TP complejidad

Nombre : Palau Enzo

Ejercicio 1:

Podemos plantear que la expresión de la izquierda es $6n.n^2$ y la expresión de la derecha es $1.n^2$.

Por lo que vemos que para que sea una igualdad $6n=1$, lo cual es <u>falso</u> por lo que concluimos que $6n^3 \neq O(n^2)$

Ejercicio 2

Un array o lista que tuviera los elementos [10,11,12,13,14,1,2,3,4,5]
Ya que si se se elije el pivote para el mejor caso, por ejemplo el 10 en la primera parte tendrán las listas el mismo tamaño.

Ejercicio 3

En el caso del Merge-Sort seria de O(N.log n) ya que no depende de la entrada

En el Quick-sort si esta todo ordenado sera $O(n^2)$

En Insertion-Sort seria de O(n) ya que solo tendría que recorrer la lista

Ejercicio 4

```python
#Ejercicio 4
def OrdListaMenores(L):
    """ordena una lista donde siempre el elemento del medio contiene antes que él
    la mitad de los elementos menores que él."""
    if len(L)<=2:
        #si la lista es menor a 2 elementos no hay nada de que hacer
        return L
    else:
        #creo dos listas aux
        ListaResult=[]
        ListaAux=[]
        #defino los casos para listas de tamaño impar y par
        if len(L)%2==0:#si es par
            Pmedio=trunc(len(L)/2)-1
        else:#si es impar
            Pmedio=trunc(len(L)/2)
        Nmedio=L[Pmedio]
        Nmenores=cantidadDeMenores(L,Nmedio)

        #a continuacion dos casos especiales
        if Nmenores is 0:
            #si no tiene menores por ejemplo en una lista con un 0
            return L
        MitadMenores=round(Nmenores/2)
        if MitadMenores is 0:
            #en este caso la variable al redondear queda en 0 pero no significa que no tenga menores asiq pongo 1
            MitadMenores=1
```

```python
    #en este bloque voy a insertar en otra lista los elemtos correspondientes
    contador=0
    #en este pongo el numero de menores que corresponda
    #primero la mitad de los menores en una lista y los demas sin el del medio en otra
    for i in range(0,len(L)):
        if L[i]<Nmedio:
            contador+=1
            if contador<=(MitadMenores):
                ListaResult.append(L[i])
            else:
                ListaAux.append(L[i])
        elif L[i]>Nmedio:
            ListaAux.append(L[i])
    #luego relleno la primera con los mayores
    contador=0
    for i in range(0,len(L)):
        if L[i]>Nmedio:
            contador+=1
            if contador<=(MitadMenores):
                ListaResult.append(L[i])
                ListaAux.remove(L[i])
    #inserto el del medio y junto las listas
    ListaResult.insert(Pmedio,Nmedio)
    ListaResult=ListaResult+ListaAux
return ListaResult
```

```python
def cantidadDeMenores(L,Num):
    #da la cantidad de menores que el numero en la lista
    menores=0
    for i in range (0,len(L)):
        if L[i]<Num:
            menores+=1
    return menores
```

Explique la estrategia de ordenación utilizada:

Mas o menos explicado en los comentarios pero en resumen básicamente me baso en crear dos listas auxiliares para poner en una la mitad de lo menores y luego rellenarla con mayores .
Luego la otra lista contendrá los elementos restantes en el orden correspondiente para luego juntarla con la primera.

Ejercicio 5

```python
#Ejercicio 5
def ContieneSuma (A,n):
    """recibe una lista de enteros A y un entero n y devuelve True
     si existen en A un par de elementos que sumados den n."""
    #comparo cada elemento con los demas y sumo
    success=False
    for i in range(0,len(A)):
        for j in range(0,len(A)):
            suma=0
            if success==False:
                if i is not j :
                    suma=A[i]+A[j]
                if suma==n:
                    success=True
    return success
```

El costo computacional de mi implementación no requiere estructuras adicionales pero es un O(n^2)

