	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Professor: Jesus Cruz Navarro

Asignatura: Estructura de Datos y Algoritmos 2

Grupo: 1

No de Práctica(s): 04

Integrante(s): Diego Santiago Gutiérrez

No. de Equipo de cómputo empleado:

No. de Lista o Brigada:

Semestre: Tercer Semestre

Fecha de entrega: 27/10/2020

Observaciones:

CALIFICACIÓN: _____

PRACTICA 04: ALGORITMO DE BUSQUEDA

OBJETIVO:

El estudiante identificará el comportamiento y características de algunos algoritmos de búsqueda por comparación.

DESARROLLO:

1. Pruebas de los algoritmos vistos en clase.

   a. Cada algoritmo debe estar en una función:
      i. Búsqueda Lineal (Mejorado)
      ii. Búsqueda Binaria (Recursivo)

   b. Desarrollar un programa para probar los algoritmos implementados anteriormente y registrar sus tiempos de ejecución con listas de tamaño n con elementos en el rango [0, n/2]. Ambos algoritmos deben recibir el mismo conjunto de elementos (es decir, si es una lista aleatoria, los algoritmos deben recibir el mismo conjunto aleatorio). El programa debe imprimir al final n, key, y el tiempo de ejecución de cada algoritmo. Busque la llave key de su preferencia.

   c. Se debe probar con listas de diferentes tamaños (n=0.2M, 0.5M, 1M) y registrar los tiempos de ejecución para el caso promedio.
   NO CONSIDERAR EL ORDENAMIENTO COMO PARTE DEL TIEMPO DE EJECUCIÓN DE LA BÚSQUEDA BINARIA.

Dado que los tiempos pueden variar de una ejecución a otra, realice las pruebas 3 veces y obtenga el promedio de los tiempos y genere una gráfica para comparar los tiempos de ambos algoritmos en función del tamaño de la lista (n vs t, es decir, las n en el eje X y los tiempos en segundos en el eje Y).
En lugar de usar time.time() y use time.perf_counter() que tiene mayor precisión. Su uso es el mismo.

1.a)

```
def BusqLineal(A, x): #Recibe un arrleglo A y X que es el elemento a buscar
    for k in (A): #Recorremos el arreglo A para ir realizando la busqueda
        if A[k] == x: #If A en la posicion k es igual a X
            return k #Te regresara K que es la posicion del arreglo
    return "No existe" #Si no se encontró nada regresa un none
```

BúsquedaLineal

Búsqueda lineal recorrerá el arreglo hasta encontrar el valor encontrado, si este no existe devolverá un mensaje anunciando lo anterior.

1.b)

```python
def BusqBinRec(A, x, izq, der): #Busqueda binaria recibe el arreglo, el elemento a buscar, el inicio del arreglo y el final del mismo.
    if izq>der: #Si izquierda es mayor a derecha , quiere decir que el arreglo no esta lleno o tiene un elemento
        return izq # Regresa un menos uno . es decir un none

    medio = (izq+der)//2 #Sacamos el medio del arreglo
    if x == A[medio]: # Si el valor x que estamos buscando es igual al elemento del medio
        return medio #Se retornara el valor del medio (indice)

    if x > A[medio]: #Si el valor que buscamos es mayor al valor del medio, se buscara por el lado derecho
        return BusqBinRec(A, x, medio+1, der) #Retoramos el valor de la funcion pero ahora dandole el valor del inicio, se le da la mitad
    else:
        return BusqBinRec(A, x, izq, medio-1) #De lo contrario se buscara del medio a la izquierda del programa

# Wrapper de búsqueda binaria modificada
# Esta versión retorna el índice donde el elemento
# debería estar, en caso de no encontrarse
def BinarioModificacion(A, x):
    return BusqBinRec(A, x,  0, len(A) - 1)
```

BusqBinRec

Esta función buscará el valor medio del arreglo para buscar sobre los extremos, es decir del medio a izquierda y del medio a la derecha. Si se encuentra el valor en el medio no es necesario buscar en otro lugar. En caso de ser negativo, la función se llamará así misma y buscará desde estas nuevas posiciones hasta encontrarlo y retornar el valor.

