METODOS DE ORDENAMIENTO

Julián Andrés Ruiz Jaramillo, Paulo Cesar Agudelo Jiménez.
Corporación de Estudios Tecnológicos del Norte del Valle
Calle 10 No 3-95 Cartago Valle del Cauca.
julianandresruizjaramillo73@gmail.com , pagudeloj@gmail.com.
3122675102 - 3217945165

***Resumen*- ordenamiento es la operación de arreglar los registros de una tabla en algún orden especial de acuerdo a un criterio de ordenamiento, el ordenamiento se efectúa con base en el valor de algún campo en un registro.**

En este trabajo se observara gran cantidad de información, sobre 6 métodos, recordando que estos métodos son de gran ayuda e importancia con grandes cantidades de datos.

Abstract: Ordering is the operation of arranging the records of a table in some special order according to a sorting criterion, sorting is done based on the value of some field in a record.

In this work we observe a great amount of information, about 6 methods, remembering that these methods are of great help and importance with large amounts of data.

- INTRODUCCIÓN

Debido a que las estructuras de datos son utilizadas para almacenar información y para poder recuperar esa información de manera eficiente, es deseable que este ordenada, los Algoritmos de ordenamiento nos permiten como su nombre lo dice, ordenar un grupo de datos para que queden en una secuencia tal que presente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente. El propósito principal de un ordenamiento es el de facilitar las búsquedas de los registros del conjunto ordenado, es importante destacar que existen diferentes métodos de ordenamiento como lo son el método por inserción, por mezcla, por montones (heap sort), rápido (Quicksort), por conteo (Counting sort), radix sort entre otros.

Abstract: Because data structures are used to store information and to be able to retrieve that information efficiently, it is desirable that this orderly, ordering algorithms allow us as their name says, sort a group of data to be in a Sequence such that it presents an order, which can be numeric, alphabetical or even alphanumeric, ascending or descending. The main purpose of an ordering is to facilitate the searches of the records of the ordered set, it is important to note that there are different sorting methods such as the insertion method, heap sort, quicksort, Counting sort, radix sort among others.

Palabras claves: alfanumérico, métodos, algoritmo, estructura, registro, datos, tiempo, hilo.

Keywords: Alphanumeric, methods, algorithm, structure, register, date, time, thread.

- Lenguaje De Programación Usado

Utilizamos el lenguaje de programación Python el cual es fácil de aprender, cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos, el intérprete de Python y la extensa biblioteca estándar están a libre disposición desde el sitio web de Python, implementación que se realiza con hilos, para mejorar el rendimiento del programa que se esta ejecutando, de igual forma se trabaja orientada a objetos que es una programación mucho más sencilla, pero que la hace un poco más lenta.

- Algoritmos usados

Ordenamiento por inserción: toma uno por uno los elemento de un arreglo y recorrerlo hacia su posición con respecto a los anteriormente ordenados.

Ordenamiento por mezcla (Merge sort): divide la lista desordenada en dos sublistas hasta tener un elemento en cada lista, luego lo compara con el que está a su lado y lo sitúa donde corresponde.

Ordenamiento por montones (heap sort): basado con comparación, usa el heap como estructura de datos, el cual representa un árbol, más lento que otros métodos pero más eficaz.

Ordenamiento rápido (Quicksort): es el más eficiente y veloz de los métodos de ordenamiento interna.

Ordenamiento por conteo (Counting sort): en él se cuenta el número de elementos de cada clase para luego ordenarlos, se recorren todos los elementos a ordenar y se cuenta el número de apariciones de cada elemento.

Ordenamiento Radix sort: procesa sus dígitos de forma individual, no está limitado solo a enteros.

- METODOLOGIA UTILIZADA

Se utilizaron diferentes herramientas, tales como internet, libros y ayudas con docentes.

- CODIGO DE LOS ALGORITMOS IMPLEMENTADOS

Inserción:

```
def insercion(lista, tam):
    for i in range(1, tam):
        v = lista[i]
        j = i - 1
        while j >= 0 and lista[j] > v:
            lista[j + 1] = lista[j]
            j = j - 1
        lista[j + 1] = v
```

Mezcla:

```
def mergesort(alist):
    #("Desordenado ",alist)
    if len(alist)>1:
        mid = len(alist)//2
        lefthalf = alist[:mid]
        righthalf = alist[mid:]

        mergesort(lefthalf)
        mergesort(righthalf)
        i=0
        j=0
        k=0
        while i < len(lefthalf)and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                alist[k]=lefthalf[i]
                i=i+1
            else:
                alist[k]=righthalf[j]
                j=j+1
               k=k+1

        while i < len(lefthalf):
            alist[k]=lefthalf[i]
            i=i+1
            k=k+1
        while j < len(righthalf):
            alist[k]=righthalf[j]
            j=j+1
            k=k+1
    #print("Ordenando ",alist)
```

