Algoritmo y Estructura de Datos II

TP COMPLEJIDAD Estudiante: Joaquin Villegas

1- $6n^3 \neq O(n^2)$
Pensemos por definición y supongamos que el enunciado es falso.
Es decir que $6n^3 = O(n^2)$.
Se dice que T(n) es O(f(n)) si existen constantes positivas C y n_0 tal que:
$T(n) <= C(f(n))$ cuando $n >= n_0$
En este caso:
$T(n) = 6n^3$ y $f(n) = O(n^2)$

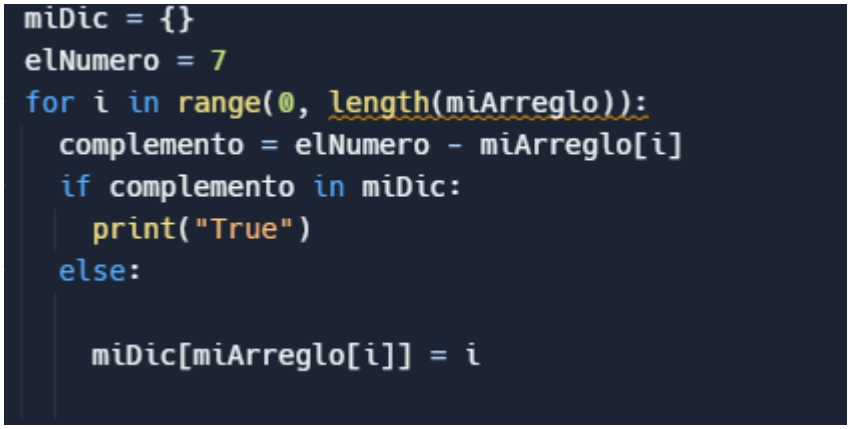
$6n^3 <= C * n^2$ y esto es un absurdo, porque hay valores de n donde n_0 se cumpla la desigualdad.
Entonces:
$6n^3 = O(n^3)$

2- Para lograr el mejor caso de QuickSort con minimo 10 elementos, es necesario que siempre los pivotes que se elijan sean los del medio, de manera que se pueda dividir el problema en partes equivalentes.

3- El tiempo de ejecución de quicksort, insertionsort y mergesort cuando los elementos tienen el mismo valor serán los siguientes:
Para MergeSort, no le importara si son iguales o no, lo hara en O(n log (n))
Para QuickSort, se comportara de manera inestable, tomando un tiempo de ejecución de $O(n^2)$
Para InsertionSort, su tiempo de ejecución será de Ω(n)

4- Pense en un algoritmo donde sacamos cual es el elemento del medio y su posición, luego crear una lista que contenga todos aquellos elementos que son menores que el elemento del medio. A esa lista nos fijamos cual es su mitad. Vamos a buscar los elementos de la lista en el array (sacar sus posiciones) y la comparamos con la posición del elemento del medio. Si es menor la posición, va a sumar un contador referidos a la izquierda, en caso contrario, un contador referidos a la derecha.
Si el contador de la izquierda superara la mitad de los elementos menores, significa que hay que reemplazar algunos con números mayores que el elemento del medio para cumplir con la consigna del ejercicio.

5-



```
miDic = {}
elNumero = 7
for i in range(0, length(miArreglo)):
  complemento = elNumero - miArreglo[i]
  if complemento in miDic:
    print("True")
  else:

    miDic[miArreglo[i]] = i
```

Costo computacional: O(n)

6-

BucketSort es un algoritmo de ordenamiento para números, se pueden llegar a encontrar implementaciones donde se trabaja en el intervalo [0, 1) (es decir entonces con números decimales) para que sea mas eficiente el algoritmo.

