hispana.

Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de Quick Sort por la velocidad con que ordena los elementos del arreglo. Su autor C.A. Hoare lo bautizó así.

**Funcionamiento.**

* Se toma un elemento x de una posición cualquiera del arreglo.
* Se trata de ubicar a x en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales a x y todos los elementos que se encuentren a su derecha sean mayores o iguales a x.
* Se repiten los pasos anteriores pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de x en el arreglo.

**Algoritmo.**

1. Elegir un elemento de la lista de elementos a ordenar, al que llamaremos [pivote](https://www.ecured.cu/index.php?title=Pivote&action=edit&redlink=1).
2. Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
3. La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
4. Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

**Ventajas y Desventajas.**

Ventajas:

* Requiere de pocos recursos en comparación a otros métodos de ordenamiento.
* En la mayoría de los casos, se requiere aproximadamente N log N operaciones.
* Ciclo interno es extremadamente corto.
* No se requiere de espacio adicional durante ejecución (in-place processing).

Desventajas:

* Se complica la implementación si la recursión no es posible.
* Peor caso, se requiere N^2
* Un simple error en la implementación puede pasar sin detección, lo que provocaría un rendimiento pésimo.
* No es útil para aplicaciones de entrada dinámica, donde se requiere reordenar una lista de elementos con nuevos valores.
* Se pierde el orden relativo de elementos idénticos.

**5.1.5 - Radix**

**Concepto/Descripción**

El ordenamiento Radix (radix sort en inglés) es un [algoritmo de ordenamiento](https://www.ecured.cu/Algoritmo_de_ordenamiento) que ordena enteros procesando sus dígitos de forma individual. Como los enteros pueden representar cadenas de caracteres (por ejemplo, nombres o fechas) y, especialmente, números en punto flotante especialmente formateados, radix sort no está limitado sólo a los enteros. La mayor parte de los ordenadores digitales representan internamente todos sus datos como representaciones electrónicas de números binarios, por lo que procesar los dígitos de las representaciones de enteros por representaciones de grupos de dígitos binarios es lo más conveniente. Existen dos clasificaciones de radix sort: el de dígito menos significativo (LSD) y el de dígito más significativo (MSD). Radix sort LSD procesa las representaciones de enteros empezando por el dígito menos significativo y moviéndose hacia el dígito más significativo. Radix sort MSD trabaja en sentido contrario.

**Funcionamiento**

En programación un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación o reordenamiento de la entrada que satisfaga la relación de orden dada. Las relaciones de orden más usadas son el orden numérico y el orden lexicográfico. Ordenamientos eficientes son importantes para optimizar el uso de otros algoritmos (como los de búsqueda y fusión) que requieren listas ordenadas para una ejecución rápida.

**Algoritmo**

1. Calcular el módulo del primer número de la lista entre r.
2. El resultado anterior lo divides entre d, al estar trabajando con enteros este resultado se truncara y dependiendo de cuál sea este resultado se acomoda en la lista que le corresponde.
3. Se hacen los 2 pasos anteriores con los demás números del arreglo y se van acomodando en las listas correspondientes detrás del último número que haya entrado a dicha lista.
4. Acomodar el arreglo según como están acomodados en las listas.
5. Cuando se termine con todos los números del arreglo m y r se multiplican por 10.
6. Se repiten los 4 pasos anteriores hasta que se evalúen todos los dígitos.

**Ventajas y Desventajas**

Ventajas

* Radix Sort es estable, preservando la orden de elementos iguales.
* Radix Sort funciona en un tiempo lineal, en comparación de varios otros métodos de ordenamiento.
* El tiempo de ordenar cada elemento es constante, ya que no se hacen comparaciones entre elementos.  
  Radix Sort es particularmente eficiente cuando se tratan con grandes grupos de números cortos.

Desventajas

* Radix Sort no funciona tan bien cuando los números son muy largos, ya que el total de tiempo es proporcional a la longitud del número más grande y al número de elementos a ordenar.

**5.2.1 – Algoritmos de ordenamiento externos.**

**Porqué son llamados de ordenamiento externo**

Los algoritmos de ordenamiento externo son aquellos que para su uso utiliza la memoria secundaria, es decir disco duro.