Heap Sort:

```
def heapsort(lista, tam):
    for k in range(tam - 1, -1, -1):
        for i in range(0, k):
            item = lista[i]
            j = (i + 1) / 2
            while j >= 0 and lista[j] < item:
                lista[i] = lista[j]
                i = j
                j = j / 2
            lista[i] = item
        temp = lista[0];
    lista[0] = lista[k];
    lista[k] = temp;
```

Quicksort:

```
def quicksort(lista, izq, der):
    i = izq
    j = der
    x = lista[(izq + der) / 2]

    while (i <= j):
        while lista[i] < x and j <= der:
            i = i + 1
        while x < lista[j] and j > izq:
            j = j - 1
        if i <= j:
            aux = lista[i];
            lista[i] = lista[j];
            lista[j] = aux;
            i = i + 1;
            j = j - 1;
       if izq < j:
            quicksort(lista, izq, j);
        if i < der:
```

```
    quicksort(lista, i, der);
```

Counting sort:

```
def count_sort(arr, max):
    m = max + 1
    counter = [0] * m
    for i in arr:
     counter[i] += 1
    a = 0

    for i in range(m):
        for k in range(counter[i]):
            arr[a] = i
            a += 1
    return arr
```

Radix sort:

```
def radix_sort(random_list):
    len_random_list = len(random_list)
    modulus = 10
    div = 1
    while True:
        # empty array, [[] for i in range(10)]
        new_list = [[], [], [], [], [], [], [], [], [], []]
        for value in random_list:
            least_digit = value % modulus
            least_digit /= div
            new_list[least_digit].append(value)
        modulus = modulus * 10
        div = div * 10

        if len(new_list[0]) == len_random_list:
            return new_list[0]

        random_list = []
        rd_list_append = random_list.append
        for x in new_list:
            for y in x:
                rd_list_append(y)
```

- CONCLUSIONES

- Los métodos de ordenamiento de datos son muy útiles.
- Nos facilitan las búsquedas de cantidades de registros.
- Podemos colocar listas detrás de otras y ordenarlas.

- COMPUTADORES USADOS

Hp
Procesador: Intel Core I5 – 2.4 GHz cuadcore
Sistemas operativos: Windows 10 X64
Memoria RAM: 4GB

Getwey
Procesador: AMD Vision – 2.0 GHz
Sistemas operativos: Windows 7 X64
Memoria RAM: 4GB

- RECOMENDACIONES

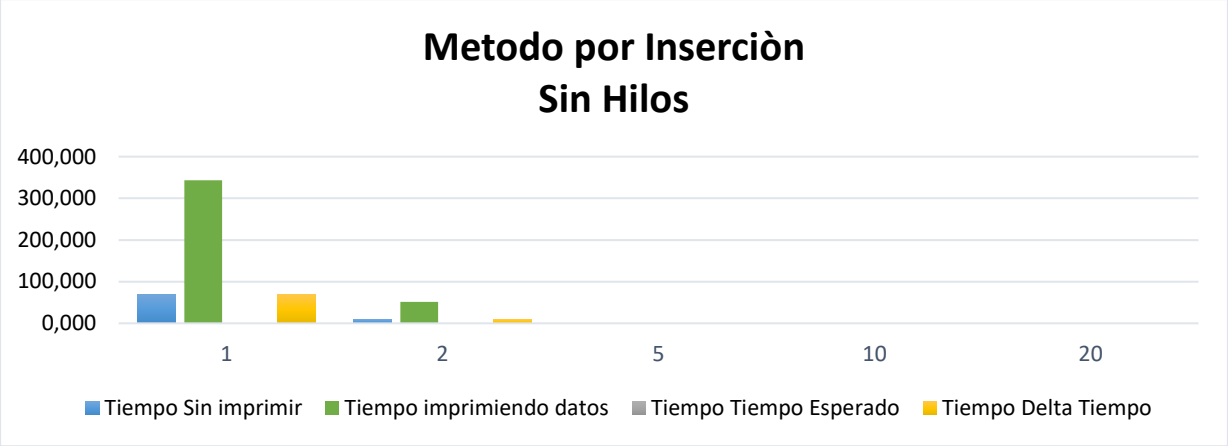
Usar equipos de buenas características, en cuanto a velocidad de procesamiento y memoria RAM.