Ejercicio 6:_

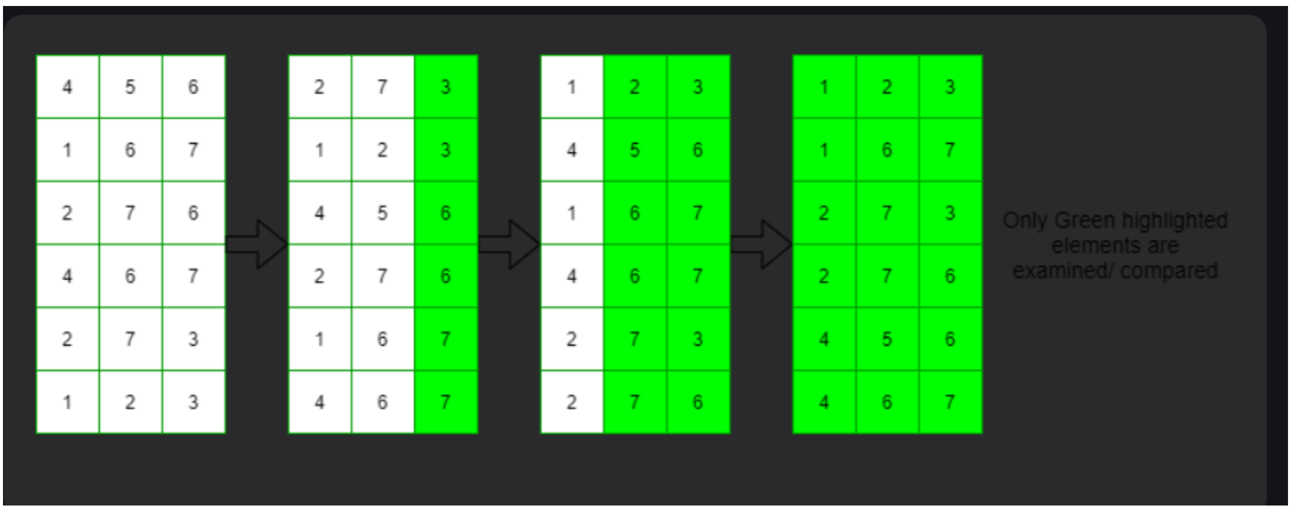
Radix-sort:

Este algoritmo ordena los elementos desde los menos significativos hasta los mas significativos, es decir ordena por unidades,decenas, centenas, etc.

Ventajas: es rápido cuando los números son chicos y es estable

Desventajas: en el aspecto “in place” no es bueno

Por ejemplo:



Complejidad:

O (d*(n+b)) d = número de dígitos, n = número de elementos del arreglo, b = rango de entrada

Pero en su mejor caso podríamos decir que es O(n)

Ejercicio 7

Realizo las primeras 3 con el método maestro completo:

a) $T(n) = 2T(n/2) + n^4$:

1) $\log_2(2) = 1 \rightarrow \Omega(n^{1+3}) = n^4$

2) Caemos en caso 3: $2(n/2)^4 \leq cn^4$ con $c < 1$

$\frac{n^4}{4} \leq \frac{1}{4} n^4 \rightarrow \Theta(n^4)$

b) $2T(7n/10) + n$:

$\log_{\frac{10}{7}}(2) = 1{,}94 \rightarrow O(n^{1{,}94 - 0{,}94}) = n$

caso 1: $\Theta(n^{\log_{10/7}(2)})$

c) $16T(n/4) + n^2$:

$\log_4 16 = 2 \rightarrow \Theta(n^2)$

caso 2: $\Theta(n^2 \log n) =$

Realizo las otras 3 con el método maestro simplificado:

D) T(n) = 7T(n/3) + n^2

Logb(a) =1,77 < 2

por lo que $T(n) = \Theta(f(n)) = \Theta(n^c)$ = $\underline{\Theta(n^2)}$

E) T(n) = 7T(n/2) + n²

2,80 > 2

por lo que $T(n) = \Theta(n^{\log_b(a)})$ = $\Theta(n^{\log_2(7)})$

F) T(n) = 2T(n/4) + √n

0,5 = ½

por lo que $T(n) = \Theta(f(n)) \lg n = \Theta(n^c \lg n)$ = $\Theta(\sqrt{n}\log n)$