MÉTODOS DE ORDENAMIENTO

HERMAN COLLAZOS CASTAÑEDA hcollazos87@gmail.com

MAURICIO GRANADA mauro-130@hotmail.com

PROGRAMACIÓN DISTRIBUIDA

CARLOS ALBERTO LONDOÑO

INGENIERO DE SISTEMAS

ORPORACIÒN DE ESTUDIOS TECNOLOGICOS DEL NORTE DEL VALLE

INGENIERIA DE SISTEMAS

CARTAGO VALLE DEL CAUCA

10/05/2019

RESUMEN

El método de ordenamiento permite la organización de una serie de datos ya sea numéricos o alfanuméricos facilitando así su análisis y toma de decisiones dentro de una organización, existe varios métodos de ordenamiento la cual son utilizados dependiendo de la necesidad y volumen de los datos. El anterior ejercicio nos permite hacer una comparación de los tiempos en que cada método de ordenamiento realiza su proceso y tiempo en el que se demora en realizar la tarea

PALABRAS CLAVES

Datos numéricos, datos alfanuméricos, Métodos de ordenamiento

ABSTRACT

The sorting methods allow the organization of a series of numerical or alphanumeric data facilitating their analysis and decision making within an organization, there are several sorting methods which are used depending on the need and volumen of the data. The previous exercise allows us to make a comparison of the times in which each ordering method performs its process and time in which it is delayed in performing the task

KEYWORDS

Numerical data, alphanumeric data, Sorting methods

INTRODUCCION

En la actualidad la información es el capital más importante dentro de una organización porque permite la toma de decisiones para actividades futuras es por esto que el ordenar la información es pieza clave para la agilización del proceso de análisis dentro de la empresa. El ordenar es simplemente colocar información en un criterio de ordenamiento, y su propósito principal es el de facilitar la búsqueda de los registros en un conjunto de datos es conveniente usarlo cuando se manejan una gran cantidad de datos y en donde el factor tiempo es primordial, teniendo esta necesidad de ordenar los datos se crearon varios métodos de ordenamiento que permiten realizar esta tarea más fácil como son el método de inserción, método de conteo entre otros. Un objetivo que se quiere mostrar dentro del ejercicio es con el método seleccionados como son el método de inserción, método de conteo, método de mezcla, método de heap sort, método quicksort, método radix sort mostrar su funcionamiento, eficiencia, facilidad y tiempo que demora en mostrar resultados En los códigos fuentes de los métodos de ordenamiento se implementarán el manejo de hilos que permitirá agilizar los procesos de ejecución de los métodos esto con el fin de crear cuadros comparativos realizando los procesos con hilos y sin hilos permitiéndonos analizar la eficiencia de cada método con y sin hilos

METODOLOGÍA UTILIZADA

se utiliza la metodología aplicada para la comparación de la evolución y respuesta de los métodos de ordenamiento en sus diferentes escenarios mediante ordenamiento de datos de tamaño moderado.

ALGORITMOS UTILIZADOS

Los algoritmos implementados son los métodos de ordenamiento con mayor uso y eficiencia operativa:

Ordenamiento por inserción

Ordenamiento por mezcla

Ordenamiento por montones (Heap Sort)

Ordenamiento rápido (QuickSort)

Ordenamiento por conteo (Counting Sort)

Ordenamiento por Radix Sort

LENGUAJE DE PROGRAMACIÓN USADO

El lenguaje de programación seleccionado para la implementación es Python, el cual es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

CÓDIGO DE LOS ALGORITMOS IMPLEMENTADOS

El código presentado a continuación son los códigos de cada uno de los métodos de ordenamientos que se realizó en el lenguaje de programación correspondiente.

MÉTODOS DE ORDENAMIENTO

SIN HILOS

Funciones.py

```
import random
```

```
import csv
def crearLista():
lista = list()
return lista
def nombreArchivo():
nombre = raw_input("Ingrese
nombre del archivo: ")
return nombre
def canRegistros():
cant = int(raw_input("Ingrese
Cantidad de registros a generar:
"))
return cant
def llenarLista(lista, limite):
for i in range(0, limite):
lista.append(random.randint(-
9999, 9999))
def llenarListaConteo(lista,
limite):
for i in range(0, limite):
lista.append(random.randint(0,
1000))
def crearCsv(lista, nombre):
datos = open(nombre + ".csv",
"w")
datos_csv = csv.writer(datos)
datos_csv.writerow(lista)
datos.close()
def imprimirLista(lista):
print lista
def leerCsv(lista_2, nombre):
datos = open(nombre + ".csv",
"r")
datos_csv = csv.reader(datos,
delimiter=",")
for variable in (datos_csv):
lista_2.append(variable)
datos.close()
return lista_2
def arreglalista(lista,nlista):
for i in lista [0]:
nlista.append(int(i))
return nlista
def longitud(lista):
return len(lista)
def imprimirTiempo(tiempo_final):
print ("Proceso finalizado en
% 0.5f segundos" % tiempo_final)
```