- REFERENCIAS BIBLIOGRÁFICAS

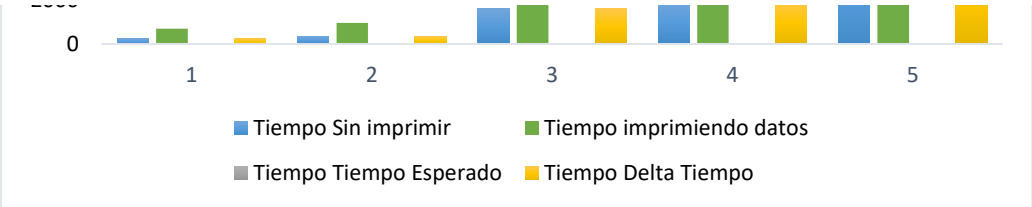
http://ict.udlap.mx/people/ingrid/Clases/IS211/Ordenar.htmlJornadas de Ingeniería Telemática, vol. 1, n. 1, pp. 1-5, 2013.

https://es.slideshare.net/jaironitsed/metodos-de-ordenacion-ordenamiento-y-busqueda-algoritmos

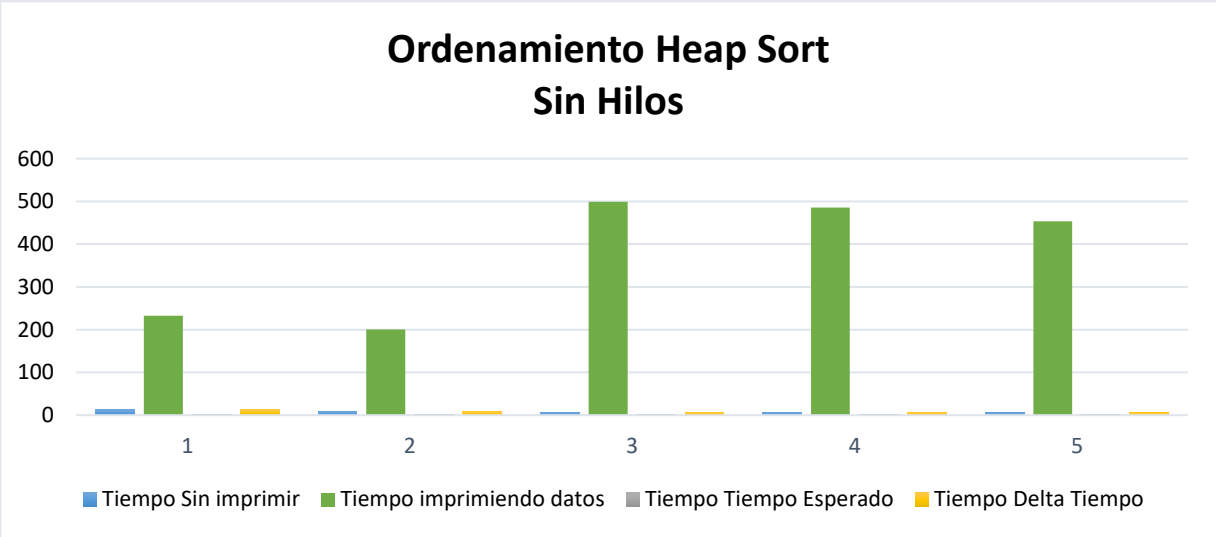
Metodo de Ordenamientopor Inserción				
Complejidad = O(n2) - inserciòn				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	68,734	343,67	1,73611E-07	68,734
2	10,243	51,215	6,94444E-07	10,243
5	SIN RESULTADO	SIN RESULTADO	4,34028E-06	#¡VALOR!
10	SIN RESULTADO	SIN RESULTADO	1,73611E-05	#¡VALOR!
20	SIN RESULTADO	SIN RESULTADO	6,94444E-05	#¡VALOR!



Metodo de Ordenamiento por Mezcla				
Complejidad = O(n log n) - Mezcla				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	260,434	704,333	0,0025	260,4315
2	354,388	964,333	0,005250858	354,3827491
5	1623,544	2900,44	0,013956188	1623,530044
10	1982,44	4455,444	0,029166667	1982,410833
20	4081,333	8549,332	0,060841917	4081,272158

