IMPLEMENTACIÓN DE LOS MÉTODOS DE ORDENAMIENTO EN PYTHON

MARCELA MURILLO MEJIA
marcemejia127@hotmail.com
FERNANDO JAVIER REBELLON HURTADO
guty-15@hotmail.com

CORPORACIÓN DE ESTUDIOS TECNOLÓGICOS DEL NORTE DEL VALLE

Resumen: Principalmente en este documento se hablara sobre los métodos de ordenamiento que se va a implementar en el lenguaje Python. Estos métodos de ordenamiento se dicen que son operaciones de arreglos de los registros de una tabla en algún orden de acuerdo al criterio de ordenamiento, ordenar un grupo de datos significa mover los datos o referencias para que queden en una secuencia que representa un orden. Los métodos de ordenamiento que se van a hablar en este artículo son: por inserción, por mezcla, por heap sort, por quicksort, por counting sort y por radix sort.

Palabras claves: python, métodos de ordenamiento, datos, orden, grupos, inserción, mezcla, heap sort.

Abstract: This document mainly discusses the methods of ordering to be implemented in the Python language. These sorting methods are said to be arranging the records of a table in some order according to the sort order, sorting a data group means moving the data or references so that they remain in a sequence representing an order. The methods of ordering this is a topic in this child article: by insertion, by blending, by heap type, by quicksort, by count and by radix type.

Keywords: python, ordering methods, data, order, groups, insertion, mixing, heap sort.

Introducción: Ordenar es simplemente colocar la información de una manera específica basándose en un criterio de ordenamiento. El propósito principal de estos ordenamientos es el de facilitar las búsquedas de los registros del conjunto de datos. Un ordenamiento se conviene usarlo cuándo se requiere hacer una cantidad considerable de búsquedas y es importante el factor tiempo.

En este caso, nos servirán para ordenar listas con valores asignados aleatoriamente. Nos centraremos en los

métodos más populares, presentando el código escrito en Python, de cada algoritmo.

Algoritmos Usados:

Ordenamiento por inserción: existen dos clases de ordenamiento por inserción uno es el de inserción directa y el otro es por inserción binaria:

Ordenamiento por inserción directa: Este algoritmo se basa en hacer comparaciones, así que para que realice su trabajo de ordenación son imprescindibles dos cosas: un array o estructura similar de elementos comparables y un criterio claro de comparación, tal que dados dos elementos nos diga si están en orden o no. En cada iteración del ciclo externo los elementos 0 a i forman una lista ordenada.

Ordenamiento por inserción binaria: es una mejora del método de **inserción directa**. La mejora consiste en realizar una búsqueda binaria en lugar de una búsqueda secuencial, para insertar un elemento en la parte izquierda del arreglo, que ya se encuentra ordenado.
El proceso al igual que el de Inserción Directa, se repite desde el 2do hasta el n-ésimo elemento.

Toma su nombre debido a la similitud de ordenamiento de los arboles binarios.

Ordenamiento por mezcla: Esta basado en la técnica de “divide y vencerás “. Primero toma el arreglo original de datos, lo divide en dos partes del mismo tamaño cada una, y lo sigue dividiendo hasta que solo quede un elemento. Cada una de las divisiones se ordena de manera separada y luego se unen para formar el arreglo ya ordenado. Este algoritmo divide inicialmente la lista hasta su mínimo valor y luego ordena el arreglo.

Ordenamiento por montones (Heap Sort): Es un algoritmo de ordenación basado en comparaciones de elementos que utiliza un heap para ordenarlos. También podemos decir que es un algoritmo de ordenación no recursivo, no estable, con complejidad computacional.

Ordenamiento rápido (Quick Sort): El ordenamiento rápido (quick sort en inglés) 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. Esta es la técnica de ordenamiento más rápida conocida. Fue desarrollada por C. Antony R. Hoare en 1960. El algoritmo original es recursivo, pero se utilizan versiones

iterativas para mejorar su rendimiento (los algoritmos recursivos son en general más lentos que los iterativos, y consumen más recursos).