Estos algoritmos se usan cuando se tienen registros y/o archivos, ya que su uso es para el manejo de datos ya sean del mismo tipo o distinto tipo y estos pueden quedar almacenados ya ordenados.

Llamamos ordenamiento externo cuando debemos ordenar archivos que son (mucho) más grandes de lo que nuestra memoria puede llegar a abarcar. Esto es así porque, en caso contrario, podríamos simplemente cargar el archivo en memoria y ordenarlo usando cualquier algoritmo de ordenamiento convencional.

**Cuales métodos existen**

* Mezcla directa
* Mezcla natural
* Intercalación simple
* Intercalación cuadrática
* Intercalación natural
* Intercalación directa
* Intercalación balanceada
* Intercalación polifásica
* Método de Hash
* Polyphase sort
* Natural merging
* Straigth merging
* Balanced mutiway merging

**Programa ejemplo**

// C# Program to implement

// Odd-Even / Brick Sort

using System;

class GFG

{

    public static void oddEvenSort(int []arr, int n)

    {

        // Initially array is unsorted

        bool isSorted = false;

        while (!isSorted)

        {

            isSorted = true;

            int temp =0;

            // Perform Bubble sort on

            // odd indexed element

            for (int i = 1; i <= n - 2; i = i + 2)

            {

                if (arr[i] > arr[i+1])

                {

                    temp = arr[i];

                    arr[i] = arr[i+1];

                    arr[i+1] = temp;

                    isSorted = false;

                }

            }

            // Perform Bubble sort on

            // even indexed element

            for (int i = 0; i <= n - 2; i = i + 2)

            {

                if (arr[i] > arr[i+1])

                {

                    temp = arr[i];

                    arr[i] = arr[i+1];

                    arr[i+1] = temp;

                    isSorted = false;

                }

            }

        }

        return;

    }

    // Driver code

    public static void Main ()

    {

        int []arr = {34, 2, 10, -9};

        int n = arr.Length;

        // Function calling

        oddEvenSort(arr, n);

        for (int i = 0; i < n; i++)

            Console.Write(arr[i] + " ");

        Console.WriteLine(" ");

    }

}

**5.2.2 - Intercalación Simple**

**Concepto/Descripción**

Se tienen dos archivos ordenados y se obtiene al final un solo archivo ordenado que contiene los elementos de los dos archivos iníciales.

Para utilizar el método se inicia con un vector de n posiciones. Comencemos con el subíndice i en la segunda posición incrementa en 1, el elemento del subíndice del vector se elimina de la secuencia y se reinserta en el vector en la posición adecuada.

**Funcionamiento**

Es un proceso utilizado en sistema de actualización, y en casos en que es necesario varias listas ordenadas. También es la única forma que hay para el ordenamiento de archivos, debido a la imposibilidad de almacenarlos en memoria y a limitaciones en el tiempo, por la cantidad de elementos a ordenar. Es el caso más simple, cuando se intercalan dos listas o archivos. Suponga que se tienen los arreglos ordenados A1 y A2 con elementos n1 y n2 respectivamente. Se desean intercalar en el arreglo A3.

**Algoritmo**

void Merge ( float a[ ] , float b[ ] , int n , int m , float c[ ] )  
{  
   int i , j , k , p ;  
  
   i = 0 ; j = 0 ; k = 0 ;  
   while ( i < n && j < m )  
   {  
      if ( a [ i ] <= b [ j ] )  
      { c [ k ] = a [ i ] ; i++ ; }  
      else  
      { c [ k ] = b [ j ] ; j++ ; }  
      k++ ;  
   }  
   if ( i == n )  
      for ( p = j ; p < m ; p++ ) { c [ k ] = b [ p ] ; k++ ; }  
   else  
      for ( p = i ; p < n ; p++ ) { c [ k ] = a [ p ] ; k++ ; }  
}

**Ventajas y Desventajas**

Ventajas

* Método de ordenamiento estable mientras la función de mezcla sea implementada correctamente.
* Muy estable cuando la cantidad de registros a acomodar es de índice bajo, en caso contrario gasta el doble del espacio que ocupan inicialmente los datos.
* Efectivo para conjunto de datos a los que se puede acceder secuencialmente (arreglos, vectores, etc.)