inserción.py

```
# * * * * * * * * * * * * * *
* * * * * * * * * * * * * *
```

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
# * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# HeapSort * * * * * * * * * * * * * * * * * * * * * * * *
def swap(lista,i, j):
    lista[i], lista[j] = lista[j], lista[i]
def heapify(lista,end,i):
    l=2 * i + 1
    r=2 * (i + 1)
    max=i
    if l < end and lista[i] < lista[l]:
        max = l
    if r < end and lista[max] < lista[r]:
        max = r
    if max != i:
        swap(lista,i, max)
        heapify(lista,end, max)
def heap_sort(lista):
    end = len(lista)
    start = end // 2 - 1 # use // instead of /
    for i in range(start, -1, -1):
        heapify(lista,end, i)
    for i in range(end-1, 0, -1):
        swap(lista,i, 0)
        heapify(lista,i, 0)
# * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# Quick Sort * * * * * * * * * * * * * * * * * * * * * * *
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)
# * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
Merge Sort * * * * * * * * *
* * * * * * * * * * * * * * *
def merge_sort(numbers):
"""Punto de entrada del
algoritmo"""
n = len(numbers)
if (n == 1): return numbers
left = merge_sort(numbers[:(n
/ 2)])
right = merge_sort(numbers[(n
/ 2):])
return merge(left, right)
def merge(left, right):
result = []
i = 0
j = 0
len_left = len(left)
len_right = len(right)
while (i < len_left or j <
len_right):
if (i >= len_left):
result.append(right[j])
j = j + 1
elif (j >= len_right):
result.append(left[i])
i = i + 1
elif (left[i] <
right[j]):
result.append(left[i])
i = i + 1
else:
result.append(right[j])
j = j + 1
return result
# * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
Ordenamiento por Conteo
* * * * * * * * * * *
* * * * * * * * * * *
def counting_sort(array, maxval):
n = len(array)
m = maxval + 1
count = [0] * m
# init with zeros
```

```
for a in array:
count[a] += 1
# count occurences
i = 0
for a in range(m):
# emit
for c in range(count[a]):
# - emit 'count[a]' copies of 'a'
array[i] = a
i += 1
return array
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
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(valu
e)
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)
Run_insertion.py
# -* - coding: utf-8 -* -
from funciones import *
from insertion import *
from threading import Thread
import time
def main():
op = " "
while op != "s" and op !=
"S":
print"————————-
————————"
```

```
print("\ t\ tMETODOS DE
ORDENAMIENTO\ t\ t")
print("(a). Inserción
Directa ")
print("(b). Ordenamiento
por Mezcla (Merge Sort) ")
print("(c). Ordenamiento
por Montones (Heap Sort) ")
print("(d). Ordenamiento
rápido (Quick Sort) ")
print("(e). Ordenamiento
por Conteo (Counting Sort) ")
print("(f). Ordenamiento
por Radix Sort ")
print("(s). Salir")
op = raw_input("Digite
una Opción: ")
if op == "a" or op ==
"A": # inserción Directa
tiempo_inicial =
time.time()
lista = crearLista()
nombre =
nombreArchivo()
# llenarLista(lista,
1000)
nreg = canRegistros()
llenarLista(lista,
nreg)
crearCsv(lista,
nombre)
imprimirLista(lista)
# ordenar
lista_ordenada =
crearLista()
leerCsv(lista_ordenada, nombre)
nlista = crearLista()
nlista2 =
arreglalista(lista_ordenada,nlist
a)
tam =
longitud(nlista2)
insercionDirecta(nlista2, tam)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "b" or op ==
"B": # Ordenamiento por Mezcla
(Merge Sort)
tiempo_inicial =
time.time()
lista = crearLista()
```

