Laboratorio 2

Santiago Álvarez Sepúlveda, *saalvarezse@unal.edu.co, Cód: 25481031*
Universidad Nacional de Colombia, Sede Bogotá

I. Problema I

Usando la Figura.1 como modelo, ilustre la operación de Insertion Sort en el arreglo $A = 31, 41, 59, 26, 41, 58$.

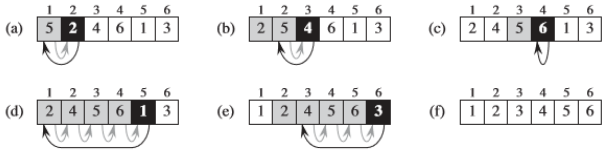

Figure 1. Ilustración del algoritmo Insertion Sort

Siguiendo las instrucciones del algoritmo Insertion Sort para el arreglo especificado en el enunciado 1, se tiene el siguiente resultado:

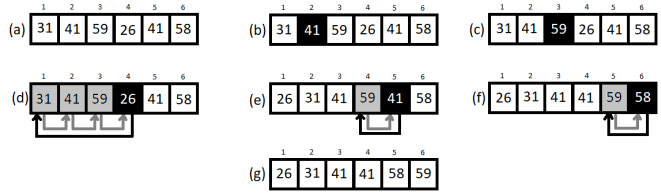

Figure 2. Ilustración del algoritmo Insertion Sort para el arreglo del enunciado del problema I.

En la anterior figura se observa el proceso desde el arreglo inicial (Figura.2.a), hasta el arreglo resultante (Figura.2.g).

II. Problema II

Reescriba el procedimiento Insertion Sort para ordenar de manera descendiente y no en orden ascendente.

Se requiere un algoritmo que ordene un arreglo de tamaño n por la técnica incremental de Insertion Sort. Modificando la condición del bucle *While* del pseudocódigo del algoritmo para ordenar de manera ascendente, de forma que la condición ahora sea $i > 0$ and $A[i] < key$, entonces se van a desplazar los elementos menores que el elemento actual hacia la derecha, posicionando los elementos mayores a la izquierda del arreglo, es decir al comienzo.

Algorithm 1 Insertion Sort Descendiente

```
1: procedure INSERTIONSORT(A)
2:     for j = 2 to A.length do
3:         key = A[j]
4:         i = j − 1
5:         while i > 0 and A[i] < key do
6:             A[i + 1] = A[i]
7:             i = i − 1
8:         A[i + 1] = key
```

III. Problema III

Considere el **problema de búsqueda**:

Entrada: Una secuencia de n enteros $A = a_1, a_2, ..., a_n$ y un valor v.

Salida: Un índice i tal que $v = A[i]$ o el valor especial *NIL* si v no existe en A.

Escriba un pseudocódigo para el algoritmo de **búsqueda lineal**, el cual recorre la secuencia buscando a v. Usando un bucle invariante pruebe que su algoritmo es correcto. Asegure que su bucle satisface las tres propiedades necesarias.

Algorithm 2 Linear Search

```
1: procedure LINEARSEARCH(A, v)
2:     key = NIL
3:     for j = 1 to A.length do
4:         if A[j] = v then
5:             key = j
       return key
```

A. Inicialización

Se debe demostrar que nuestro bucle invariante contiene el subconjunto $A[1, ..., j - 1]$. Al inicializar la rutina, nuestro bucle invariante consiste de el elemento NIL, el cual retornaremos en el caso que no exista v en el arreglo A. Esto demuestra que el bucle invariante se mantiene antes de iniciar la primera iteración del ciclo *for* del algoritmo.

B. Mantenencia

Cada iteración del algoritmo mantiene el bucle invariante ya que este va a cambiar su valor almacenado cuando se encuentre el elemento v en el arreglo A. De esta manera vemos que si se conserva el subconjunto de A durante todas las iteraciones del algoritmo.

C. Terminación

Vemos que al terminar las iteraciones el ciclo *for* del algoritmo, j va a tener el valor $(n+1)$, una posición del arreglo que no es posible acceder. Por esta razón en este punto de la ejecución no se bisca indexar de nuevo una posición del arreglo con este índice, sino retorna el valor inicializado de la posición de v en A, o NIL en su defecto.

IV. Problema IV

Considere el problema de sumar dos números enteros binarios de n bits, almacenados en dos arreglos de n elementos A y B. La suma de los dos enteros debería ser almacenada en forma binaria en un arreglo de $(n+1)$ elementos C. Desarrolle el problema formalmente y escriba pseudocódigo para sumar estos dos números.

Algorithm 3 Suma Binaria

```
1: procedure BINARYSUM(A, B, n)
2:     C = zeros(n + 1)
3:     while n > 0 do
4:         C[n + 1] = (A[n] ⊕ B[n]) ⊕ C[n + 1]
5:         C[n] = (A[n] ∧ (B[n] ∨ C[n])) ∨ (B[n] ∧ C[n])
6:         n − −
       return C
```

A. Inicialización

El anterior pseudocódigo asume que se tienen dos arreglos de bits A y B de tamaño n. Se inicializa con ceros un arreglo para almacenar el resultado de la suma C de tamaño $(n+1)$. Así se garantiza que se reserva memoria suficiente para almacenar el resultado final.

B. Mantenencia

En cada iteración del ciclo *While* del algoritmo estamos utilizando el subarreglo $C[n, n+1]$ y $A[n], B[n]$ los cuales existen para cada valor de n puesto que este valor no puede tomar valores menores a 1, posición inicial de todos los arreglos.

C. Terminación

Ya que se ha impuesto un decremento al parámetro n y una condición que haga que no se ejecuten más operaciones sobre elementos de los arreglos iniciales cuando este parámetro tenga un valor menor a 1, es decir que se van a realizar n veces las iteraciones en el ciclo *While* y al ser $n = 0$ se culminará la ejecución.