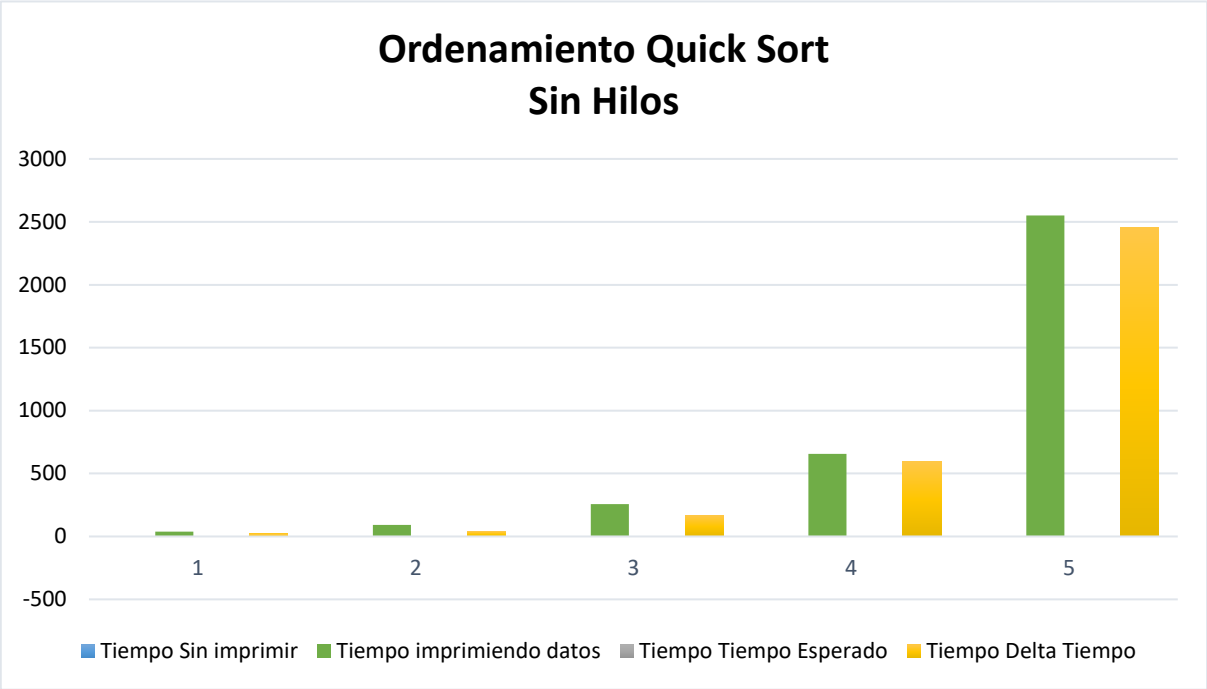


Metodo de Ordenamiento Heap Sort				
Complejidad = O(n log n) - Selecciòn				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	12,344	232,323	0,0025	12,3415
2	8,333	200,323	0,005250858	8,327749142
5	6,232	499,322	0,013956188	6,218043812
10	6,434	485,421	0,029166667	6,404833333
20	5,444	453,434	0,060841917	5,383158083

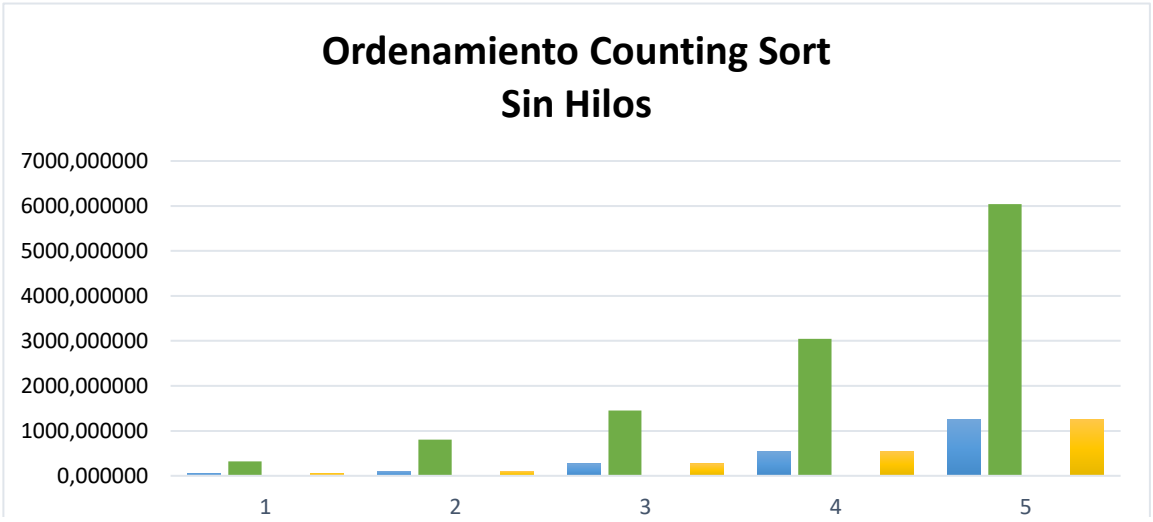


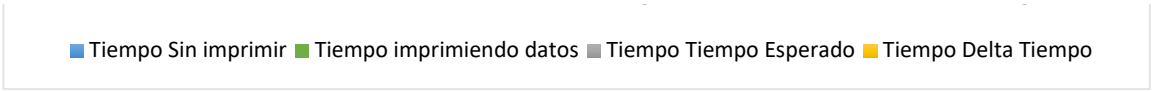
Metodo de Ordenamiento Quick Sort				
Complejidad = O(n * log2(n)) - No estable				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	21,66	38,79	-0,001408421	21,65999983
2	43,37	91,99	-0,002565984	43,36999931
5	168,77	257,457	-0,005585919	168,7699957
10	599,65	655,99	-0,009917547	599,6499826

