

Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Estructuras de datos y algoritmos II

Grupo 1

Proyecto A: Algoritmos de Ordenamiento.

Profesor: Cruz Navarro Jesús

Alumno: Camarena Ruiz Diego Henok

Descripción del programa.

- Modificaciones a los algoritmos de ordenamiento para que el orden sea descendente.

Los cambios son muy ligeros, en realidad solo hubo necesidad de cambiar algunos operadores lógicos

- Merge Sort.

Normal

```
41 def Merge(A,I,D):
42     global comp
43     indIzq=0
44     indDer=0
45     for i in range (len(A)):
46         if (indDer >= len(D)) or (indIzq<len(I) and I[indIzq]<D[indDer]):
47             A[i]=I[indIzq]
48             indIzq=indIzq+1
49             comp=comp+1
50         else:
51             A[i]=D[indDer]
52             indDer=indDer+1
53             comp=comp+1
54     print("Num comp=",comp)
55
```

Modificado

```
15 def Merge(A,I,D):
16     global comp
17     indIzq=0
18     indDer=0
19     for i in range (len(A)):
20         if (indDer >= len(D)) or ( indIzq<len(I) and I[indIzq]>D[indDer] ):
21             A[i]=I[indIzq]
22             indIzq=indIzq+1
23             comp=comp+1
24         else:
25             A[i]=D[indDer]
26             indDer=indDer+1
27             comp=comp+1
28     #print("Num comp=",comp)
29
```

El único cambio se hizo en la línea 46 y 20 (respectivamente) dentro de la función Merge. Se modificó el último signo de desigualdad que se aprecia en la línea. Pasó de ser "<" a ">".

- Quick Sort.

Normal

```
12 def particion(A,low,high):
13     global contador
14     i=low-1
15     pivote=A[high]
16     for j in range(low,high):
17         contador=contador+1
18         if (A[j]<=pivote):
19             i=i+1
20             swap(A,j,i)
```

Modificado

```
50 def particion(A,low,high):
51     global contador
52     i=low-1
53     pivote=A[high]
54     for j in range(low,high):
55         contador=contador+1
56         if (A[j]>=pivote):
57             i=i+1
58             swap(A,j,i)
```

El cambio se puede apreciar en las líneas 18 y 56 (respectivamente) dentro de la función partición. En el condicional if, nuevamente se cambió el signo de desigualdad; pasó de ser “<=” a ser “>=”.

- Insertion Sort

Normal

```
46 def insertionSort(A):
47     global contador
48     for i in range(1,len(A)):
49         valor=A[i]
50         j=i-1
51         contador=contador+1
52         while (j>=0 and A[j]>valor):
53             contador=contador+1
54             A[j+1]=A[j]
55             j=j-1
56         A[j+1]=valor
57
```

Modificado

```
70 def insertionSortInverso(A):
71     global contador
72     for i in range(1,len(A)):
73         valor=A[i]
74         j=i-1
75         contador=contador+1
76         while (j>=0 and A[j]<valor):
77             contador=contador+1
78             A[j+1]=A[j]
79             j=j-1
80         A[j+1]=valor
81     #INSERTION SORT INVERSO
```

La modificación se hizo en las líneas 53 y 77 (respectivamente). Se cambió el signo de ">" a "<" en la segunda condición del ciclo while.

- Radix Sort

Normal

```
13 def CountingSort(A,pos,base):
14     k=base-1
15     n=len(A)
16     B=[0]*n
17     C=[0]*(k+1)
18     for i in range(n):
19         digito=ObtenerDigito(A[i],pos,base)
20         C[digito]=C[digito]+1
21     C[0]=C[0]-1
22     print(C[base-1])
23     for i in range(1,len(C)):
24         C[i]=C[i]+C[i-1]
```