Ordenamiento por conteo (counting sort): este método utiliza un arreglo auxiliar para contabilizar el número de llaves que son mayores que la llave actual. El arreglo de contadores, especifica la posición final donde debería estar cada elemento.

Ordenamiento por Radix Sort: Es un 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; Sin embargo radix sort no está limitado sólo a los enteros. Se clasifica en: Digito menos significativo (LSD) O Digito significativo (MSD).

Lenguaje de programación: Python es un lenguaje de programación poderoso y 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. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python también puede usarse como un lenguaje de extensiones para aplicaciones personalizables.

Código de los algoritmos:

Método de Inserción Directa:

```
def insercionDirecta(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
```

Método de Inserción Binaria:

```
def insercionBinaria(lista, tam):
    for i in range(1, tam):
        aux = lista[i]
        izq = 0
        der = i - 1
        while izq <= der:
            m = (izq + der) / 2
            if aux < lista[m]:
                der = m - 1
            else:
                izq = m + 1
        j = i - 1
        while j >= izq:
            lista[j + 1] = lista[j]
```

```
        j = j - 1
    lista[izq] = aux
```

Método de mezcla (mergesort):

```
def mergeSort(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
```

Metodo por montones (heapsort)

```
def heapsort(aList):
   length = len(aList) - 1
   leastParent = length / 2
   for i in range(leastParent, -1, -1):
      moveDown(aList, i, length)
   for i in range(length, 0, -1):
      if aList[0] > aList[i]:
         swap(aList, 0, i)
         moveDown(aList, 0, i - 1)
```

Método por Quick Sort:

```
def quickSort(alist):
   quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
  if first<last:

      splitpoint = partition(alist,first,last)

      quickSortHelper(alist,first,splitpoint-
1)

quickSortHelper(alist,splitpoint+1,last)
```

Método de counting sort:

```
def counting_sort(array, maxval):
   n = len(array)
   m = maxval + 1
   count = [0] * m
   for a in array:
```

```
        count[a] += 1
    i = 0
    for a in range(m):
        for c in range(count[a]):
            array[i] = a
            i += 1
    return array
```

Método por Radix Sort:

```
def radix_sort(random_list):
    len_random_list = len(random_list)
    modulus = 10
    div = 1
    while True:
        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)
```

Conclusión: Los métodos de ordenamiento de datos son muy útiles, ya que se puede ordenar los registros de una tabla en algún orden secuencial de acuerdo al método que se utilice, el cual puede ser numérico, alfabético o alfanumérico, ascendente o descendente. Facilita las búsquedas de cantidades de registros en un moderado tiempo, en modelo de eficiencia. Mediante sus técnicas podemos colocar listas detrás de otras y luego ordenarlas, como también se puede comparar los valores, e intercambiarlos si no se encuentran en sus posiciones correctas.

Recomendaciones:

- Para hacer este tipo de trabajos de comparar los métodos con hilos y sin hilos se necesita un computador con un procesador de gama alta.
- Es tener un conocimiento básico de los métodos de ordenamiento y de los hilos en el lenguaje de programación en python.
- Tener en cuenta que python tiene limitada la recursividad a cien

datos, implementar comando para aumentar dicha recursividad

Referentes bibliográficos:

- https://saforas.wordpress.com/2011/01/24/metodos-de-ordenamiento-hechos-en-python/.
- https://blog.zerial.org/ficheros/Informe_Ordenamiento.pdf.
- http://progpython.blogspot.com.co/2011/09/algoritmos-de-ordenamiento-en-python-el.html
- http://www.codegaia.com/index.php/Complejidad_Algoritmos_de_Ordenamiento

Computador Usado:

