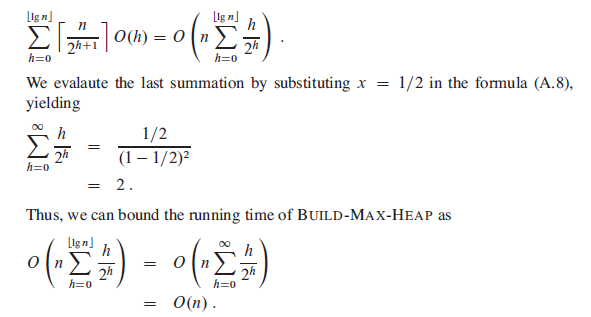
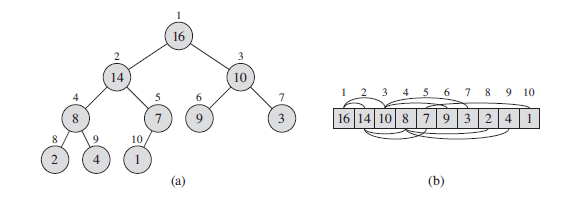
**Análisis de complejidad temporal y espacial.**

**HeapSort**

* El procedimiento MAX-HEAPIFY, que se ejecuta en O(log n), es la clave para mantener la propiedad max-heap.
* El procedimiento BUILD-MAX-HEAP, que se ejecuta en tiempo lineal, produce un maxheap desde una matriz de entrada desordenada.
* El procedimiento HEAPSORT, que se ejecuta en O (n log n), ordena una matriz en lugar.





**QuickSort**

* Peor de los casos:

El peor de los casos de Quicksort se produce cuando la rutina de partición produce

un subproblema con n- 1 elementos y uno con 0 elementos. Supongamos que esta partición desequilibrada surge en cada llamada recursiva. Los costos de partición n/ time. Desde la llamada recursiva en una matriz de tamaño 0 simplemente devuelve, T(0) = Ω(1) y la recurrencia de la ejecución tiempo es:

T(n) = T(n-1) + T(0) + Ω(n)

= T(n-1) + Ω(n)

* Mejor de los casos:

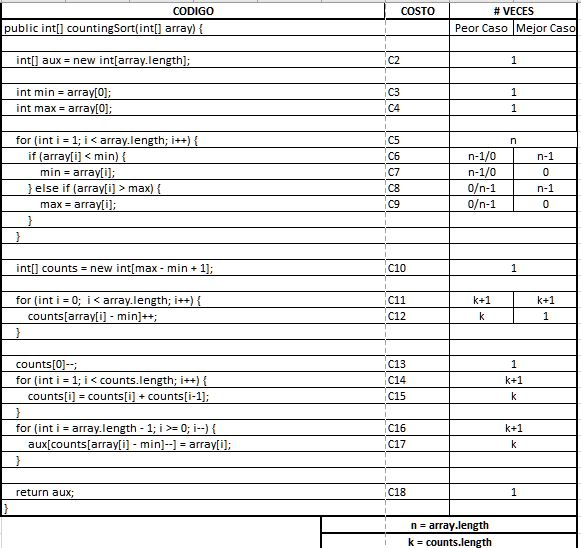
En la división más pareja posible, produce dos subproblemas, cada uno de tamaño no más que [n/2], ya que uno es de tamaño [n/2] y el otro de tamaño [n/2]-1. En esto caso, Quicksort funciona mucho más rápido. La recurrencia para el tiempo de ejecución es entonces:

T(n) = 2T (n/2) + Ω(n)

*Quicksort y Heapsort análisis tomado del Libro de Cormen, páginas 150 y 170*

**Counting Sort**

* **Compejidad temporal**



T(n) = 1+1+1+n+(n-1)+(n-1)+1 0 = n+n+n+3 = 3n+3

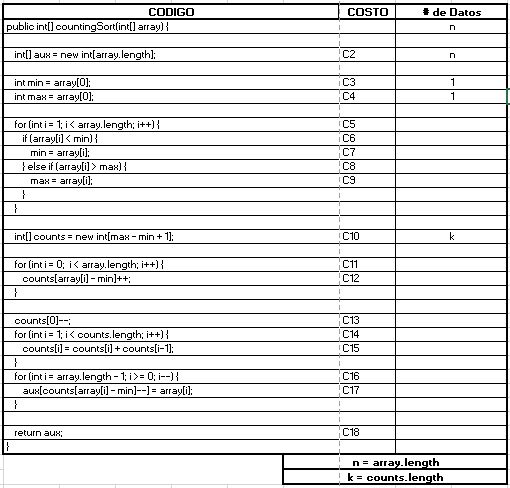
T(n) = Ω(n)

T(k) = 1+(k+1)+k+(k+1)+k+1 = k+k+k+k+4 = 4k+4

T(k) = Ω(k)

Complejidad temporal total = Ω(n) + Ω(k) = Ω(n+k) = **Ω(n).**

* **Complejidad espacial**



T(n) = n+n+1+1= n+n+3 = 2n+2

T(n) = Ω(n)

T(k) = k

T(k) = Ω(k)

Complejidad espacial total = Ω(n) + Ω(k) = Ω(n+k) = **Ω(n).**