Modificado

```
95 def CountingSort(A,pos,base):
96     k=base-1
97     n=len(A)
98     B=[0]*n
99     C=[0]*(k+1)
100    for i in range(n):
101        digito=ObtenerDigito(A[i],pos,base)
102        C[digito]=C[digito]+1
103    C[base-1]=C[base-1]-1
104    print(C[base-1])
105    for i in range( len(C)-1,0,-1 ):
106        C[i-1]=C[i-1]+C[i]
```

Este fue el más complicado de modificar. Se hicieron dos cambios:

1. Líneas 21 y 103 respectivamente:

Originalmente, al arreglo auxiliar C se le debía restar una unidad a la posición C[0] al momento de mapear, pues el dígito más pequeño debía quedar en la posición 0 del arreglo original A. Mi método fue hacerlo al revés: restarle una unidad a la última posición del arreglo C, en este caso, a C[9].

2. Ciclo for de las líneas 23 y 105 respectivamente:

Este for es para mapear los índices de C que están relacionados con los valores contenidos en A. En el original, esto se hace de derecha a izquierda (de C[0] a C[9]). Mi solución fue hacerlo a la inversa, ya que, con lo hecho en el apartado 1, el dígito más pequeño se guarda en la posición más grande de A. Por ejemplo, si tenemos el dígito 3 y después tenemos el 7, el 3 se guardará en C[9] y el 7 en C[8]. Haciendo esto, el resultado será una lista ordenada descendentemente.

- Código restante
Función “todoElPrograma”

```

117 def todoElPrograma(A):
118     ArregloParaInsertion = A
119     ArregloParaRadix = A
120     ArregloParaMerge = A
121     ArregloParaQuick = A
122
123     t1=time.time()
124     insertionSortInverso(ArregloParaInsertion)
125     t2=time.time()
126     tiempoInsertion = t2-t1
127     t3=time.time()
128     x = RadixSort(ArregloParaRadix)
129     t4=time.time()
130     tiempoRadix = t4-t3
131     #print("Arreglo con Radix Sort:",x)
132     t5=time.time()
133     MergeSort(ArregloParaMerge)
134     t6=time.time()
135     tiempoMerge = t6-t5
136     t7=time.time()
137     quickSortInverso(ArregloParaQuick)
138     t8=time.time()
139     tiempoQuick = t8-t7
140     diccionarioAlgoritmos = {'Quicksort':tiempoQuick, 'MergeSort':tiempoMerge, 'RadixSort':tiempoRadix, 'InsertionSort':tiempoInsertion}
141     print("DICCIONARIO: ", diccionarioAlgoritmos)
142

```

```

151
152     with open('resultados.eda2','w') as resultados:
153         resultados.write(str(now)+"\n")
154         tam = str(len(A))
155         resultados.write("Tamaño de la colección: "+tam+"\n")
156         alg_sort = sorted(diccionarioAlgoritmos.items(), key=operator.itemgetter(1), reverse=False)
157         for name in enumerate(alg_sort):
158             print(name[1][0], ' ha tardado ', diccionarioAlgoritmos[name[1][0]])
159             resultados.write( str(name[1][0])+ ' ha tardado ' + str(diccionarioAlgoritmos[name[1][0]])+"[s]\n")
160

