Ignacio Coppede | Algoritmos y Estructuras de Datos II | TP°2

Ejercicio 1

Demuestre que $6n^3 \neq O(n^2)$

$T(n) = O(f(n))$ si existe $C > 0$ y $n_0 \in \mathbb{N}$ Tal que

$T(n) \leq C \cdot f(n)$, $n \geq n_0$

$6n^3 \neq O(n^2)$ ya que no existe $C > 0$ ni $n_0 \in \mathbb{N}$ tal que

$6n^3 \leq C \cdot n^2$, $n \geq n_0$

$\frac{6n^3}{n^2} \leq C \Rightarrow 6n \leq C$ Por lo tanto C no sería constante

Ejercicio 3:

QuickSort(A): $O(n^2)$
Insertion-Sort(A): $O(n)$
Merg-sort(A): $O(n \cdot \log n)$

Ejercicio 2:

1	2	3	4	5	6	7	8	9	10

De esta forma la lista se va a dividir hasta quedan listas de tamaño 1, pero como ya estan ordenadas solo se van a tener que unir

Ejercicio 6:

Bucket sort

Es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre numero finito de casilleros cada casillero solo puede contener los elementos que cumplan con unas determinadas condiciones

Ej: A = [0, 1, 25, 12, 30, 29, 13, 7, 5, 9, 15]

1 7 / 5 9		12 13 / 15		25	30 29
0-5	5-10	10-15	15-20	20-25	25-30

Despues se ordena individualmente cada casillero

1 5 / 7 9		12 13 / 15		25	29 30

Despues llevar al Array

A = [1, 5, 7, 9, 12, 13, 15, 25, 29, 30]

El orden de complejidad puede variar según el método que usemos para ordenar cada casillero y como dividamos el rango de cada casillero

El peor caso puede suceder cuando agremamos de forma incremente los casilleros y todos los elementos se depositan en un solo casillero

A=[1,2,3,4,5,11]

[1,2,3,4,5] [11]
1-10 10-20

El mejor caso es cuando cada elemento este en un solo casillero

A=[1, 10, 20, 30]

[1] [10] [20] [30] A = [1, 10, 30, 30]
[0-10] [10-19] [20-29] [30-39]

Ejercicio 7:

a) $T(n) = 2T(n/3) + n^{1}$ $a=2$ $b=2$ $f(n)=n^{1}$

$\log_2(2) = 1$

caso 3: $f(n) = n^{1} = O(n^{1+\varepsilon})$

$n^{1} = O(n^{1+\varepsilon})$; $\varepsilon = 3$

Entonces $T(n) = O(n^{1})$

b) $T(n) = 2 \cdot T\left(\frac{7n}{10}\right) + n$ $a=2$ $b=\frac{10}{7}$ $f(n)=n$

caso 1 $f(n) = n = O(n^{1,94-\varepsilon})$ $\log_{\frac{10}{7}}(2) \approx 1,94$