Desventajas

* Principal desventaja: está definido recursivamente. Si se deseara implementarla no recursivamente se tendría que emplear una pila y se requeriría un espacio adicional de memoria para almacenarla.

**5.2.3 – Intercalación cuadrática**

**Concepto/Descripción**

La idea central de este algoritmo consiste en realizar sucesivas particiones y fusiones a un arreglo para producir secuencias ordenadas de longitud cada vez mayor. En la primera pasada la longitud de la partición es de 1 y la fusión produce una secuencia ordenada de longitud 2.

**Funcionamiento**

En la primera pasada la longitud de la partición es de 1 y la fusión produce una secuencia ordenada de longitud 2. En la segunda pasada las longitudes se duplican y este proceso se repite hasta que la longitud de la secuencia de la partición sea mayor o igual que el número de elementos del arreglo original.

**Algoritmo**

1.- Inicio  
2.- Dividir el arreglo en N subarreglos de tamaño 1 e intercalar pares adyacentes separados de los subarreglos.  
3.- Incrementar las particiones del arreglo en duplos, cuadruplos, etcétera, y así sucesivamente.  
4.- Repetir el proceso hasta que solo quede un arreglo de tamaño N.  
5.- Fin del algoritmo.

**Ventajas y Desventajas**

Ventajas

* Eficaz
* Sencillo
* Código reducido para realiza el ordenamiento

Desventajas

* Consume bastante tiempo de computadora
* Requiere muchas lecturas/escrituras en memoria

**5.2.4 - Merge**

**Concepto/Descripción**

Cuando se dispone de dos vectores ya ordenados y se desea obtener un tercer vector también ordenado, se puede realizar la ordenación con un método denominado Mezcla (Merge en ingles). Supongamos que A es un vector ordenado de m elementos y B es otro vector ordenado de n elementos. La operación de mezcla producirá un nuevo vector de m + n elementos.

**Funcionamiento**

El método más sencillo, pero menos eficaz, consiste en colocar una lista detrás de la otra y luego ordenarla. Sin embargo este método no aprovecha la propiedad de que los vectores A y B ya están ordenados, por ello debe recurrir normalmente al sistema de mezcla el cual cosiste en comparar los dos primeros elementos de los vectores (A y B) y enviar al menor al tercer vector, continuando con el elemento comparado pero no enviado con el siguiente elemento del vector que contiene al elemento menor comparado anteriormente y así sucesivamente. Una vez que se terminaron los elementos de un vector, se procede a vaciar los elementos restantes del otro vector.

**Algoritmo**

m=tamaño del vector1

n=tamaño del vector2

int m,n,i=0,j=0,k=0,p;

while( i < m && j < n ){

if( vec1 <= vec2[j] ){

mezcla[k]=vec1;

i++;

}

else{

mezcla[k]=vec2[j];

j++;

}

k++;

}

if( i>= m){

for( p=j; p < n; p++){

mezcla[k]=vec2[p];

k++;

}

}

if( j>=n ){

for( p=i; p < m; p++ ){

mezcla[k]=vec1[p];

k++;

}

}

**Ventajas y Desventajas**

Ventajas

* Método de ordenamiento estable mientras la función de mezcla sea implementada correctamente.
* Muy estable cuando la cantidad de registros a acomodar es de índice bajo, en caso contrario gasta el doble del espacio que ocupan inicialmente los datos.
* Efectivo para conjunto de datos a los que se puede acceder secuencialmente (arreglos, vectores, etc.)

Desventajas

* Principal desventaja: está definido recursivamente. Si se deseara implementarla no recursivamente se tendría que emplear una pila y se requeriría un espacio adicional de memoria para almacenarla.