```
nombre =
nombreArchivo()
nreg = canRegistros()
llenarLista(lista,
nreg)
crearCsv(lista,
nombre)
imprimirLista(lista)
lista_ordenada =
crearLista()
leerCsv(lista_ordenada, nombre)
nlista = crearLista()
nlista2 =
arreglalista(lista_ordenada,
nlista)
List2 =
sorted(nlista2, reverse = True)
# print "Lista 1 "
# print nlista2
# print "Lista 2 "
# print List2
merge_sort(nlista2)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "c" or op ==
"C": # Ordenamiento por Montones
(Heap Sort)
tiempo_inicial =
time.time()
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
llenarLista(lista,
nreg)
crearCsv(lista,
nombre)
imprimirLista(lista)
# ordenar
lista_ordenada =
crearLista()
leerCsv(lista_ordenada, nombre)
nlista = crearLista()
nlista2 =
arreglalista(lista_ordenada,nlist
a)
tam =
longitud(nlista2)
heap_sort(nlista2)
imprimirLista(nlista2)
tiempo_final =
```

```
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "d" or op ==
"D": # Ordenamiento rápido (Quick
Sort)
tiempo_inicial =
time.time()
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
llenarListaConteo(lista, nreg)
crearCsv(lista,
nombre)
imprimirLista(lista)
# ordenar
lista_ordenada =
crearLista()
leerCsv(lista_ordenada, nombre)
nlista = crearLista()
nlista2 =
arreglalista(lista_ordenada,nlist
a)
tam =
longitud(nlista2)
quicksort(nlista2, 0,
tam - 1)
# quicksort(nlista2,
tam)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "e" or op ==
"E": # Ordenamiento rápido (Quick
Sort)
tiempo_inicial =
time.time()
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
llenarLista(lista,
nreg)
crearCsv(lista,
nombre)
imprimirLista(lista)
# ordenar
lista_ordenada =
crearLista()
leerCsv(lista_ordenada, nombre)
nlista = crearLista()
nlista2 =
```

```
arreglalista(lista_ordenada,nlist
a)
tam =
longitud(nlista2)
# print len(nlista2)
counting_sort(nlista2,
len(nlista2))
# counting_sort(nlista2, 7)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "f" or op ==
"F": # Ordenamiento Radix Sort
tiempo_inicial =
time.time()
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
llenarLista(lista,
nreg)
crearCsv(lista,
nombre)
imprimirLista(lista)
# ordenar
lista_ordenada =
crearLista()
leerCsv(lista_ordenada, nombre)
nlista = crearLista()
nlista2 =
arreglalista(lista_ordenada,nlist
a)
radix_sort(nlista2)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "S" or op ==
"s":
print "Hasta Pronto."
exit()
else:
print("Digite una
Opción \ n")
if __name__ == '__main__':
main()
```

METODOS DE ORDENAMIENTOS CON HILOS

Run_insertion.py

```
# -* - coding: utf-8 -* -
from funciones import *
```

```
from insertion import *
import threading
import time
def main():
op = " "
while op != "s" and op !=
"S":
print"————————-
————————"
print("\ t\ tMETODOS DE
ORDENAMIENTO\ t\ t")
print("(a). Inserción
Directa ")
print("(b). Ordenamiento
por Mezcla (Merge Sort) ")
print("(c). Ordenamiento
por Montones (Heap Sort) ")
print("(d). Ordenamiento
rápido (Quick Sort) ")
print("(e). Ordenamiento
por Conteo (Counting Sort) ")
print("(f). Ordenamiento
por Radix Sort ")
print("(s). Salir")
op = raw_input("Digite
una Opción: ")
if op == "a" or op ==
"A": # inserción Directa
# tiempo_inicial =
time.time()
lista = crearLista()
nombre =
nombreArchivo()
nreg=canRegistros()
hilo =
threading.Thread(target=llenarLis
ta, args=(lista, nreg,))
hilo.start()
hilo2 =
threading.Thread(target=crearCsv,
args=(lista, nombre,))
hilo2.start()
tiempo_inicial =
time.time()
imprimirLista(lista)
lista_ordenada =
crearLista()
nlista = crearLista()
hilo3 =
threading.Thread(target=leerCsv,
args=(lista_ordenada, nombre,))
hilo3.start()
hilo3.join()
```