20	2457,889	2549,98	-0,01732651	2457,888931

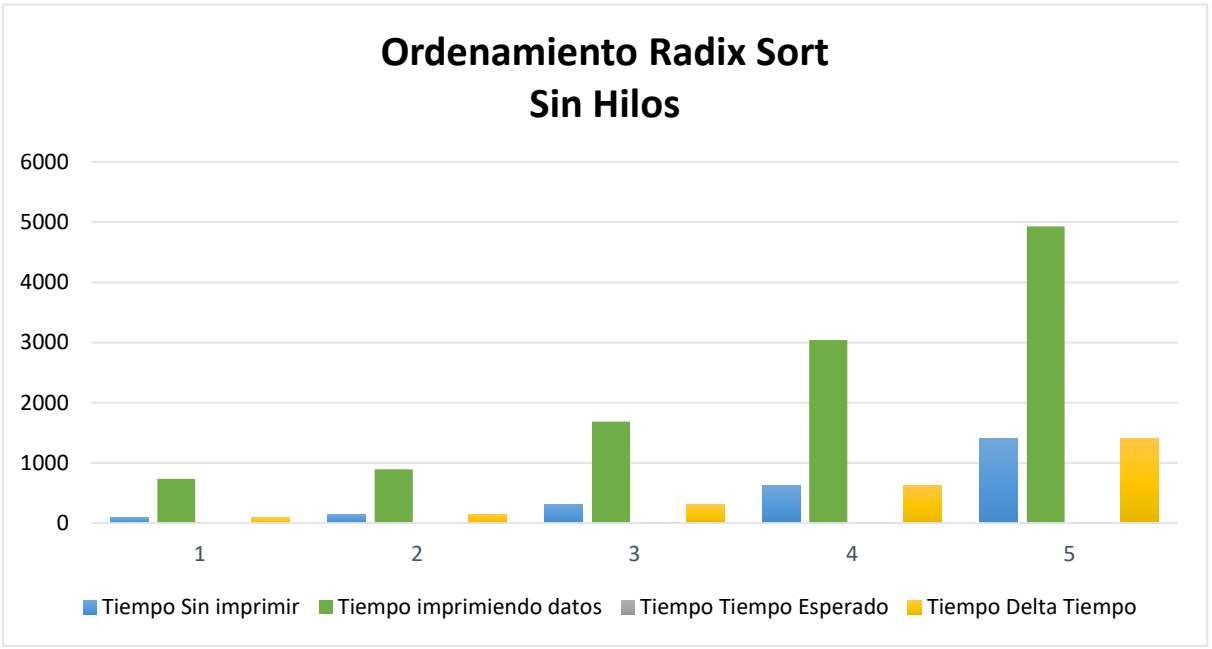


Metodo de Ordenamiento Counting Sort				
Complejidad = O(n+k) - No Compartivo				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	48,159000	320,19	0,000416667	48,158583
2	105,443	805,533	0,000833333	105,442167
5	267,735	1453,233	0,002083333	267,732917
10	542,2	3045,433	0,004166667	542,195833
20	1250,8877	6043,665	0,008333333	1250,879367