**CAPITALES:**

| **Método de**  **Ordenamiento:** | **Complejidad Temporal** | **Complejidad Espacial** | **Número de Pasadas** | **Número de**  **Comparaciones** | **Número de**  **Intercambios** |
| --- | --- | --- | --- | --- | --- |
| **1.-BURBUJA.** | **17.4865123** | **12578** | **32** | **512** | **54** |
| **2.-SHELL.** | **11.5896475** | **13254** | **6** | **145** | **351** |
| **3.-QUICKSORT.** | **14.4756647** | **13632** | **40** | **127** | **51** |
| **4.-RADIX.** |  | **12879** |  |  |  |
| **5.-Intercalación Simple.** | **27.4879563** | **12796** | **15** | **155** | **199** |
| **6.-Intercalación**  **Cuadrática.** | **09.8796532** | **13658** | **5** | **120** | **320** |
| **7.-MERGE.** | **07.4789541** | **13124** | **6** | **118** | **190** |

**NÚMEROS DE CONTROL:**

| **Método de**  **Ordenamiento:** | **Complejidad Temporal** | **Complejidad Espacial** | **Número de Pasadas** | **Número de**  **Comparaciones** | **Número de**  **Intercambios** |
| --- | --- | --- | --- | --- | --- |
| **1.-BURBUJA.** | **14.2549475** | **13312** | **50** | **1250** | **184** |
| **2.-SHELL.** | **09.9516706** | **13236** | **6** | **253** | **351** |
| **3.-QUICKSORT.** | **17.8502363** | **13308** | **79** | **264** | **51** |
| **4.-RADIX.** |  |  | **9** | **0** | **99** |
| **5.-Intercalación Simple.** | **23.5970614** | **13740** | **69** | **215** | **142** |
| **6.-Intercalación**  **Cuadrática.** | **06.9580243** | **12603** | **6** | **226** | **600** |
| **7.-MERGE.** | **08.9247211** | **13528** | **6** | **125** | **192** |

**Conclusiones**

Los métodos de ordenación juegan un papel muy importante en la estructura de los datos. Cada método tiene sus ventajas y desventajas, que son diferentes a la cantidad de datos a procesar, por lo que es muy importante saber utilizar herramientas de comparación y poder realizar comparaciones . Calcule el símbolo Big Or para comprender el mejor caso para cada situación. Ayuda a buscar una gran cantidad de registros con un modelo de eficiencia en el momento adecuado. Con su tecnología, podemos colocar listas después de otras listas y luego ordenarlas, porque también podemos comparar valores clave emparejados e intercambiarlos si no están en la posición correcta.

1. **Fuentes de Información.**

<http://academicos.azc.uam.mx/jfg/diapositivas/almacenamiento/Unidad_6.pdf>

<http://mapaches.itz.edu.mx/~mbarajas/edinf/Ordenamiento.pdf>

<http://itpn.mx/recursosisc/3semestre/estructuradedatos/Unidad%20V.pdf>

<https://cotearratia.wordpress.com/unidad-i-algoritmo-de-busqueda-y-ordenamiento/eficiencia-de-los-algoritmos/tipos-de-ordenamiento/>

<https://www.ecured.cu/Ordenamiento_de_burbuja>

<https://techlandia.com/ventajas-desventajas-algoritmos-ordenamiento-info_181515/>

<http://csharp-facilito.blogspot.com/2013/07/metodo-de-ordenamiento-quick-sort-en-c-sharp.html>

<https://www.ecured.cu/QuickSort>

<https://quicksortweb.wordpress.com/2017/10/07/ventajas-desventajas-y-aplicaciones/>

<https://www.ecured.cu/Ordenamiento_Radix>

<http://csharp-facilito.blogspot.com/2013/07/metodos-de-ordenamiento-en-c-sharp.html#:~:text=Descargas%20%E2%96%BC-,Metodos%20de%20Ordenamiento%20en%20C%23,la%20relaci%C3%B3n%20de%20orden%20dada>

<https://sites.google.com/a/uabc.edu.mx/radix-sort/algoritmo>

<http://radixsort.byethost7.com/?i=1>

<https://itslr.edu.mx/archivos2013/TPM/temas/s3u5.html>

<https://algoritmos-rw.github.io/algo2/material/apuntes/ordenamientos_externos/>

<https://www.geeksforgeeks.org/odd-even-sort-brick-sort/>

<http://metodosbusqueda.blogspot.com/2016/03/ordenamiento-por-intercalacion-simple.html>

<https://vdocuments.mx/metodo-de-intercalacion.html>

<http://themergesort.blogspot.com/2012/05/ventajas-y-desventajas.html>