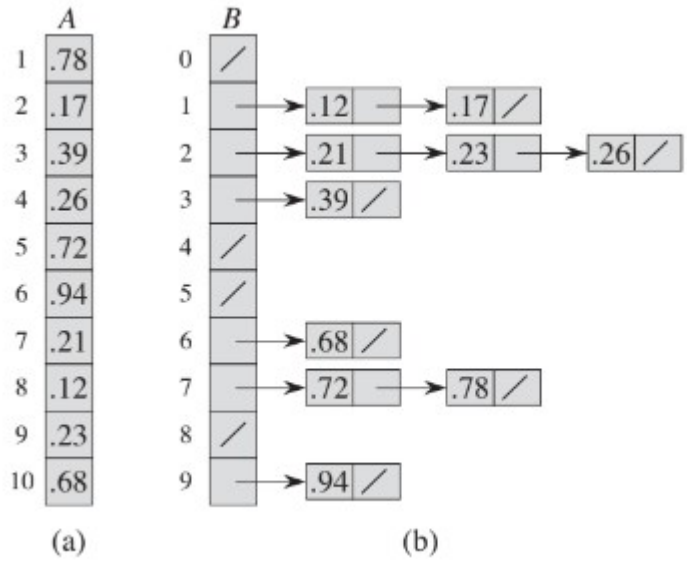
El algoritmo consiste en que recibe como entrada un arreglo (o lista) A con números desordenados, luego crearemos otro arreglo (o lista) B de 10 posiciones, que estos serán los conocidos “buckets”. A su vez, dentro de cada uno crearemos una lista de elemento.

Al recorrer la lista A, vamos a ir guardando cada numero en su respectivo bucket. Es decir, si yo tengo un elemento “0,73”, lo voy a tener que guardar en la lista de la posición 7. Si luego tengo un elemento “0.31” este será llevado a la posición y guardado también en la lista. Si después tengo un elemento “0,72” ira a la posición 7 y será guardado en su lista.

Asi con todos los elementos de la lista A. de forma tal que después tendremos que realizar en cada lista de los buckets otro algoritmo de ordenamiento (se utiliza frecuentemente el insertion sort).

Al final, lo que devolveremos será una lista definitiva en donde insertaremos cada elemento de cada lista. Quedaran ordenados ya que sabemos que hemos aplicado insertion sort y también van primero por ejemplo los del bucket 0, luego los del bucket 1, etc.

Un Ejemplo:



0,12	0,17	0,21	0,23	0,26	0,39	0,68	0,72	0,78	0,94

Ahora respecto a su complejidad temporal;

En el mejor caso, tiene una $\Omega(n)$ porque supongamos que divide los elementos en los buckets de manera equitativa, luego el insertion sort podría llegarse aplicar o no. De manera que terminamos con una complejidad lineal.

Luego para el caso promedio, $\Theta(n)$ llega a pasar aunque no esten del todo equitativos los buckets.

Por ultimo el peor caso, $O(n^2)$, ocurre cuando todos los elementos entran a un mismo bucket y de manera desordenada, generando asi que el insertion sort sea de $O(n^2)$.

7-

a) $T(n) = 2T\left(\frac{n}{2}\right) + n^4$ $a=2$ $b=2$ $f(n)=n^4$

$n^{\log_b a} = n = \Theta(n)$

Vemos que $n^4 \neq \Theta(n)$

Aplicamos el caso 3: $\Omega(n^{\log_b a + \epsilon}) = f(n)$

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

hay que corroborar si cumple con: $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

Para alguna constante $c < 1$ y todos los n suficientemente grandes.

$= 2 \cdot f\left(\frac{n}{2}\right)$

$= 2 \cdot \left(\frac{n}{2}\right)^4$

$= 2 \cdot \frac{n^4}{2^4} = \frac{n^4}{2^3} \leq c \cdot n^4$ con $c = \frac{1}{8} < 1$