Fabricante Dell

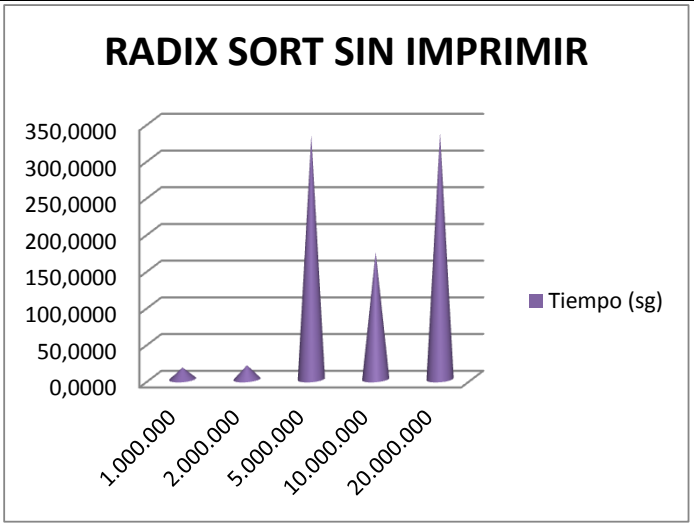
Modelo Optiplex 3040

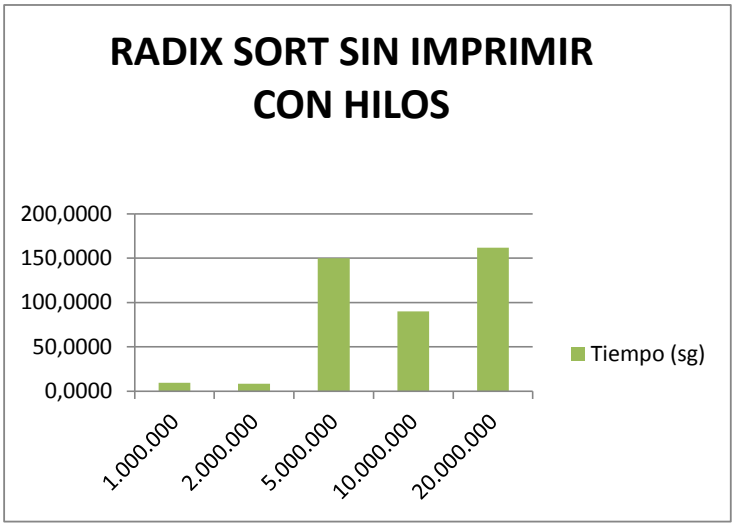
Procesador Intel(R) COre(TM) i7-6100 CPU @4.70GHz 4.70 GHz

Memoria (RAM) 8,00 GB

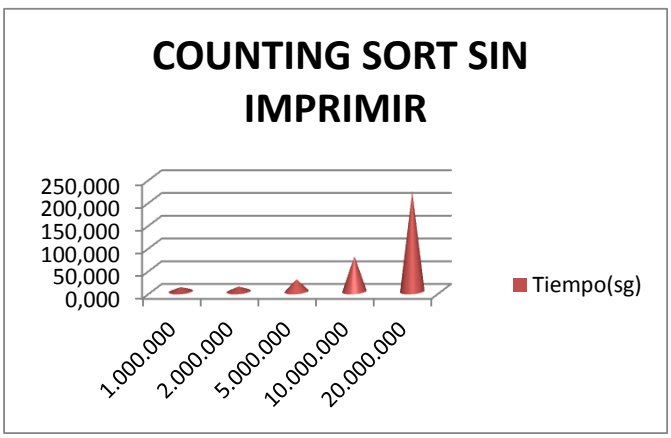
Tipo Sistema Sistema Operativo de 64 bits

