Algoritmos y Estructuras de Datos III

Instructores: Diego Ihara, Marcos Villagra — Parcial #1
Fecha: 31-mar-2023

---

1. Conteste brevemente.

1. Sean f y g dos funciones que toman valores positivos tales que $g = O(f)$. Entonces, ¿es cierto que $f + g = \Theta(f)$?. Justifique su respuesta.
2. Si *Algoritmo 1* requiere tiempo $\Theta(n^2)$ y produce una salida de longitud $n\ log\ n$, y *Algoritmo 2* requiere tiempo $\Theta(n^2)$ y su salida es un numero c de k bits, donde k es una constante. ¿Cual es la complejidad de tiempo de otro algoritmo (*Algoritmo 3*) que ejecuta *Algoritmo 1* y luego aplica *Algoritmo 2* a la salida de *Algoritmo 1*? Explique brevemente.
3. Liste las siguientes expresiones en orden ascendente según la tasa de crecimiento. Es decir, si una función $g(n)$ es listada inmediatamente después de la función $f(n)$ en la respuesta, entonces debe ser el caso que $f(n)$ es $O(g(n))$.
   - $f_1(n) = 7^n$
   - $f_2(n) = n^{1/3}$
   - $f_3(n) = n^n$
   - $f_4(n) = log_2\ n$

Rúbrica

Criterio	0 Puntos	1 Punto	2 Puntos	3 Puntos
Pregunta 1	Respuesta incorrecta (F o V) o sin justificación.	Respuesta correcta (F o V) pero justificación incorrecta	Respuesta y justificación correcta	
Pregunta 2	Respuesta incorrecta o sin justificación.	Respuesta correcta pero justificación incorrecta	Respuesta y justificación correcta	
Pregunta 3	Dos o menos funciones en orden correcto.	Tres funciones en orden correcto	Todas las funciones en orden correcto	

Respuesta.

1. Se debe demostrar que $f + g = \Omega(f)$ y $f + g = O(f)$. Para $f + g = \Omega(f)$, para todo $n >= 0$, se tiene que $f(n) + g(n) >= f(n)$. Para $f + g = O(f)$, dado que $g = O(f)$ y $f = O(f)$ se puede citar la propiedad que dice que para funciones s, t, u si $t = O(u)$ y $s = O(u)$ entonces $t + s = O(u)$.
2. Dentro de *Algoritmo 3*, la entrada de *Algoritmo 2* es la salida de *Algoritmo 1* y es de longitud *nlogn*. Por este motivo, la "subrutina" *Algoritmo 2* requiere tiempo $\Theta(n^2\ log^2\ n)$. En total el tiempo que requiere *Algoritmo 3* es $\Theta(n^2) + \Theta(n^2\ log^2\ n)$ o $\Theta(n^2\ log^2\ n)$.

3. Orden ascendente: f_4, f_2, f_1, f_3. En el caso de f_3, para todo $n >= 7$ se da que $f_1 <= c \cdot f_3$.

2. Tablas de dispersión.

1. Se tiene una tabla de dispersión de tamaño $M = 11$. Se desea insertar 6 elementos: $3, 30, 9, 53, 35, 46$. La función de dispersión es $h(k) = k \ mod \ 11$. ¿Cuál es la tabla resultante si:
   (a) La exploración es lineal.
   (b) La exploración es cuadrática.
2. Se tiene una tabla de dispersión de tamaño $M = 13$ con doble hashing y funciones de dispersión $h_1(k) = k \ mod \ 13$ y $h_2(k) = 1 + (k \ mod \ 11)$. ¿Cuál es la tabla resultante al insertar los numeros $79, 69, 98$ y 72?

Rúbrica

Criterio	0 Puntos	1 Punto	2 Puntos	3 Puntos
1.a	La tabla tiene dos errores o más	La tabla tiene 1 error	La tabla no tiene errores	
1.b	La tabla tiene dos errores o más	La tabla tiene 1 error	La tabla no tiene errores	
2	La tabla tiene más de un error	La tabla tiene 1 error	La tabla no tiene errores	

Respuesta.