BinarioModificacion nos ayudará a encontrar el valor de k en la parte dos de la práctica.

2. <u>Búsqueda de los m vecinos más cercanos en un radio k.</u>

   a. Diseñar y desarrollar un algoritmo que permita, utilizando búsqueda binaria y búsqueda lineal, encontrar los m elementos más cercanos a una distancia máxima r de un valor x, en una colección de enteros aleatorios. X puede no existir dentro de la colección. Si se encuentra menos elementos que m, regresar solo estos.

   b. El algoritmo debe estar en una función BusquedaVecinos(arreglo, x, r, m) y regresar una lista con los elementos encontrados. Si no hay elementos, regresa una lista vacía.

```python
def findClosestNeighbors(A, num, max, busqueda):

    A = A.copy() # hacemos una copia del arreglo
    A.sort()  # Arreglamos el arreglo en caso de no estarlo
    index = BinarioModificacion(A, busqueda) #Copiamos el indice del valor devuelto
    vecinos = [] #Creamos el arreglo con el que llenaremos de vecinos

    def distancias(idx):              #Calculamos las distancias si son menor al radio
        return abs(A[idx] - busqueda) #Regresamos el valor de dicha operacion

    leftPointer = index - 1 #Elemento por la izquierda
    rightPointer = index #Elemento por la derecha

    # Diferencia entre los elementos anteriores
    # y el elemento buscado
    difDer = distancias(rightPointer) #Calculamos la distancia del puntero a la derecha
    difIzq = math.inf #Sacamos math.ing a difIzq
    if leftPointer >= 0: #Si el puntero es mayor a 0
        difIzq = distancias(leftPointer) #difIzq será la distancia que hay con el punteroizquierdo

    while len(vecinos) < num and (difIzq <= max or difDer <= max):
    #Si el tamaño de vecinos es menor al numero }
    #Solicitado y que sean menor o mayor al rango por izq o derecha
    #ambos vecinos candidatos están dentro del rango


        if difIzq <= difDer:  # La distancia al vecino izquierdo es menor o igual
            vecinos.append(A[leftPointer]) #Insertas el valor en el arreglo
            leftPointer -= 1 #Recorres un valor a la izquierda

            if leftPointer == -1: #Si te sales del arreglo
                difIzq = math.inf #invalidas el lado
            else:
                difIzq = distancias(leftPointer) #La diferencia de la distancia con el puntero izquierda
```

```
    else: # La distancia al vecino derecho es mayor
        vecinos.append(A[rightPointer]) #Insertar el valor que se encuentra en el puntero derecho a vecinos
        rightPointer += 1 #Incrementas el valor

        if rightPointer == len(A): #si se llega al final
            difDer = math.inf #invalidas el valor
        else:
            difDer = distancias(rightPointer) #Regresas el valor de la distancai con respecto al puntero derecha

vecinos.sort() #Acomodas el arreglo
return vecinos #Retornas el valor
```

FindClosesNeihgbors

Esta función funcionará con los principios de la búsqueda lineal y binaria, ya que la lineal nos ayudará a encontrar el elemento y la binaria a buscar, localizar y dividir en izquierda y derecha el arreglo para realizar la búsqueda por ambos extremos.

```
def distancias(idx):              #Calculamos las distancias si son menor al radio
    return abs(A[idx] - busqueda) #Regresamos el valor de dicha operacion
```

Distancias

Distancias nos ayudará a saber si los valores que buscamos se encuentran dentro del rango de búsqueda.

Utilizando unos punteros que nos ayudarán a fijar los lugares para empezar la búsqueda por el lado izquierdo y derecho con referencia al índice. Realizamos comparaciones si la diferencia de estos valores era mayor a 0, lo que significa que hay valores dentro de este y posterior sacamos la distancia.