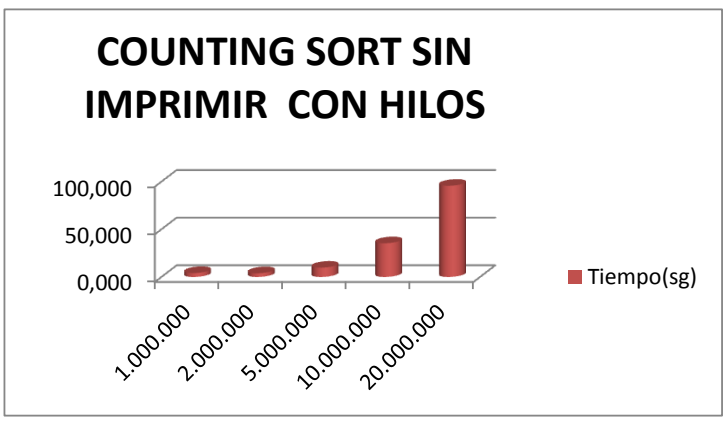
Algoritmo - Radix Sort							
Complejidad = O(nk)							
Numero de Datos	Tiempo en Segundos						
Millones	Sin Imprimir	Sin Imprimir con Hilos	Imprimiendo Datos	Imprimiendo Datos con hilos	Tiempo Esperado	Delta Tiempo	Delta Tiempo con Hilos
1.000.000	15,8180	9,4220	41,309	21,575	0,000213	15,8	9,4
2.000.000	18,9700	8,5020	90,997	57,841	0,000426	19,0	8,5
5.000.000	332,8930	149,9520	189,532	104,148	0,001064	332,9	150,0
10.000.000	173,3790	89,9020	778,694	598,617	0,002128	173,4	89,9
20.000.000	335,9060	161,7120	974,919	569,855	0,004255	335,9	161,7



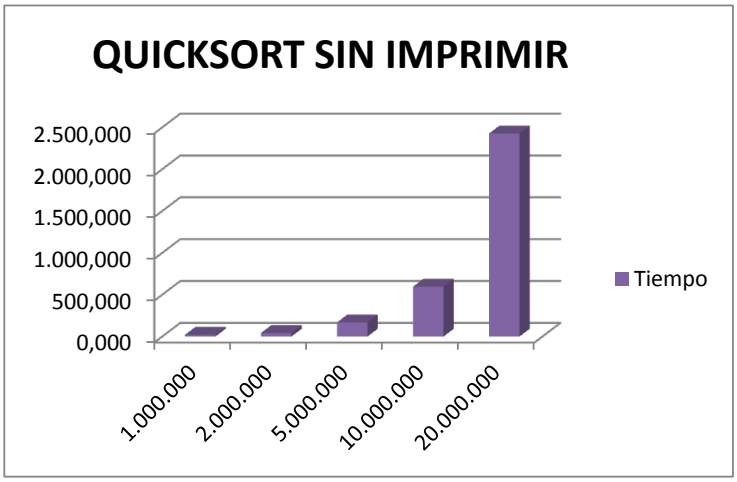


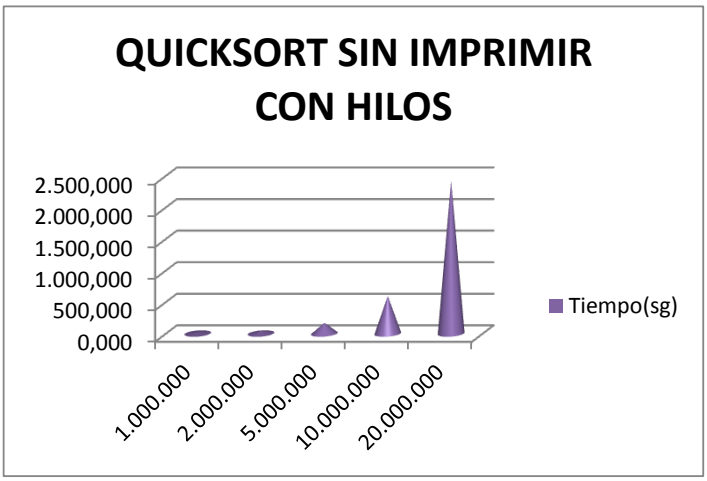
Algoritmo - Couting Sort							
Complejidad = n+k							
Numero de Datos	Tiempo en Segundos						
Millones	Sin Imprimir	Sin Imprimir con Hilos	Imprimiendo Datos	Imprimiendo Datos con hilos	Tiempo Esperado	Delta Tiempo	Delta Tiempo con Hilos
1.000.000	9,547	4,134	158,754	138,911	0,000213	9,55	4,13
2.000.000	10,936	3,916	165,784	133,850	0,000426	10,94	3,92
5.000.000	26,736	9,765	370,326	291,551	0,001064	26,73	9,76
10.000.000	77,855	35,397	396,159	224,090	0,002128	77,85	35,39
20.000.000	220,083	95,689	980,348	646,848	0,004255	220,08	95,68



