UNCUYO UNIVERSIDAD NACIONAL DE CUYO — FACULTAD DE INGENIERÍA

NOMBRE: ALVARO BLANCO
LEGAJO: 10622
TRABAJO PRACTICO COMPLEJIDAD

BLANCO, Alvaro | TP Complejidad | 1
Legajo: 10622

Ejercicio 1

$6n^3 \neq O(n^2)$ Dividiendo por n^2 a ambos lados

$$\frac{6n^3}{n^2} \leq c\frac{n^2}{n^2} \Rightarrow 6n \leq c$$

Eligiendo un n lo suficientemente grande tenemos que 6n no puede estar acotado superiormente por una constante c y esto demuestra que $6n^3 \neq O(n^2)$. No existe cota superior $O(n^2)$ que contenga a $6n^3$

Ejercicio 2

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Esto permite lograr $O(n \log n)$ ya que cada eleccion de pivote dividira el arreglo en 2 subarreglos de igual tamaño y siempre la entrada n se dividira en 2 partes iguales y asi sucesivamente garantizando el comportamiento logaritmico.

Ejercicio 3

QuickSort → $O(n \log n)$ caso similar al ejercicio 2

InsertionSort → $O(n)$ no habra ningun intercambio, sin embargo debe recorrer todo el arreglo para verificar esto.

MergeSort → $O(n \log n)$ la parte de combinar los resultados no es necesaria ya que todos los subarreglos de longitud 1 son iguales y el tiempo de ejecucion depende de dividir los arreglos recursivamente.

Ejercicio 4: elegiría MergeSort, dividiendo la lista en 2 mitades ordenando por separado dichas mitades y luego combinamos los resultados. Las mitades serían de igual tamaño y la complejidad temporal sería $O(n \log n)$

Ejercicio 5: utilizamos MergeSort para ordenar la lista $O(n \log n)$ y luego tendríamos 2 punteros i, j con i al inicio de la lista y j al final y en cada iteración sumo los elementos en i y en j si es n, devuelvo TRUE en caso contrario si la suma es menor que n muevo el puntero i (inicio de la lista) a la derecha y si es mayor el puntero j a la izquierda. El costo total es $O(n)$

Ejercicio 6

RadixSort: ordena elementos por sus dígitos individuales, funciona similar a QuickSort o MergeSort pero en lugar de comparar elementos completos compara los dígitos individuales.

Suponiendo que se tiene [170, 45, 75, 90, 802, 24, 2, 66]

Primero buscamos el mayor de la lista 802 y contamos la cantidad de dígitos (3), luego ordenamos según el dígito menos significativo

[802, 2, 24, 45, 66, 170, 75, 90]

Luego repetimos para las decenas y centenas

[802, 2, 24, 45, 75, 66, 170, 90]

Finalmente [2, 24, 45, 66, 75, 90, 170, 802]

El mejor rendimiento es cuando el número de dígitos es relativamente pequeño en comparación con la cantidad de datos. $O(Kn)$ donde K es la cantidad mayor de dígitos, si hay muchos dígitos en algún número entonces K será grande y el algoritmo pierde rendimiento

BLANCO, Alvaro Legajo 106222	TP Complejidad	2

Ejercicio 1

a) $T(n) = 2T(n/2) + n^4$ con $a = 2$, $b = 2$ y $f(n) = n^4$

$n^{\log_b a} = n^{\log_2 2} = n^1$

C3: $n^{\log_2 2 + \epsilon}$ con $\epsilon > 0$ $n = f(n)$ con $\epsilon = 3$

$T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n) = \Theta(n^4 \lg n)$

b) $T(n) = 2T(7n/10) + n$ con $a = 2$, $b = \frac{10}{7}$ y $f(n) = n$

$n^{\log_b a} = n^{\log_{10/7} 2} \approx 1{,}943$

C1 con $\epsilon = 0{,}943$

$T(n) = \Theta(n^{1{,}943})$

c) $T(n) = 16T(n/4) + n^2$ con $a = 16$, $b = 4$, $f(n) = n^2$

$n^{\log_b a} = n^{\log_4 16} = n^2$

$n^{\log_b a} = f(n) \Rightarrow$ C2 $T(n) = \Theta(n^2)$

d) $T(n) = 7T(n/3) + n^2$ con $a = 7$, $b = 3$, $c = 2$

$\log_b a = 1{,}77$ como es menor a $c \Rightarrow$ Caso 3

$T(n) = \Theta(n^2)$ DIRECTO

e) $T(n) = 7T(n/2) + n^2$ con $a = 7$, $b = 2$ y $c = 2$

$\log_b a = 2{,}80$ como es mayor a $c \Rightarrow$ Caso 1

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{2{,}80})$ DIRECTO

f) $T(n) = 2T(n/4) + \sqrt{n}$ con $a = 2$, $b = 4$ y $c = 1/2$

$\log_b a = 0{,}5$ como es igual a $c \Rightarrow$ Caso 2

$T(n) = \Theta(n^{1/2} \lg n)$ DIRECTO.

EJERCICIO 4

```
def merge_sort_middle(lst):
    if len(lst) <= 1:
        return lst
    middle = len(lst) // 2
    left_half = lst[:middle]
    right_half = lst[middle:]
    left_half = merge_sort_middle(left_half)
    right_half = merge_sort_middle(right_half)
    return merge(left_half, right_half)

def merge(left_half, right_half):
    result = []
    i = j = 0
    while i < len(left_half) and j < len(right_half):
        if left_half[i] < right_half[j]:
            result.append(left_half[i])
            i += 1
        else:
            result.append(right_half[j])
            j += 1
    result += left_half[i:]
    result += right_half[j:]
    return result
```

EJERCICIO 5

```
def contiene_suma(A, n):
    A.sort()
    i = 0
    j = len(A) - 1
    while i < j:
        if A[i] + A[j] == n:
            return True
        elif A[i] + A[j] < n:
            i += 1
        else:
            j -= 1
    return False
```

EJERCICIO 6

```
def radix_sort(arr):
    max_num = max(arr)
    exp = 1
    while max_num/exp > 0:
        counting_sort(arr, exp)
        exp *= 10

def counting_sort(arr, exp):
    n = len(arr)
    output = [0] * n
    count = [0] * 10

    for i in range(n):
        index = arr[i] // exp
        count[index % 10] += 1
```

```python
for i in range(1, 10):
    count[i] += count[i-1]

i = n - 1
while i >= 0:
    index = arr[i] // exp
    output[count[index % 10] - 1] = arr[i]
    count[index % 10] -= 1
    i -= 1

for i in range(n):
    arr[i] = output[i]
```