$T(n) = \Theta(n^4)$

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

$n^{\log_{10/7} 2} = n^{1,94} = \Theta(n^{1,94})$

$f(n) \neq \Theta(n^{1,94})$

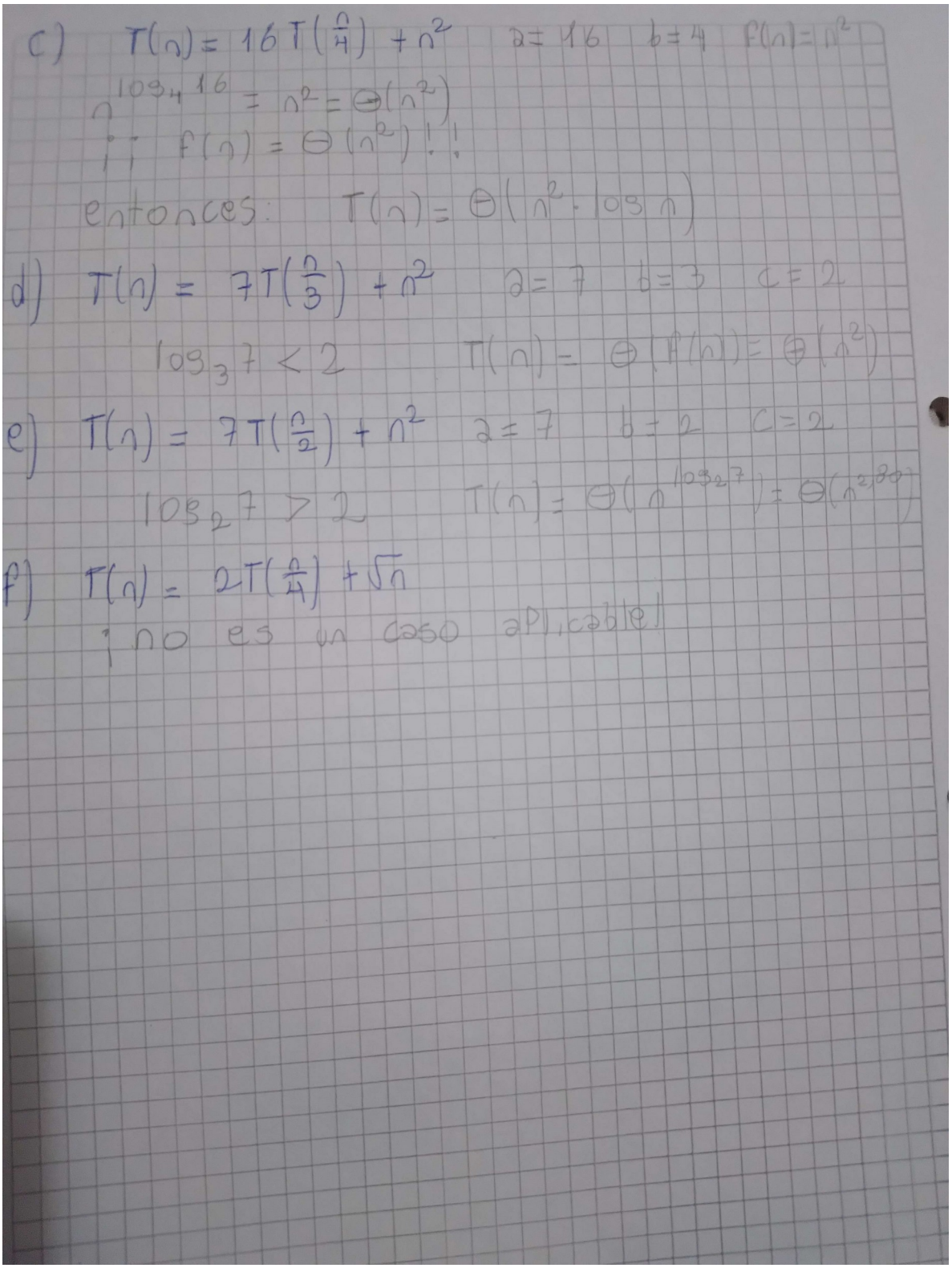
Utilizamos el caso 1: $f(n) = O(n^{\log_{b} a - \epsilon})$

Para alguna constante $\epsilon > 0$

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

entonces:

$T(n) = \Theta(n^{1,94})$



Entonces su orden ascendente seria:

b, d, c, e, a