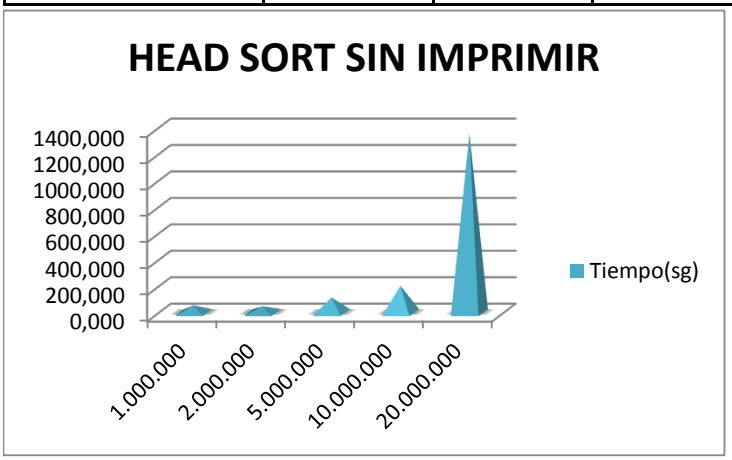


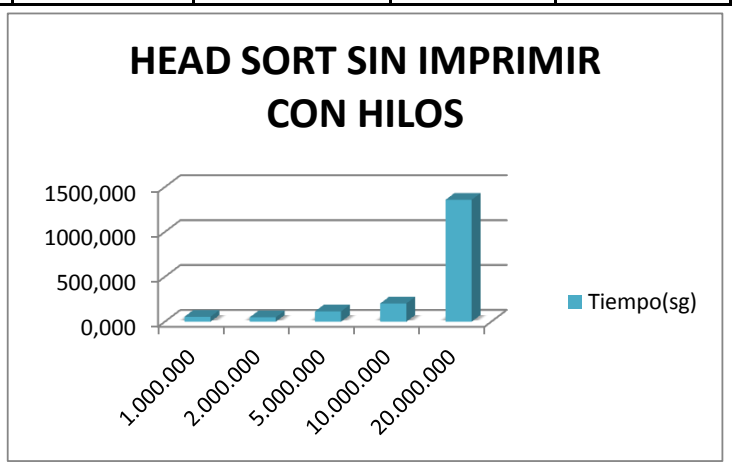
Algoritmo - Rapido (Quicksort)							
Complejidad = Nlog n							
Numero de Datos	Tiempo en Segundos						
Millones	Sin Imprimir	Sin Imprimir con Hilos	Imprimiendo Datos	Imprimiendo Datos con hilos	Tiempo Esperado	Delta Tiempo	Delta Tiempo con Hilos
1.000.000	21,653	6,437	37,777	14,694	0,000284	21,65	6,44
2.000.000	43,273	9,150	91,987	30,688	0,000696	43,27	9,15
5.000.000	169,871	25,904	256,375	71,889	0,002372	169,87	25,90
10.000.000	596,448	42,478	650,332	126,384	0,005905	596,44	42,47
20.000.000	2.427,853	309,832	2.538,068	359,501	0,014405	2.427,84	309,82