```

La función “todoElPrograma” hace que todos los algoritmos se ejecuten con el mismo arreglo, además de calcular el tiempo que tardó cada uno. Para imprimirlos en función del tiempo, creé un diccionario con el nombre como llave y el tiempo que tardaron como valor.

```

162 arr = []
163 try: #Primer try verifica que exista el archivo
164     with open("C:/Users/52552/Desktop/indput.eda2", "r") as archivo:
165         archivo.readline()
166         print("Se encontró el archivo")
167
168 except:
169     print("El archivo no existe. La lista se generará aleatoriamente")
170     while(True): #Este while es para evitar que el programa excepcione si el usuario no ingresa int's
171         try:
172             n = int( input("Escribe el número de elementos del arreglo") )
173             Arreglo=random.sample(range(0,n*n),n) #rango [0, n^2]
174             todoElPrograma(Arreglo)
175             sys.exit()
176         except:
177             print("Ingresa bien la cantidad de elementos que debe tener el arreglo")
178
179 try: #El segundo verifica que esté en el formato correcto
180     with open("C:/Users/52552/Desktop/input.eda2", "r") as archivo:
181         lista = [int(linea) for linea in archivo] #Esta línea generaría la excepción en caso de que el archivo no esté en el formato correcto
182         arr = lista
183         todoElPrograma(arr)
184         print(arr)
185
186 except:
187     print("Formato de archivo incorrecto.")
188     sys.exit(0)

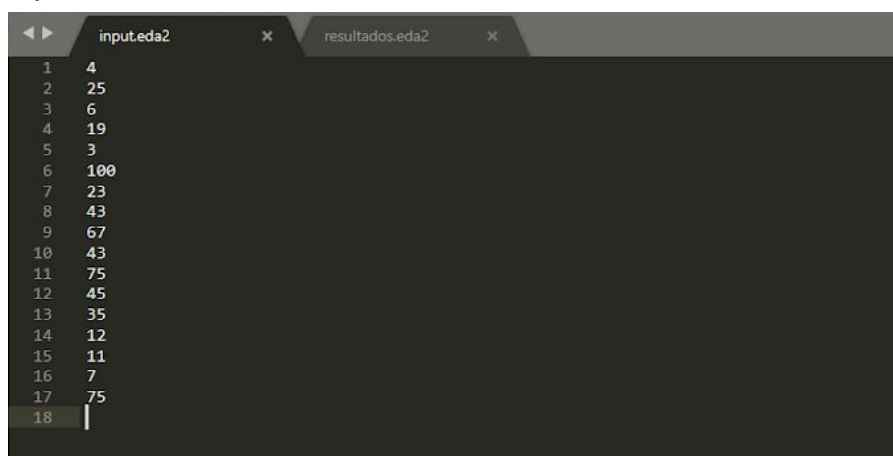
```

Finalmente, las excepciones. Si el archivo no tiene el formato correcto, el programa se cierra. Si no existe el archivo, se le pide al usuario que ingrese la cantidad de elementos en la lista. Con el ciclo while procuré cualquier tipo de excepción posible, por ejemplo, si un usuario ingresa una cadena o caracter en lugar de un entero.

Entradas y salidas.

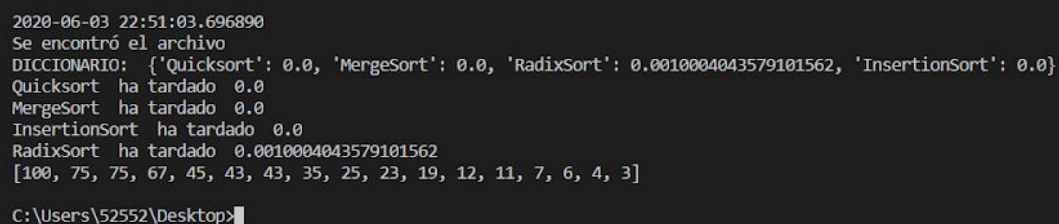
- Caso 1: Si el archivo de texto existe y tiene el formato correcto

input.eda2



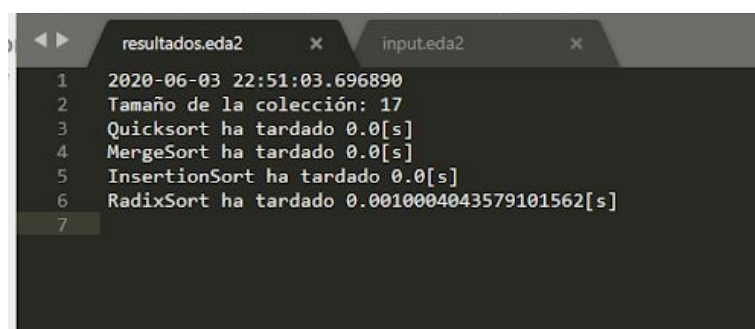
```
1 4
2 25
3 6
4 19
5 3
6 100
7 23
8 43
9 67
10 43
11 75
12 45
13 35
14 12
15 11
16 7
17 75
18
```

