Nicolás Ramírez Calderón
257048

EJERCICIOS

- Referring back to the searching problem (see Exercise 2.1-3), observe that if the sequence *A* is Sorted, we can check the midpoint of the sequence against *v* and eliminate half of the sequence from further consideration. Binary search is an algorithm that repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive, for binary search. Argue that the worst-case running time of binary search is Θ(lg n).

```
binaryS ( A[], int v, int inicio, int fin)
{
        int mitad=0;
        while (inicio<=fin)
        {
                mitad=(inicio+fin)/2;
                if( v = =A[mitad] )
                        return mitad;
                else if(v > A[mitad] )
                        inicio=mitad+1;
                else
                        fin=mitad-1;
        }

        return -1;
}
```

Usando el teorema maestro para $T(n) = T(n/2) + 1$ se tiene que a=1 y b= 2. Haciendo $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$, el caso 2 del teorema maestro dice que:

$$T(n) = \mathrm{O}(1 \lg n) = O(\lg n)$$

- Use mathematical induction to show that when *n* is an exact power of 2, the solution of the recurrence.

$$T(n) = \begin{cases} 2 & \text{if } n = 2, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$
$$\text{is } T(n) = n \lg n.$$

El caso base es cuando n=2, entonces $n \lg n = 2 * \lg 2 = 2 * 1 = 2$
En el paso de inducción suponemos que $T(n/2 = (n/2) + \lg(n/2)$ entonces:

$$T(n) = 2T(n/2) + n$$
$$T(n) = 2(n/2) + \lg(n/2) + n$$
$$T(n) = n(\lg n - 1) + n$$
$$T(n) = n\lg n - n + n$$
$$T(n) = n\lg n$$

Por lo tanto se demuestra que $T(n) = n\lg n$

- Insertion sort can be expressed as a recursive procedure as follows. In order to sort *A*[1- *n*], we recursively sort *A*[1 _ *n* -1] and then insert *A*[*n*] into the sorted array *A*[1 _ *n* - 1]. Write a recurrence for the running time of this recursive version of insertion sort.

Se sabe que para un solo elemento se necesita solo un paso. La recurrencia dice que se tienen que ordenar los elementos y luego insertar A[n] en el arreglo.

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + n & n > 1 \end{cases}$$

La solución de esta recurrencia es $T(n) = O(n^2)$

PROBLEMAS

- Let *A*[1 _ *n*] be an array of *n* distinct numbers. If *i* < *j* and *A*[*i*] > *A*[*j*], then the pair (*i*, *j*) is called an *inversion* of *A*.

  a) List the five inversions of the array [2, 3, 8, 6, 1].
  b) What array with elements from the set [1, 2, …, n] has the most inversions? How many does it have?
  c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

  a) Los índices de las inversiones son (1, 5), (2, 5), (3, 4), (3, 5), (4, 5).
  b) El arreglo que tiene el mayor numero de inversiones es el que esta totalmente desordenado, es decir el arreglo {n, n-1, n-2,..,1}. El número total de inversiones de este arreglo es $\frac{n(n-1)}{2}$
  c) Cada iteración que realiza el ciclo while en insertionSort corresponde a la eliminación de una inversión en el arreglo.

- BubbleSort is a popular sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

```
BUBBLESORT(A)
1 for i ← 1 to length[A]
2     do for j ← length[A] downto i + 1
3           do if A[j] < A[j - 1]
4                 then exchange A[j] ↔ A[j - 1]
```

a) Let A' denote the output of BUBBLESORT(A). To prove that BUBBLESORT is correct, we need to prove that it terminates and that (2.3) where=length[A]. What else must be provef to show that BUBBLESORT actually sort?

b) State precisely a loop invariant for the for loop in lines 2-4, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter.

R//

a) También es necesario mostrar que lo elementos de A' son una permutación de los elementos de A.

b)

Invariante de loop: Al comienzo de cada iteración del ciclo for se tiene que A[j]=min{A[k];j<=k<=n}, y el subarreglo A[j..n] es una permutación del arreglo inicial en A[j..n].

Inicialización: Al iniciar el ciclo, j=n y el subarreglo A[j..n] consiste en apenas un elemento

Mantenencia: Se considera la iteración para cualquier valor de j. Por el invariante de loop se sabe que A[j] es el menor dentro del subarreglo A[j…n]. Al momento de hacer la comparación A[j]<A[j-1] si la condición se cumple se intercambian estos dos valores de manera que el nuevo arreglo desde A[j…n] sigue siendo una permutación de los valores iniciales del arreglo A[j..n].

Terminación: El ciclo termina cuando el valor de j sobrepasa al valor de i. Esto quiere decir que el valor de A[i]=min{A[k] ; i<=k<=n] y se puede afirmar que el arreglo A[i..n] es una permutación del arreglo inicial entre A[i…n], de tal manera que los elementos del arreglo están ordenados.