1. (a)

0	1	2	3	4	5	6	7	8	9	10
		35	3	46				30	9	53

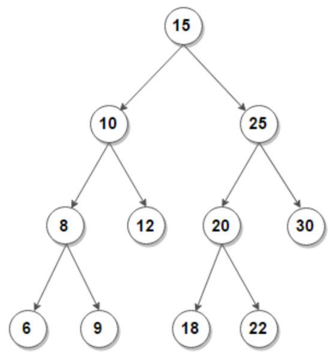
(b)

0	1	2	3	4	5	6	7	8	9	10
		35	3			46		30	9	53

2.

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			69			98	72				

3. Sea T un BST con n nodos y $[a, b]$ un rango con a, b números enteros positivos y $a < b$. Escribe un algoritmo de tiempo $O(n)$ que retorne el número de subárboles de T que posea nodos en $[a, b]$. Por ejemplo, para el BST de abajo y si consideramos el rango $[5, 20]$ tenemos los subárboles con raíces en 6, 9, 8, 10, 12, y 18.



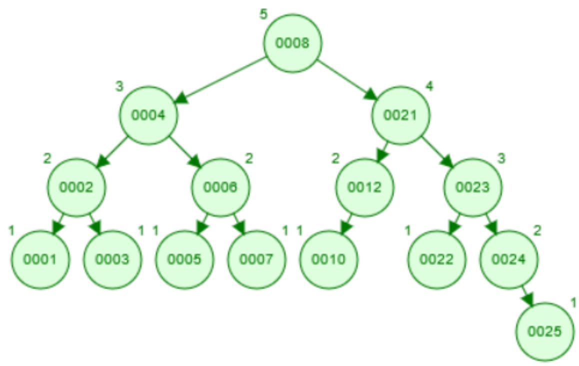
1. Construye un ejemplo de árbol AVL que posea 15 vértices y 5 subárboles para el rango $[5, 20]$.
2. Escribe un pseudocódigo para el problema solicitado. Presenta una explicación del funcionamiento correcto de tu algoritmo.
3. Presenta una explicación de la complejidad de tu algoritmo.

Rúbrica

Criterio	0 Puntos	1 Punto	2 Puntos	3 Puntos
Ejemplo	No presenta ningún ejemplo, o el ejemplo dado es incorrecto.	Presenta un ejemplo correcto pero no es un árbol AVL.	Presenta un ejemplo correcto de árbol AVL.	
Algoritmo	No presenta ningún pseudocódigo o el pseudocódigo no intenta resolver el problema o el pseudocódigo funciona para una minoría ($\leq$ %50) de las entradas o el algoritmo no cumple con la cota solicitada.	Presenta un pseudocódigo que cumple con la cota solicitada y retorna el resultado correcto en una mayoría ($>$ %50) de las entradas.	Presenta un pseudocódigo que retorna el resultado correcto en todas las entradas y el algoritmo cumple con la cota solicitada.	
Explicación	No presenta ninguna explicación del funcionamiento correcto de su algoritmo.	Escribe una explicación correcta del algoritmo presentado justificando su funcionamiento.		
Complejidad	No presenta ninguna justificación de la complejidad o la justificación es incorrecta.	Presenta una justificación correcta de la complejidad.		

Respuesta.

1. El árbol AVL de abajo[1] fue construido insertando los elementos 1 al 15 y 21 al 25 en orden. Luego se borraron vértices hasta obtener 15 vértices y 5 subárboles.



   Las raíces de los subárboles en el rango $[5, 20]$ son: 5, 7, 6, 10, 12.

2. Presentamos abajo el pseudocódigo solicitado. La idea es realizar un recorrido *in order* y mantener una variable booleana para indicar los subárboles que están o no en el rango.

```
//Subrutina de conteo de subárboles
//El input es un árbol binario T y un rango
SubTreeCount(T, rango)
begin
    if T == NIL then
        Return 1
    end-if
    l = SubTreeCount(T.left,rango)
    r = SubTreeCoutn(T.right,rango)
    if T.key >= rango[0] and T.key <= rango[1] and l and r then
        Return 1
    end-if
    Return 0
end
```

3. La complejidad del algoritmo `SubTreeCount` es lineal porque el recorrido es *in order* y todos los vértices son visitados exactamente una vez.