```
nlista2 =
arreglalista(lista_ordenada,
nlista)
tam =
longitud(nlista2)
insercionDirecta(nlista2, tam)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
nombre_final =
nombreArchivo()
hilo2 =
threading.Thread(target=crearCsv,
args=(nlista2, nombre_final,))
hilo2.start()
if op == "b" or op ==
"B": # Ordenamiento por Mezcla
(Merge Sort)
tiempo_inicial =
time.time()
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
hilo =
threading.Thread(target=llenarLis
ta, args=(lista, nreg,))
hilo.start()
hilo2 =
threading.Thread(target=crearCsv,
args=(lista, nombre,))
hilo2.start()
tiempo_inicial =
time.time()
imprimirLista(lista)
lista_ordenada =
crearLista()
leerCsv(lista_ordenada, nombre)
nlista = crearLista()
nlista2 =
arreglalista(lista_ordenada,
nlista)
imprimirLista(mergesort(nlista2))
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
nombre_final =
nombreArchivo()
hilo2 =
threading.Thread(target=crearCsv,
args=(mergesort(nlista2),
nombre_final,))
```

```
hilo2.start()
if op == "c" or op ==
"C": # Ordenamiento por Montones
(Heap Sort)
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
hilo =
threading.Thread(target=llenarLis
ta, args=(lista, nreg,))
hilo.start()
hilo2 =
threading.Thread(target=crearCsv,
args=(lista, nombre,))
hilo2.start()
tiempo_inicial =
time.time()
imprimirLista(lista)
lista_ordenada =
crearLista()
nlista = crearLista()
hilo3 =
threading.Thread(target=leerCsv,
args=(lista_ordenada, nombre,))
hilo3.start()
hilo3.join()
nlista2 =
arreglalista(lista_ordenada,nlist
a)
tam =
longitud(nlista2)
heap_sort(nlista2)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "d" or op ==
"D": # Ordenamiento rápido (Quick
Sort)
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
hilo =
threading.Thread(target=llenarLis
ta, args=(lista, nreg,))
hilo.start()
hilo2 =
threading.Thread(target=crearCsv,
args=(lista, nombre,))
hilo2.start()
tiempo_inicial =
```

```
time.time()
imprimirLista(lista)
lista_ordenada =
crearLista()
nlista = crearLista()
hilo3 =
threading.Thread(target=leerCsv,
args=(lista_ordenada, nombre,))
hilo3.start()
hilo3.join()
nlista2 =
arreglalista(lista_ordenada,
nlista)
tam =
longitud(nlista2)
quicksort(nlista2, 0,
tam - 1)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "e" or op ==
"E": # Ordenamiento rápido (Quick
Sort)
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
hiloe =
threading.Thread(target=llenarLis
taConteo, args=(lista, nreg,))
hiloe.start()
hiloe2 =
threading.Thread(target=crearCsv,
args=(lista, nombre,))
hiloe2.start()
tiempo_inicial =
time.time()
imprimirLista(lista)
# ordenar
lista_ordenada =
crearLista()
nlista = crearLista()
hiloe3 =
threading.Thread(target=leerCsv,
args=(lista_ordenada, nombre,))
hiloe3.start()
hiloe3.join()
nlista2 =
arreglalista(lista_ordenada,
nlista)
tam =
longitud(nlista2)
```