El ciclo while iterará hasta que ya no haya más vecinos que agregar y que la diferencia de difIzquierdo sea menor o igual al rango o el lado derecho.
Mientras este el ciclo se realizarán comparaciones del lado izquierdo, tomando en cuenta que deben de estar dentro del rango, cada vez reduciendo un lugar en su casilla hasta que se llegue al final. Si la dif del lado izq no es menor o igual al derecho, se buscará del lado derecho realizando los mismos procesos. En ambos casos se insertará el elemento en caso de ser afirmativo.

Se acomoda el arreglo y regresa este mismo.

RESULTADOS:

N=200,000

```
¿Que elemento deseas buscar?
666
BUSQUEDA BINARIA
        El numero:  666  indice:  1343
        Tiempo busqueda Binaria: {0:f} segundos 3.460000000021779e-05
        n:  200000

BUSQUEDA LINEAL
        El numero:  666  indice:  1342
        Tiempo busqueda Binaria: {0:f} segundos 0.00025010000000058596
        n:  200000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

```
¿Que elemento deseas buscar?
666
BUSQUEDA BINARIA
        El numero:  666  indice:  1346
        Tiempo busqueda Binaria: {0:f} segundos 3.6500000000216914e-05
        n:  200000

BUSQUEDA LINEAL
        El numero:  666  indice:  1346
        Tiempo busqueda Binaria: {0:f} segundos 0.0005664000000003
        n:  200000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

```
[40, 42, 43]
[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]

        n:  500000
        Tiempo busqueda Binaria: {0:f} segundos 0.0005150000000018084
        El numero:  666  indice:  1346
BUSQUEDA LINEAL

        n:  500000
        Tiempo busqueda Binaria: {0:f} segundos 3.4000000000218e-05
        El numero:  666  indice:  1346
BUSQUEDA BINARIA
666
¿Que elemento deseas buscar?
```

N=500,000

```
¿Que elemento deseas buscar?
6666
BUSQUEDA BINARIA
        El numero:  6666  indice:  13334
        Tiempo busqueda Binaria: {0:f} segundos 3.970000000030893e-05
        n:  500000

BUSQUEDA LINEAL
        El numero:  6666  indice:  13334
        Tiempo busqueda Binaria: {0:f} segundos 0.005142300000001043
        n:  500000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

```
¿Qué elemento deseas buscar?
66666
BUSQUEDA BINARIA
        El numero:  66666  indice:  133827
        Tiempo busqueda Binaria: {0:f} segundos 4.479999999773554e-05
        n:  500000

BUSQUEDA LINEAL
        El numero:  66666  indice:  133827
        Tiempo busqueda Binaria: {0:f} segundos 0.06174299999999988
        n:  500000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

```
¿Qué elemento deseas buscar?
66666
BUSQUEDA BINARIA
        El numero:  66666  indice:  133827
        Tiempo busqueda Binaria: {0:f} segundos 4.479999999773554e-05
        n:  500000

BUSQUEDA LINEAL
        El numero:  66666  indice:  133827
        Tiempo busqueda Binaria: {0:f} segundos 0.06174299999999988
        n:  500000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

N=1,000,000

```
¿Qué elemento deseas buscar?
666666
BUSQUEDA BINARIA
        El numero:  666666  indice:  1000000
        Tiempo busqueda Binaria: {0:f} segundos 4.7500000000866294e-05
        n:  1000000

BUSQUEDA LINEAL
        El numero:  666666  indice:  No existe
        Tiempo busqueda Binaria: {0:f} segundos 0.26110219999999984
        n:  1000000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