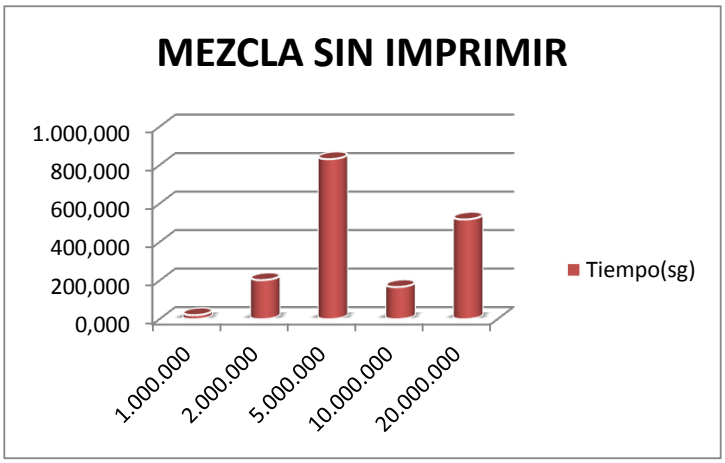


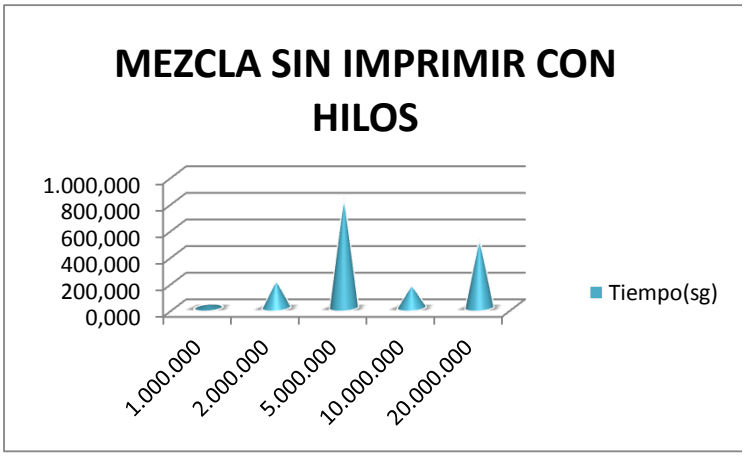
Algoritmo - Montones (Head Sort)							
Complejidad = N log n							
Numero de Datos	Tiempo en Segundos						
Millones	Sin Imprimir	Sin Imprimir con Hilos	Imprimiendo Datos	Imprimiendo Datos con hilos	Tiempo Esperado	Delta Tiempo	Delta Tiempo con Hilos
1.000.000	54,890	10,567	216,33	191,44	0,000494	54,890	10,567
2.000.000	48,375	13,931	310,35	263,49	0,00117	48,4	13,93
5.000.000	114,770	12,449	247,96	116,97	0,00489	114,8	12,44
10.000.000	203,423	29,680	410,61	133,18	0,01363	203,4	29,67
20.000.000	1358,086	940,716	1.066,71	425,17	0,04513	1358,0	940,67





Algoritmo - Mezcla (mergesort)							
Complejidad = N log n							
Numero de Datos	Tiempo en Segundos						
Millones	Sin Imprimir	Sin Imprimir con Hilos	Imprimiendo Datos	Imprimiendo Datos con hilos	Tiempo Esperado	Delta Tiempo	Delta Tiempo con Hilos
1.000.000	21,263	7,519	241,43	214,50	0,00028	21,263	7,51872
2.000.000	203,695	27,069	274,84	229,15	0,00098	203,694	27,06802
5.000.000	832,405	219,492	182,77	59,39	0,00311	832,402	219,48889
10.000.000	167,405	31,825	476,29	215,43	0,00473	167,400	31,82027
20.000.000	518,830	150,451	1.054,94	399,25	0,01155	518,818	150,43945





Algoritmo - Insercion (insertsort)							
Complejidad = N^2							
Numero de Datos	Tiempo en Segundos						
Millones	Sin Imprimir	Sin Imprimir con Hilos	Imprimiendo Datos	Imprimiendo Datos con hilos	Tiempo Esperado	Delta Tiempo	Delta Tiempo con Hilos
1.000.000	483,900	268,900	725,85	403,35	0,000000045269	483,900	268,90000
2.000.000	967,800	752,800	1.451,70	1.129,20	0,000000181077	967,800	752,80000
5.000.000	2.419,500	2.204,500	3.629,25	3.306,75	0,000001131734	2.419,500	2.204,50000
10.000.000	4.839,000	4.624,000	7.258,50	6.936,00	0,000004526935	4.839,000	4.624,00000
20.000.000	9.678,000	9.463,000	14.517,00	14.194,50	0,000018107741	9.678,000	9.462,99998