```
# print len(nlista2)
counting_sort(nlista2, tam)
#
counting_sort(nlista2, 7)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "f" or op ==
"F": # Ordenamiento Radix Sort
lista = crearLista()
nombre =
nombreArchivo()
nreg = canRegistros()
hilof =
threading.Thread(target=llenarLis
taConteo, args=(lista, nreg,))
hilof.start()
hilof2 =
threading.Thread(target=crearCsv,
args=(lista, nombre,))
hilof2.start()
tiempo_inicial =
time.time()
imprimirLista(lista)
# ordenar
lista_ordenada =
crearLista()
nlista = crearLista()
hilof3 =
threading.Thread(target=leerCsv,
args=(lista_ordenada, nombre,))
hilof3.start()
hilof3.join()
nlista2 =
arreglalista(lista_ordenada,nlist
a)
radix_sort(nlista2)
imprimirLista(nlista2)
tiempo_final =
time.time() - tiempo_inicial
imprimirTiempo(tiempo_final)
if op == "S" or op ==
"s":
print ("Hasta Pronto.")
exit()
else:
print("Digite una
Opción \ n")
if __name__ == '__main__':
main()
```

RESULTADOS ALCANZADOS

Uno de los grandes problemas que se presenta actualmente en la organización por la gran cantidad de datos que se manejan y es precisamente el procesamiento de sus datos, es tanta la cantidad que su herramienta de computo colapsaron generando represamientos y Perdida de información indispensable en la toma de decisiones en la organización. Es importante destacar que en la actualidad existen unos métodos de ordenamiento que permiten la búsqueda y ordenamiento de los Datos más fáciles el objetivo es presentar he implementar algunos métodos su funcionamiento e implementar en su código fuente la utilización de hilos para la agilización de sus procesos.

EVALUACION DE LOS RESULTADOS DE LOS ALGORITMOS IMPLEMENTADOS

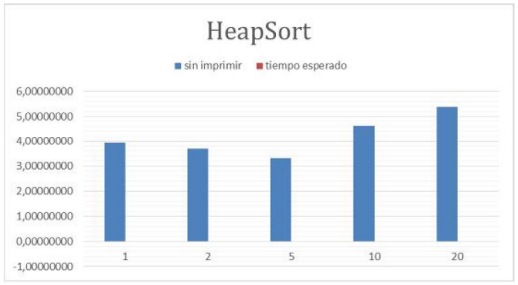
METODO DE INSERCION

Algoritmo Inserción Directa O(n²)						
numero datos	tiempo en minutos					
	Sin Hilos		Con Hilos			
millones	sin imprimir	imprimiendo datos	sin imprimir	imprimiendo datos	tiempo esperado	delta tiempo
1	9,126000	605,36300 s	0.00997 s	4.21528 s	0,00000017361100 0	9
2	4,244000	1300,2563 s	0.04193 s	6.50953 s	0,00000069444444	4
5	4,027000	3748,1203 s	0.23644 s	16.06289 s	0,00000434027778	4
10	5,616000	10022,02214 s	0.09881 s	33.86321 s	0,00001736111111	6
20	6,427000	Error de Memoria	0.02977 s	64.20960 s	0,00006944444444	6

METODO MERGE SORT

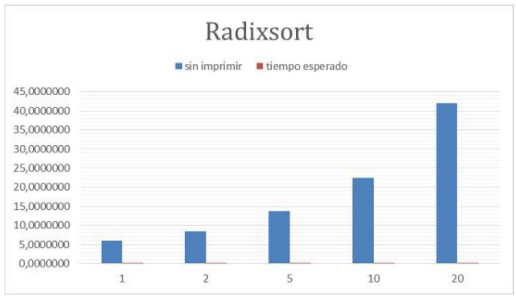
Algoritmo Merge Sort sin hilos (n logn)						
numero datos	tiempo en minutos					
	Sin Hilos		Con Hilos			
millones	sin imprimir	imprimiendo datos	sin imprimir	imprimiendo datos	tiempo esperado	delta tiempo
1	5,602000	40.96900 s	0.00055 s	3.55823 s	-0,009917547	5,611917547
2	6,973000	72.35400 s	0.00902 s	6.34005 s	-0,002565984	6,975565984
5	13,266000	167.35700 s	0.00300 s	15.59070 s	-0,005585919	13,27158592
10	24,170000	369.28900 s	0.00491 s	31.05123 s	-0,009917547	24,17991755
20	41,901000	Error de Memoria	0.00455 s	63.82649 s	-0,01732651	41,91832651

METODO HEAP SORT



Algoritmo Heap Sort sin hilos (n log n)						
numero datos	tiempo en minutos					
	Sin Hilos		Con Hilos			
millones	sin imprimir	imprimiendo datos	sin imprimir	imprimiendo datos	tiempo esperado	delta tiempo
1	3,94700000	51.64100 s	0.05899 s	3.24244 s	-0,009917547	3,95691755
2	3,69700000	100.40900 s	0.02747 s	6.38755 s	-0,002565984	3,69956598
5	3,32500000	236.70900 s	0.05179 s	15.72822 s	-0,005585919	3,33058592
10	4,62300000	514.75100 s	0.02124 s	31.57545 s	-0,009917547	4,63291755
20	5,36600000	339.91800 s	0.14495 s	65.25553 s	-0,01732651	5,38332651

METODO QUICK SORT

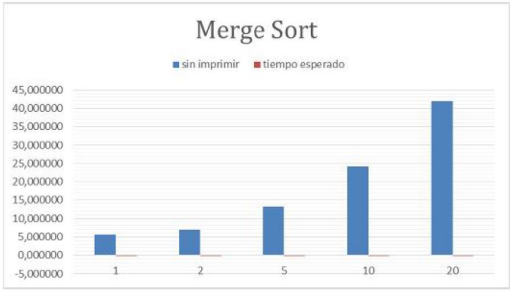