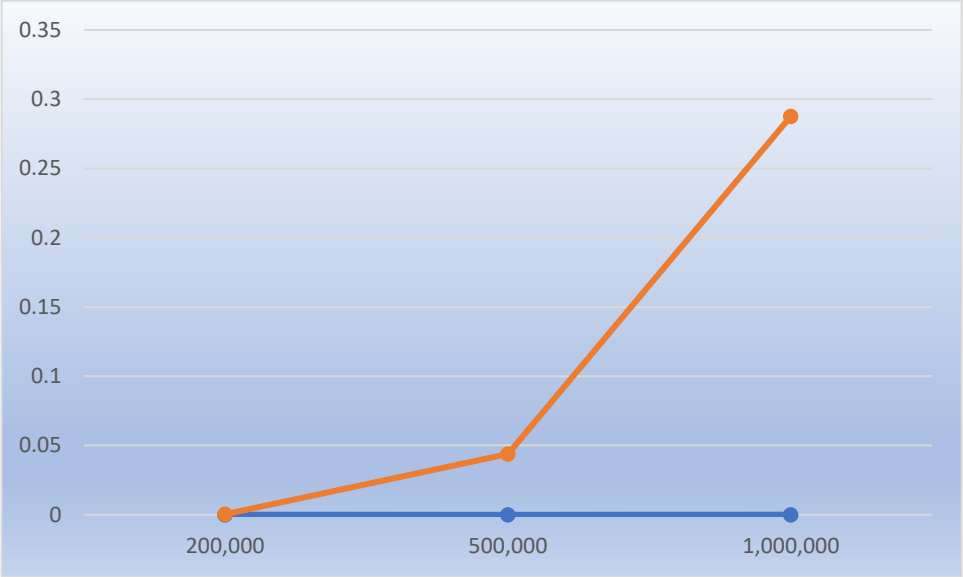
```
¿Qué elemento deseas buscar?
666666
BUSQUEDA BINARIA
        El numero:  666666  indice:  1000000
        Tiempo busqueda Binaria: {0:f} segundos 5.4099999999834836e-05
        n:  1000000

BUSQUEDA LINEAL
        El numero:  666666  indice:  No existe
        Tiempo busqueda Binaria: {0:f} segundos 0.3005279999999999
        n:  1000000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

```
¿Qué elemento deseas buscar?
666666
BUSQUEDA BINARIA
        El numero:  666666  indice:  1000000
        Tiempo busqueda Binaria: {0:f} segundos 5.4099999999834836e-05
        n:  1000000

BUSQUEDA LINEAL
        El numero:  666666  indice:  No existe
        Tiempo busqueda Binaria: {0:f} segundos 0.3005279999999999
        n:  1000000

[80, 82, 84, 85, 86, 88, 90, 91, 92, 94]
[40, 42, 43]
```

TABLAS DE COMPARACIÓN:

N	p1	p2	p3	PROMEDIO
200,000	3.46E-05	3.65E-05	3.47E-05	3.52667E-05
500,000	3.97E-05	4.48E-05	5.45E-05	4.63333E-05
1,000,000	4.75E-05	5.41E-05	5.53E-05	5.23E-05
BINARIA				

N	p1	p2	p3	PROMEDIO
200,000	2.50E-04	0.0005664	0.0002167	0.0003444
500,000	0.0051423	0.061743	0.0649226	0.043935967
1,000,000	0.2611022	0.300528	0.3012381	0.287622767
LINEAL				



NARANJA: BÚSQUEDA LINEAL
AZUL= BÚSQUEDA BINARIA

X= N DATOS
Y= TIEMPO

CONCLUSIONES:

Los algoritmos de búsqueda son realmente importantes en el mundo de la informática, debido a que estos son manejados a escalas enormes debido a su importancia de almacenamiento. Siempre es importante conocer que además de ordenar, también es importante la búsqueda de forma eficaz y rápida, en estos casos evaluando lo que es la búsqueda lineal y binaria.

Como bien es apreciable, en la grafica vemos como hay un aumento de tiempo bastante considerable en lo que es Búsqueda Lineal en comparación de la búsqueda binaria, esto es debido a que es de una complejidad asintótica ya que es necesario recorrer todo el arreglo de inicio a final. Para los datos de una búsqueda binaria es importante que estos se encuentren ordenados debido a que sino su eficiencia se reduce y se vuelve un algoritmo ineficiente. La implementación de la búsqueda lineal es mejor, pero queda a consideración de quien vaya a aplicarla según la cantidad de elementos que se tengan para buscar.