$n = O(n^{1,94-0,94})$ $\varepsilon \approx 0,94$

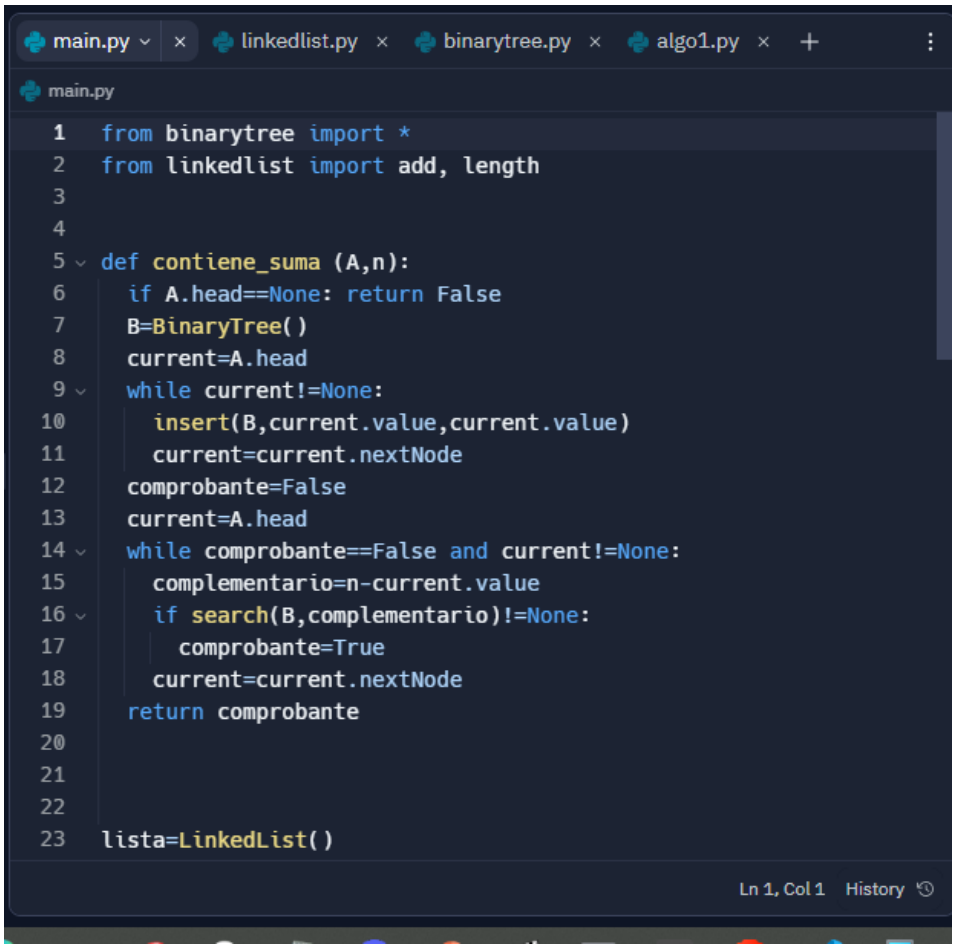
$T(n) = O(n^{\log_{\frac{10}{7}}(2)})$

c) $T(n) = 16\,T\left(\frac{n}{4}\right) + n^2$ $a=16$ $b>1$ $f(n)=n^2$

$\log_4 16 = 2$

caso 2: $f(n) = n^2 = O(n^2)$

$T(n) = O(n^2 \cdot \log n$


main.py
linkedlist.py
binarytree.py
algo1.py
main.py
1 from binarytree import *
2 from linkedlist import add, length
3
4
5 def contiene_suma (A,n):
6 if A.head==None: return False
7 B=BinaryTree()
8 current=A.head
9 while current!=None:
10 insert(B,current.value,current.value)
11 current=current.nextNode
12 comprobante=False
13 current=A.head
14 while comprobante==False and current!=None:
15 complementario=n-current.value
16 if search(B,complementario)!=None:
17 comprobante=True
18 current=current.nextNode
19 return comprobante
20
21
22
23 lista=LinkedList()
Ln 1, Col 1
History

main.py | linkedlist.py | binarytree.py | algo1.py

main.py

```
20
21
22
23  lista=LinkedList()
24  add(lista,5)
25  add(lista,1)
26  add(lista,1)
27  add(lista,1)
28  add(lista,-2)
29  add(lista,9)
30  add(lista,7)
31  add(lista,8)
32  add(lista,1)
33
34
35  print(contiene_suma(lista,5))
36  lista=[1,5,8,4,2,7,3,9,0,11,13,67,43,-1,12]
37
38
39  def ordenamiento_mitad(L):
40    n=len(L)
41    L.sort()
42    num=L[round(n/2)]
```

Ln 1, Col 1 History

main.py | linkedlist.py | binarytree.py | algo1.py

main.py

```
34
35  print(contiene_suma(lista,5))
36  lista=[1,5,8,4,2,7,3,9,0,11,13,67,43,-1,12]
37
38
39  def ordenamiento_mitad(L):
40    n=len(L)
41    L.sort()
42    num=L[round(n/2)]
43
44    for i in range (0,round(n/4)):
45      temporal=L[i]
46      L[i]=L[n-i-1]
47      L[n-i-1]=temporal
48    print (L)
49
50  ordenamiento_mitad(lista)
51
```

Ln 1, Col 1 History