Algoritmo quick Sort sin hilos O(n·log n)						
numero datos	tiempo en minutos					
	Sin Hilos		Con Hilos			
millones	sin imprimir	imprimiendo datos	sin imprimir	imprimiendo datos	tiempo esperado	delta tiempo
1	5,67800000	34,54782 s	1.82627 s	3.40005 s	-0,009917547	5,68791755
2	7,19200000	93,14563 s	1.72050 s	6.82093 s	-0,002565984	7,19456598
5	13,19900000	303,1546 s	0.24087 s	16.93130 s	-0,005585919	13,20458592
10	23,14600000	991,2256 s	16.10980 s	33.03148 s	-0,009917547	23,15591755
20	42,22800000	2458,9514 s	49.22254 s	66.72416 s	-0,01732651	42,24532651

METODO COUTING SORT

Algoritmo Counting Sort sin hilos O(n+k)						
numero datos	tiempo en minutos					
	Sin Hilos		Con Hilos			
millones	sin imprimir	imprimiendo datos	sin imprimir	imprimiendo datos	tiempo esperado	delta tiempo
1	18,502000	32.93400 s	3.92404 s	5.98475 s	0,004166667	18,497833
2	7,571000	53.54600 s	5.60954 s	9.08253 s	0,000833333	7,570167
5	18,081000	127.41400 s	8.92366 s	18.92688 s	0,002083333	18,078917
10	22,486000	266.65700 s	7.04798 s	34.84244 s	0,004166667	22,481833
20	42,383000	498,3365 s	6.86905 s	67.01715 s	0,008333333	42,374667

METODO RADIX SORT



Algoritmo Radix Sort sin hilos O(nk)						
numero datos	tiempo en minutos					
	Sin Hilos		Con Hilos			
millones	sin imprimir	imprimiendo datos	sin imprimir	imprimiendo datos	tiempo esperado	delta tiempo
1	6,0290000	86,1354 s	0.05080 s	3.20842 s	0,004166667	6,024833333
2	8,5230000	214,254 s	0.14823 s	6.26211 s	0,000833333	8,522166667
5	13,7510000	1001,5874 s	0.34594 s	15.16091 s	0,002083333	13,74891667
10	22,3600000	Error de Memoria	0.29293 s	31.88739 s	0,004166667	22,35583333
20	42,0690000	Error de Memoria	0.04104 s	63.67557 s	0,008333333	42,06066667

CONCLUSIONES

Al realizarse el ejercicio de los métodos de ordenamiento de inserción mescla, heap sort, quicksort, ridex sort,conteo implementados en su código fuente los hilos y el mismo ejercicio sin hilos esto con el fin de realizar un cuadro comparativo en sus tiempos de procesos se llegaron a unas conclusiones Al implementar los métodos de ordenamiento con hilos se nota la gran diferencia en sus tiempos de ejecución de los procesos haciendo más eficiente el ordenamiento. El uso de programación con hilos permite conocer más a profundidad el funcionamiento de los procesos que se ejecutan en el procesador. Conocer los diferentes métodos de ordenamiento y la forma de programar usando diferentes técnicas, permite adquirir nuevas herramientas al momento de desarrollar software.

RECOMENDACIONES

Teniendo en cuenta las observaciones en la aplicación de los diferentes métodos de ordenamientos donde se implementaron los hilos y sin hilos se llegaron a algunas conclusiones y sobre estas algunas recomendaciones como son: Para un buen funcionamiento de cada uno de los métodos de ordenamiento es fundamental tener buen recurso en sistemas de cómputo especialmente procesador. De acuerdo a la necesidad que se tenga en la búsqueda y organización de datos se debe implementar un método de ordenamiento. Actualmente uno de los métodos más recomendados y el más eficiente para la búsqueda y organización de datos es el quick sort.

COMPUTADOR USADO

Procesador Intel Core i7 4.0 GHz

Memoria RAM 8GB

REFERENCIAS BIBLIOGRÁFICAS

http://www.unilibrecucuta.edu.co/portal/images/investigacion/pdf/formato_papers.pdf

https://beastieux.com/2011/01/24/codigo-python-ordenamiento-por-insercion-directa/

https://programacion.net/articulo/algoritmos_de_ordenacion_y_busqueda_en_python_1387