Consola



```
2020-06-03 22:51:03.696890
Se encontró el archivo
DICCIONARIO: {'Quicksort': 0.0, 'MergeSort': 0.0, 'RadixSort': 0.0010004043579101562, 'InsertionSort': 0.0}
Quicksort ha tardado 0.0
MergeSort ha tardado 0.0
InsertionSort ha tardado 0.0
RadixSort ha tardado 0.0010004043579101562
[100, 75, 75, 67, 45, 43, 43, 35, 25, 23, 19, 12, 11, 7, 6, 4, 3]
C:\Users\52552\Desktop>
```

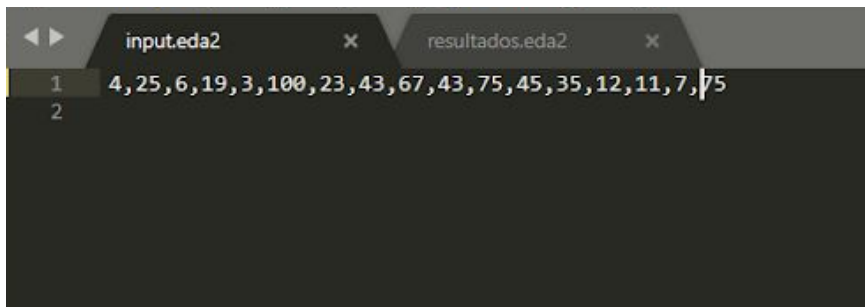
resultados.eda2



```
1 2020-06-03 22:51:03.696890
2 Tamaño de la colección: 17
3 Quicksort ha tardado 0.0[s]
4 MergeSort ha tardado 0.0[s]
5 InsertionSort ha tardado 0.0[s]
6 RadixSort ha tardado 0.0010004043579101562[s]
7
```

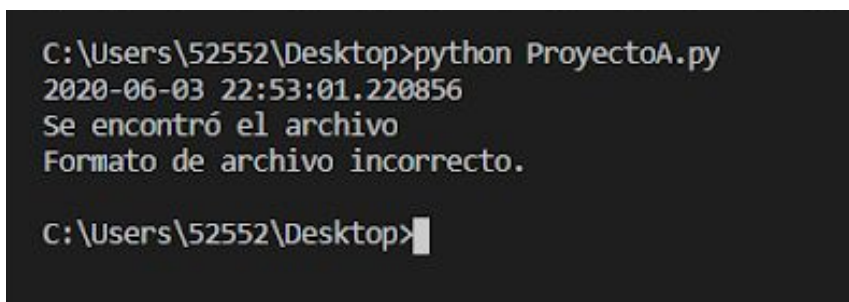

- Caso 2: Si el archivo existe pero no tiene el formato correcto

input.eda2



```
input.eda2 x resultados.eda2 x
1 4, 25, 6, 19, 3, 100, 23, 43, 67, 43, 75, 45, 35, 12, 11, 7, 75
2
```