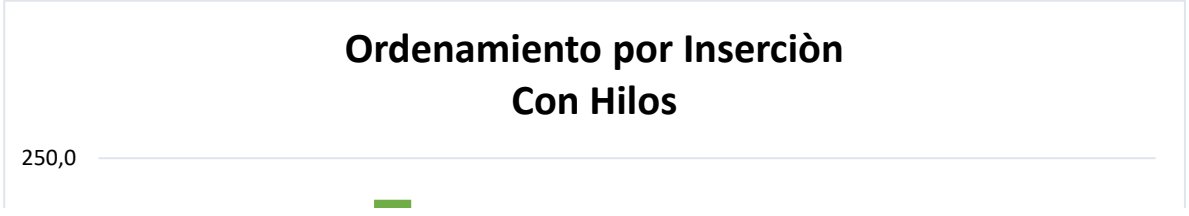


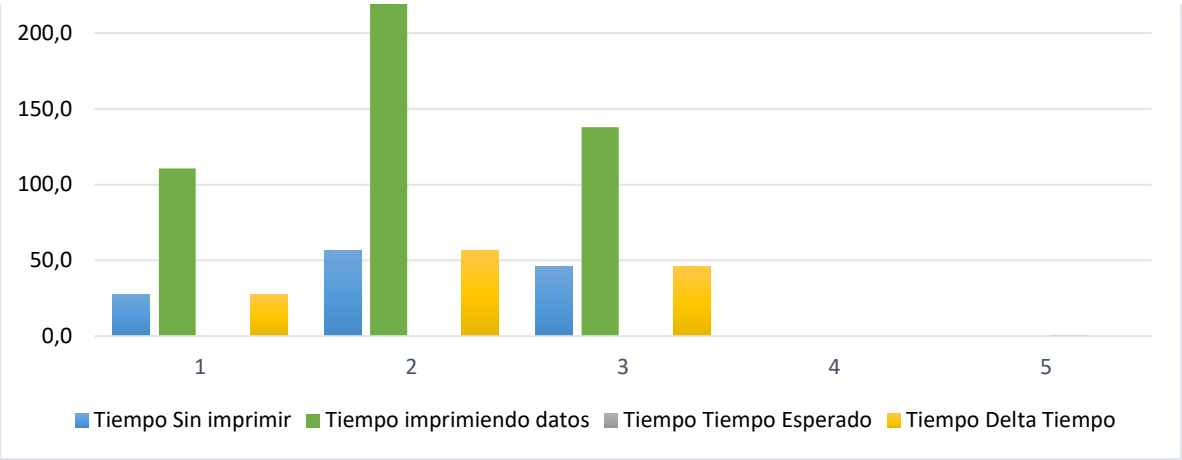


Metodo de Ordenamiento Radix Sort				
Complejidad = O(nk) - No Comparativo				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	89,424	734,545	0,000416667	89,42358333
2	146,43	894,54	0,000833333	146,4291667
5	304,434	1685,43	0,002083333	304,4319167
10	618,434	3043,43	0,004166667	618,4298333
20	1405,3	4928,865	0,008333333	1405,291667

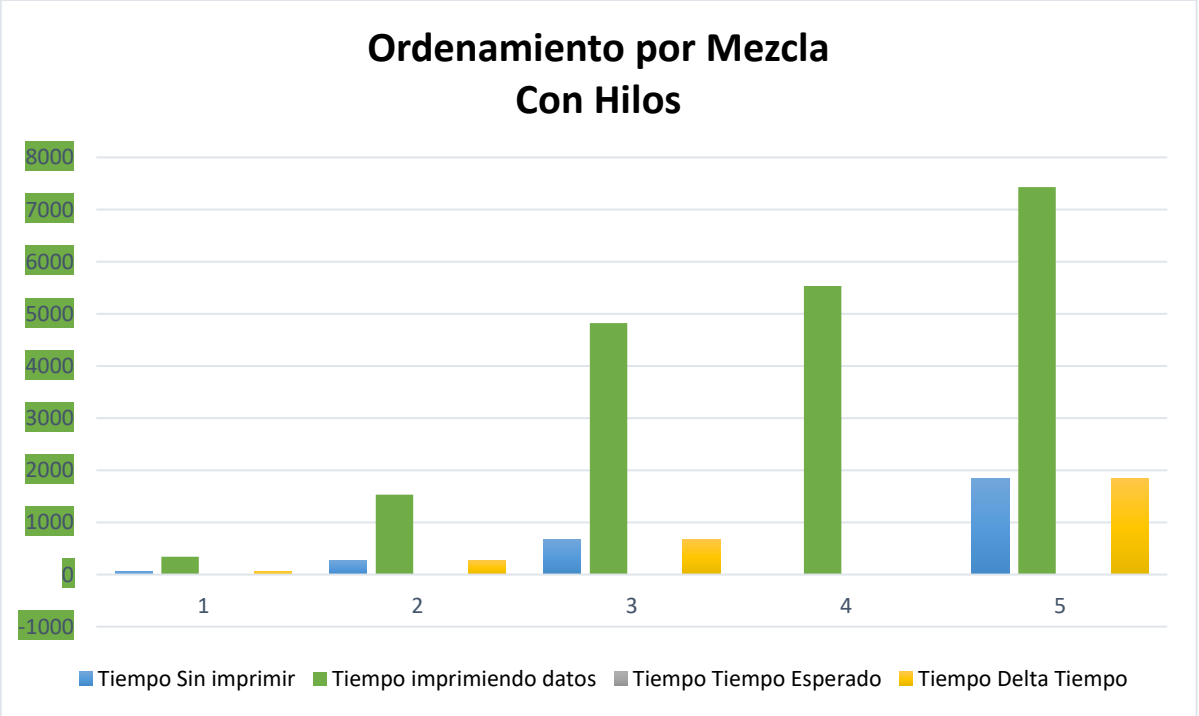


Metodo de Ordenamiento por inserción				
Complejidad = Con Hilos				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	27,7	110,8	1,7E-07	27,6879998
2	56,4	222,5	6,9E-07	56,3759993
5	45,8	137,8	4,3E-06	45,7889957
10	SIN RESULTADO	SIN RESULTADO	1,7E-05	#¡VALOR!
20	SIN RESULTADO	SIN RESULTADO	6,9E-05	#¡VALOR!



