Martins Ezequiel 14000

Trabajo Práctico Complejidad (U1)

https://replit.com/@EzeMarts/tp-complejidad

Ejercicio 1

Bueno para demostrar que $6n^3 \neq O(n^2)$, primero vamos a suponer que el enunciado es falso, por lo que vamos a plantear una contradicción del mismo $6n^3 = O(n^2)$. En general se dice que: $T(n)$ es $O(f(n))$ si existen constantes positivas c y n_0 tal que: $T(n) \leq cf(n)$ cuando $n \geq n_0$. En este caso tenemos que $T(n) = 6n^3$ y $O(f(n)) = O(n^2)$, por lo que quedaría: $6n^3 \leq c.n^2$, si dividimos ambos términos por n^2 nos quedaría $6n \leq c$, lo cual es una contradicción ya que existen $n \geq n_0$ donde $6.n \geq c$.

Ejercicio 2

Un array ejemplo para el mejor caso de Quicksort(n) seria: [2, 3, 1, 4, 5, 8, 6, 10, 7, 9] , o cualquier otro array que este desordenado, ya que si el array se encuentra ordenado, el tiempo de ejecución del programa seria de $O(n^2)$, pero al estar desordenado nos aseguramos que la lista se vaya dividiendo permitiendo que el tiempo de ejecución quede en O(n log n).

Ejercicio 3

Para Quicksort(A) el tiempo de ejecución es de $O(n^2)$, dejando el pivote seleccionado en una lista aparte a los demás repetidos, ya que si se queda en la misma lista que los demás no tendría fin el programa.
Para Insertionsort(A) el tiempo de ejecución es de O(n), ya que mientras recorre nunca se cumple la condición de que el siguiente nodo sea menor al actual y por ende recorre la lista solo una vez.
Para Mergesort(A) el tiempo de ejecución es de $O(n^2)$, dejando el pivote seleccionado en una lista aparte a los demás repetidos, ya que si se queda en la misma lista que los demás no tendría fin el programa.

Ejercicio 4

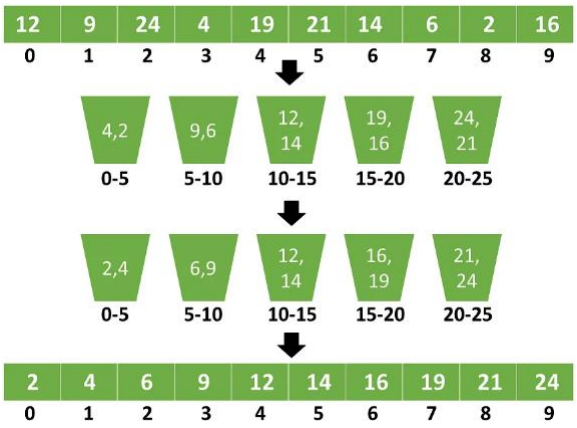
Ejercicio 5

```
def ContieneSuma(L, n):
  mainNode = L.head
  loopNode = mainNode.nextNode
  for i in range(0, length(L)-1):
    for j in range(i+1, length(L)):
      if mainNode.value + loopNode.value == n:
        return True
      loopNode = loopNode.nextNode
    mainNode = mainNode.nextNode
    loopNode = mainNode.nextNode
  return False
```

Peor Caso: $O(n^2)$
Mejor Caso: $\Omega(1)$
Caso Promedio: $\Theta(n^2)$

Ejercicio 6

BucketSort: El programa de ordenamiento BucketSort se encarga de agrupar en “casillas” los nodos de la lista que se encuentren en algún intervalo, en el caso de la imagen, recopila los nodos que se encuentren en intervalos de valores de x, x+5 y los introduce en una nueva lista donde posteriormente serán ordenados con algún programa de ordenamiento (a elección del programador), luego una vez que se han ordenado los casilleros, se vuelven a unir dando por resultado la lista ordenada.
Peor Caso: El peor caso seria que todos los elementos sean ubicados en la misma casilla, quedando de $O(n^2)$ con un programa de ordenamiento como insertion/bubble/selectionsort o en O(n log n) con un programa de ordenamiento por división merge/quicksort.
Mejor Caso: El mejor caso seria que los elementos a ordenar ya estén ordenados, por lo que quedaría en $\Omega(n)$.
Caso Promedio: En un caso promedio el programa se ejecutaría en $\Theta(n)$, ya que para encontrar el valor máximo se puede hacer en $\Theta(n)$, esto sirve para establecer los intervalos donde ir dividendo los elementos, introducir los elementos en las casillas también es de tiempo $\Theta(n)$, luego el $\Theta()$ del programa de ordenamiento dentro de las casillas se ve reducido por los pocos elementos hasta dar $\Theta(n)$ también, por lo que en un caso promedio quedaría en $\Theta(n)$.



Ejercicio 7

a. **$T(n) = 2T(n/2) + n^4$** *// $T(n) = aT(n/b) + n^c$*

$a = 2$, $b = 2$, $c = 4$

$\log_2 2 = 1 < 4$

$T(n) = \Theta(n^c) = \mathbf{\Theta(n^4)}$

b. **$T(n) = 2T(7n/10) + n$** *// $T(n) = aT(n/b) + n^c$*

$a = 2$, $b = 10$, $c = 1$

$\log_{10} 2 \approx 0.3 < 1$

$T(n) = \Theta(n^c) = \mathbf{\Theta(n)}$

c. **$T(n) = 16T(n/4) + n^2$** *// $T(n) = aT(n/b) + n^c$*

$a = 16$, $b = 4$, $c = 2$

$\log_4 16\ 2 = 2$

$T(n) = \Theta(f(n))\log n = \Theta(n^c \log n) = \mathbf{\Theta(n^2 \log n)}$

d. **$T(n) = 7T(n/3) + n^2$** *// $T(n) = aT(n/b) + f(n)$*

$a = 7$, $b = 3$, $c = 2$

$n^{\log_b(a)} = n^{\log_3(7)} \approx n^{1.77}$

$f(n) = n^2 = O(n^{\log_3(7) + \varepsilon})$

$f(n) = n^2 = O(n^{1.77 + \varepsilon})$

$\varepsilon \approx 0.23$

Caso 3: $af(n/b) \leq cf(n)$ $(\varepsilon<1)$

$7(n/3)^2 \leq cn^2$

$7(n^2/9) \leq cn^2$

$c = 7/9 < 1$

$T(n) = \mathbf{\Theta(n^2)}$

e. **$T(n) = 7T(n/2) + n^2$** *// $T(n) = aT(n/b) + f(n)$*

$a = 7$, $b = 2$, $c = 2$

$n^{\log_b(a)} = n^{\log_2(7)} \approx n^{2.8}$

$f(n) = n^2 = O(n^{\log_2(7) - \varepsilon})$

$f(n) = n^2 = O(n^{2.8 - \varepsilon})$

$\varepsilon \approx 0.8$

Caso 1: $T(n) = \Theta(n^{\log_b(a)}) = \mathbf{\Theta(n^{\sim 2.8})}$

f. **$T(n) = 2T(n/4) + \sqrt{n}$** *// $T(n) = aT(n/b) + f(n)$*

$a = 2$, $b = 4$, $c = 1/2$

$n^{\log_b(a)} = n^{\log_4(2)} = n^{0.5}$

$f(n) = n^{0.5} = O(n^{\log_4(2)})$

$f(n) = n^{0.5} = O(n^{0.5})$

Caso 2: $T(n) = \Theta(n^{\log_b(a)} \log n) = \mathbf{\Theta(n^{0.5} \log n)}$

Ordenados de forma ascendente seria: b, f, d, e, c, a