consola

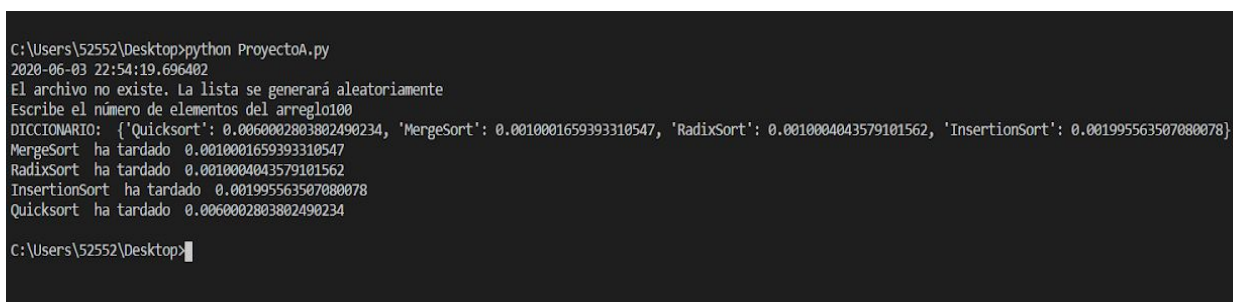


```
C:\Users\52552\Desktop>python ProyectoA.py
2020-06-03 22:53:01.220856
Se encontró el archivo
Formato de archivo incorrecto.

C:\Users\52552\Desktop>
```

- Caso 3: El archivo no existe (le he cambiado el nombre a “dinput.eda2”)

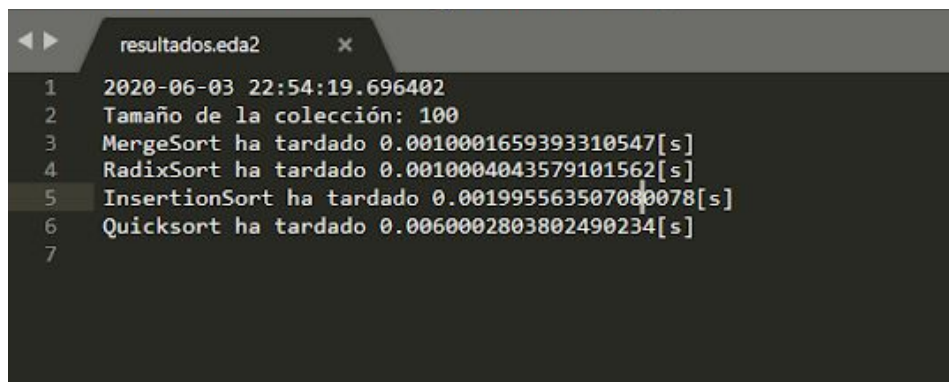
consola



```
C:\Users\52552\Desktop>python ProyectoA.py
2020-06-03 22:54:19.696402
El archivo no existe. La lista se generará aleatoriamente
Escribe el número de elementos del arreglo100
DICCIONARIO: {'Quicksort': 0.0060002803802490234, 'MergeSort': 0.0010001659393310547, 'RadixSort': 0.0010004043579101562, 'InsertionSort': 0.001995563507080078}
MergeSort ha tardado 0.0010001659393310547
RadixSort ha tardado 0.0010004043579101562
InsertionSort ha tardado 0.001995563507080078
Quicksort ha tardado 0.0060002803802490234

C:\Users\52552\Desktop>
```


resultados.eda2



```
resultados.eda2 x
1 2020-06-03 22:54:19.696402
2 Tamaño de la colección: 100
3 MergeSort ha tardado 0.0010001659393310547[s]
4 RadixSort ha tardado 0.0010004043579101562[s]
5 InsertionSort ha tardado 0.001995563507080078[s]
6 Quicksort ha tardado 0.0060002803802490234[s]
7
```

Conclusiones.

Sobre la modificación de los algoritmos, no se me complicó demasiado, me tardé un poco para Radix Sort pero con lápiz y papel resolví el problema. Lo que considero más difícil fue el pasar el arreglo desordenado a la función “todoElPrograma”, pues aunque intentaba reiniciar al arreglo desordenado cada vez que pasaba por un algoritmo, incluso el arreglo auxiliar se modificaba. Aún no entiendo la razón pero por eso decidí crear un arreglo auxiliar para cada algoritmo, aunque entiendo que es un gasto mayor de memoria. Las excepciones me confundieron un poco pero con un poco de lógica antes de intentar programarlas, todo se hizo mejor y más rápido. Creo que cubrí todas las excepciones que se podrían generar.