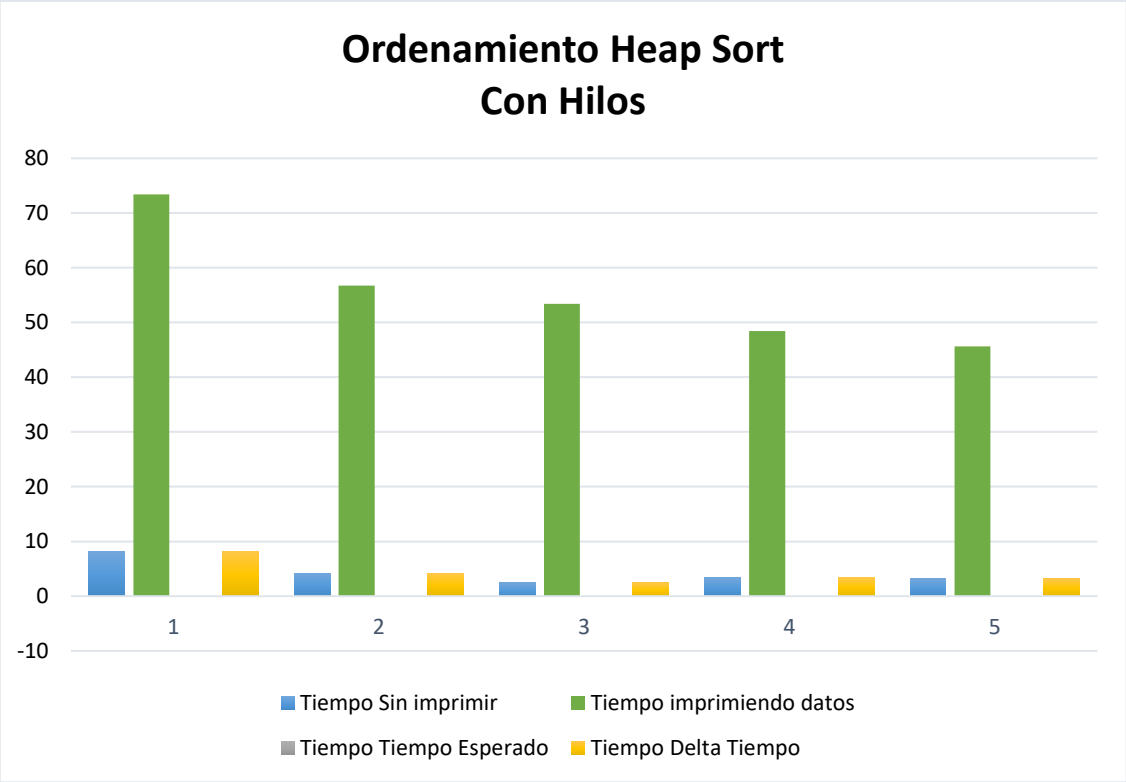


Metodo de Ordenamiento por Mezcla				
Complejidad = Con Hilos				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	52,44	342,433	-0,001408421	52,44140842
2	265,88	1534,55	-0,002565984	265,882566
5	680,44	4824,55	-0,005585919	680,4455859
10	1320.33	5532,334	-0,009917547	#¡VALOR!
20	1840,33	7433,99	-0,01732651	1840,347327

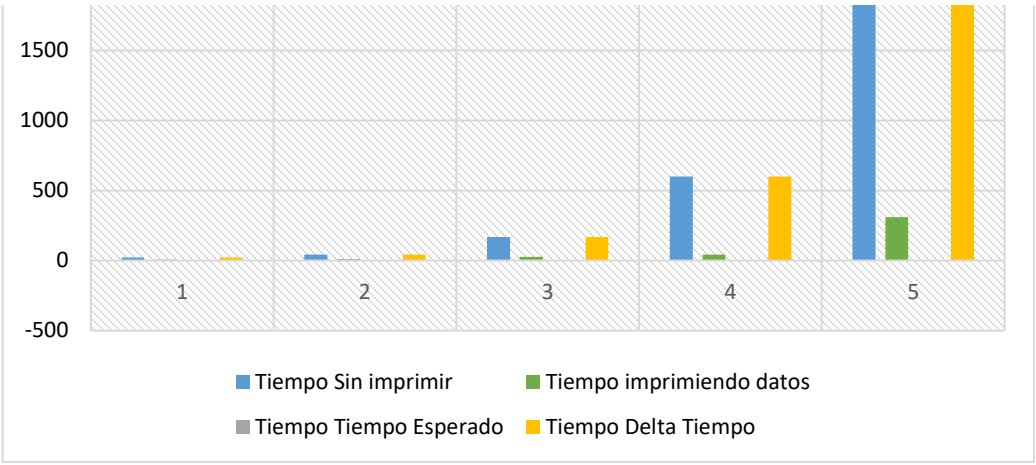


Metodo de Ordenamiento Heap Sort				
Complejidad = Con Hilos				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo

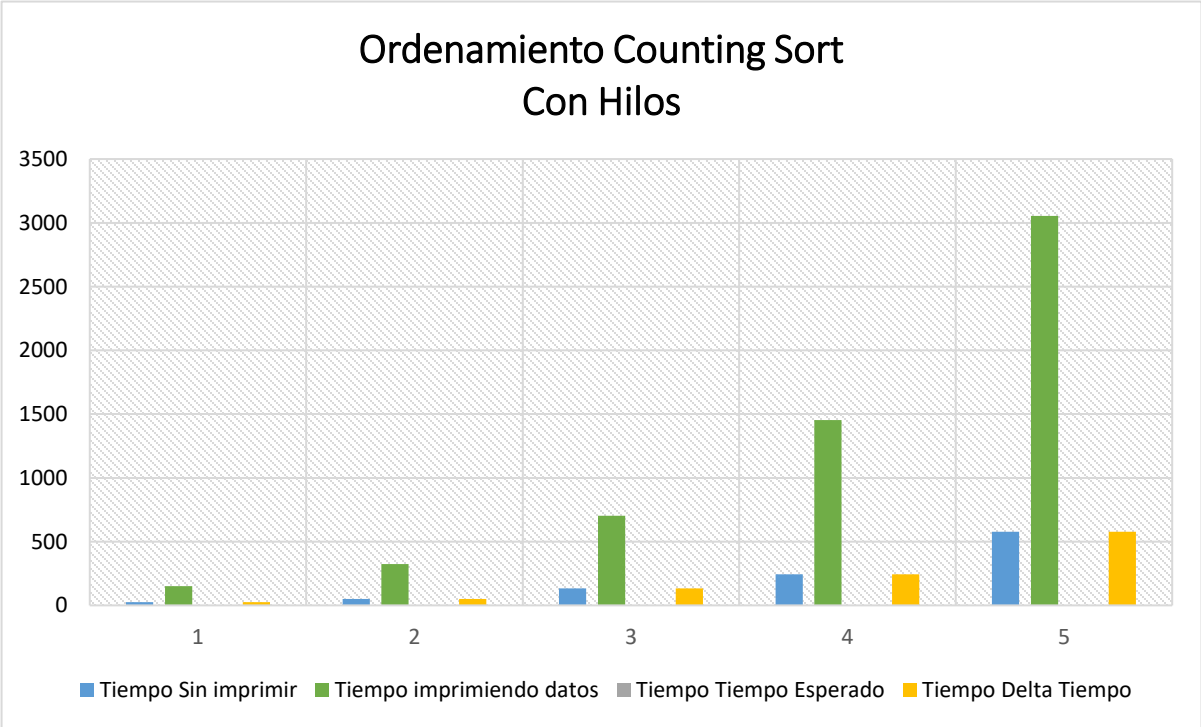
1	8,1666	73,4343	-0,001408421	8,168008421
2	4,212	56,76	-0,002565984	4,214565984
5	2,455	53,434	-0,005585919	2,460585919
10	3,423	48,438	-0,009917547	3,432917547
20	3,212	45,655	-0,01732651	3,22932651



Metodo de Ordenamiento Quick Sort				
Complejidad = Con Hilos				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	6,55	6,55	-0,004678674	21,66467867
2	9,77	9,77	-0,008524016	43,37852402
5	27	27	-0,018556022	168,788556
10	42,55	42,55	-0,032945377	599,6829454
20	309,456	309,456	-0,057557422	2457,946557



Metodo de Ordenamiento Counting Sort				
Complejidad = Con Hilos				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	25,21	150,3	0,000416667	25,20958333
2	50,299	323,899	0,000833333	50,29816667
5	132,07	703,344	0,002083333	132,0679167
10	244,388	1453,355	0,004166667	244,3838333
20	578,332	3054,09	0,008333333	578,3236667



Metodo de Ordenamiento Radix Sort

Complejidad = Con Hilos				
Número de datos	Tiempo			
Millones	Sin imprimir	imprimiendo datos	Tiempo Esperado	Delta Tiempo
1	46,434	342,434	0,000416667	46,43358333
2	70,424	634,545	0,000833333	70,42316667
5	130,434	1453,54	0,002083333	130,4319167
10	268,434	2041,54	0,004166667	268,4298333
20	583,323	3031,54	0,008333333	583,3146667