

Tarea 3

Programacion Declarativa

Palacios Gómez Esnesto Rubén
Peto Gutierrez Emanuel

15 de abril de 2018

1.-

Primero lo que hacemos es dividir todos los elementos del conjunto entre n llamaremos esta conjunto A'

$A' = \{b_1, b_2, \dots, b_n\}$ tal que $b_i = a_i/n$

Observamos que $0 < b_i \leq n$ por lo que podemos aplicar bucketSort a A' definimos los buckets como $\{[0, 1], (1, 2], (2, 3], \dots, (n-1, n]\}$

sea $A'' = \text{bucketSort}(A')$ ahora multiplicamos todos los elementos de A'' por n lo cual nos regresaria los elementos del conjunto original ordenados

Observamos que la primera parte del algoritmo es de orden $O(n)$ ya que pasa solo una vez por los elementos sabemos que bucketSort tiene orden de $O(n)$ y la ultima parte igual pasa solo una vez por todos los elementos por lo que tambien es de orden $O(n)$ ahora tenemos que la complejidad de este algoritmo es $O(n) + O(n) + O(n) = 3 \cdot O(n) = O(n)$ por lo que nuestro algoritmos tiene complejidad en tiempo lineal

2.-

primero ordenamos b con un algoritmo optimo por ejemplo heap sort

$B' = \text{heapSort } B$

esto nos da una complejidad de $O(n \log(n))$

Sean a b indice variables enteras , bandera una variable booleana para saber si encontramos el numero

```
bandera = false;
i = 0;
while (!bandera || i >= |A|){
    a = get(A, i);
    indice = busquedaBinaria(B', x-a);
```

```

        if ( indice != -1){
            bandera = true;
            b = get (B' , indice)
        }
        i++;
    }
    if(!bandera){
        a = -1;
        b = -1;
    }
    return (a,b);

```

Observamos que en el ciclo while en el peor de los casos (que es cuando no hay ninguna combinación que forme el número o que el ultimo número del conjunto A tenga un par en B que sumen x) entonces se ejecutara busqueda binaria $|A|$ -veces y sabemos que $|A| = n$ y que la complejidad de busqueda binaria es $O(\log(n))$ por lo que la complejidad de nuestro algoritmo seria $O(n \log(n)) + (n * O(\log(n))) = O(n \log(n)) + O(n \log(n)) = O(2n \log(n))$ pero las constantes se pueden quitar por lo que el algoritmo tendria una complejidad de $O(n \log(n))$

3.-

4.-

Primero recorremos el arreglo moviendo los verdes al inicio

```

int j = 0;
for (i = 0 ; i < n ; i++){
    if ( color (A , i ) = verde){
        swap (A , i , j) ;
        j++;
    }
}

```

Despues recorremos el arreglo a partir de j (al final del ciclo j tiene el lugar donde iria el primer rojo despues de todos los verdes)

```

for( i = j ; i < n ; i++){
    if(color(A , i ) = rojo){
        swap (A , i , j);
        j++;
    }
}

```

Observamos que en el primer ciclo se recorre todo el conjunto lo cual la complejidad es $O(n)$ y en el segundo ciclo se recorren $n - (\# \text{ de verdes})$ por lo

que el la complidad es $O(n-k)$ donde k es el # de verdes dentro del conjunto en el peor de los casos es que el conjunto no tenga verdes por lo que la complejidad en el peor de los casos es $O(n + n-0) = O(2n) = O(n)$ por lo que el algorimo es lineal

5.-

Obsevamos que el trinomio se puede calcular de la siguiente manera

$$(ax + b)(cx + d) = ac(x^2) + ((a + b)(c + d) - ac - bd)x + bd$$

basta calcular

$$A = ac$$

$$B = bd$$

$$C = (a+b)(c+d)$$

Y regresar:

$$A(x^2) + (C - A - B)x + B$$