Laboratorio 2 | Análisis Asintótico y Algoritmos de ordenamiento

Geordie Quiroa

August 9, 2018

1 - Algoritmo de ordenamiento

El algoritmo desarrollado en python (Heapsort.py) está adjunto en el path de este repositorio. Lo que este algoritmo genera, es un archivo .txt que permite comparar el array ordenado, el número de veces que se ejecutó cada línea, así como el total de iteraciones del programa como tal.

2 - Heap Verifier
| Algoritmo de verificación de heaps

En esta sección desarrollé un algoritmo que verifica si un arreglo representa o no un heap (adjunto en el path de este repositorio como heap-verifier.py).

Para esto tengo dos funciones > controlador-de-verificacion(possible-heap) y > evaluar-nodo-padre(possible-heap, i), en donde la función controlador-de-verificacion llama y envía ambos parámetros a la segunda función por medio de un ciclo while. Este ciclo while se va a ejecutar, en el peor de los casos, n/2 veces. Esto quiere decir que va a llamar la funcion que verifica si la llave del nodo i es mayor o igual que las llaves de los dos hijos (2xi, 2xi+1) n/2 veces. Sin embargo, a pesar de que la segunda función se ejecuta esa misma cantidad de veces, cada nodo, a excepción de la raíz y nodos hoja, es evaluado 2 veces; 1 vez cuando el nodo es comparado como hijo y la segunda vez cuando el nodo es padre. Por esta razón el algoritmo es de orden logarítmico: $\theta(\mathbf{nlg(n)})$, siendo n el número total de elementos. De la misma forma es posible concluir que la ejecución es en función de la altura del árbol ya que no se evalúan dos veces las llaves de los nodos hoja y root, de lo contrario sería un algoritmo de orden cuadrático.

3 - Conversión de Heapsort recursivo a iterativo

Pseudocódigo

Algorithm 1 Max-Heapify

```
1: procedure ITERATIVE-HEAPIFY(A, i)
2:     A is an array of integers
3:     for i ← 1 to (A.length/2) do
4:         l ← Left(i)
5:         r ← Right(i)
6:         if l <= A.length and A[l] > A[i] then
7:             largest ← l
8:         else
9:             largest ← i
10:        if r <= A.length and A[r] > A[largest] then
11:            largest ← r
12:        if largest != i then
13:            exchange A[i] ← A[largest]
```