[1]El dibujo del árbol AVL fue hecho con `https://www.cs.usfca.edu/~galles/visualization/AVLtree.html`

4. Sea A un arreglo de tamaño n. Sabemos que el algoritmo `MergeSort` puede ordenar el arreglo A de menor a mayor en tiempo $O(n \log n)$. Queremos hacer una modificación a `MergeSort` para que además de ordenar el vector A, pueda retornar un conteo de el número de inversiones que hace `MergeSort` al ordenar A. Una inversión en A es un par de índices (i, j) tal que $i < j$ y $A[i] > A[j]$.

1. ¿Cuántas inversiones tiene el arreglo $A = [8, 4, 2, 1]$? Lista todas las inversiones.
2. ¿Cuánto es la complejidad temporal de un algoritmo de fuerza bruta que cuenta las inversiones en un arreglo de n elementos? Justifica tu respuesta.
3. Escribe un pseudocódigo de `MergeSort` y agrega la modificación que realiza el conteo. Presenta una explicación del porque esa modificación que realizaste logra correctamente el conteo de inversiones.

Rúbrica

Criterio	0 Puntos	1 Punto	2 Puntos	3 Puntos
Ejemplo	No resuelve el ejercicio o lo resuelve de forma incorrecta.	Presenta una lista correcta de todas las inversiones y su conteo correcto.		
Fuerza bruta	No responde a la pregunta sobre el algoritmo de fuerza bruta o la complejidad es incorrecta.	Escribe la complejidad correcta pero no presenta ninguna explicación o su explicación es incorrecta.	Escribe la complejidad correcta y la explicación es correcta.	
Pseudocódigo	No presenta ningún pseudocódigo de `MergeSort` o el pseudocódigo presentado no corresponde a `MergeSort`.	Presenta un pseudocódigo de `MergeSort` correcto.	Además de lo anterior, el cambio para el conteo solicitado es correcto.	Además de lo anterior, presenta una explicación correcta del porque la modificación realizada cuenta de forma correcta las inversiones.

Respuesta.

1. El arreglo tiene 6 inversiones y son (8, 4), (4, 2), (8, 2), (8, 1), (4, 1), (2, 1).
2. Un algoritmo de fuerza bruta debe de verificar todas las combinaciones tomadas de a dos en un arreglo de n elementos. Por lo tanto su complejidad es $O(\binom{n}{2}) = O(n^2)$.
3. Presentamos abajo un pseudocódigo de `mergesort` modificado para contar el número de inversiones que realiza.

```
//Subrutina estándar de MergeSort
MergeSort(A[1...n], conteo=0)
begin
    if n==1 then
        return A
    end-if
    A1 = MergeSort(A[1...n/2], conteo)
```

```
    A2 = MergeSort(A[n/2+1...n], conteo)
    //Merge retorna el conteo de inversiones en los vectores A1 y A2
    conteo_aux, A3 = Merge(A1,A2)
    //sumamos las inversiones de A1 y A2
    conteo = conteo + conteo_aux
    return conteo, A3
end

//Subrutina Merge modificada para contar inversiones
Merge(A1[1...n],A2[1...m])
begin
    j=0
    k=0
    B=[1...m+n]
    contador=0//contador de inversiones
    for i=0 to m+n-1 do
        if A1[j] <= A2[k] then
            B[i] = A1[j]
            j = j+1
        else//aquí ocurre una inversión
            B[i] = A2[k]
            k = k+1
            contador = contador+1
        end-if
    end-for
end
```

Las inversiones se detectan en la subrutina `Merge`. Sabemos que todos los elementos en `A1` preceden a los elementos en `A2`. Al intentar construir un vector ordenado en `Merge` una inversión se detecta cuando un elemento en `A2` es menor que un elemento en `A1`. Por lo tanto, incrementamos el contador de inversiones cuando detectamos este caso.