UNIVERSIDAD FRANCISCO MARROQUIN

ALGORITMIA Y COMPLEJIDAD

Laboratorio 2

Alejandro Madrazo 20170478

August 10, 2018

1 PROBLEMA

Problema de Sorting
Supongamos que trabajan en una famosa empresa de tecnologia cuyo principal producto es un motor de busqueda. Para saber en que orden mostrar las paginas al hacer una busqueda, su trabajo es ordenar la lista de paginas web por su pagerank de mayor a menor. Hacer esto en un tiempo menor a n^2

$$Input: websitePageRanks = [3,39,61,91,57,22,75,89,9,90,63,78,28,73,20]$$

- En este ejercicio utilice QuickSort, revisar **quicksort.py** dentro de los archivos adjuntados.

2 PROBLEMA

Escriban un programa que verifique si el siguiente arreglo es un heap.

$$Input: possibleHeap = [16,14,10,8,7,9,3,2,4,1]$$

Demuestre el running time de este programa.

- Para encontrar si el arreglo es un heap, compare cada "root" con sus hijos nada mas. Si el root es mayor a sus hijos, se procede a comparar con el siguiente nodo. la cantidad de nodos es igual a: (n-2)/2, por eso si es mayor retorna.

- En este caso tuve que utilizar un "try" y "except" debido a que al comparar con el hijo a la derecha el indice excedia el largo del arreglo.

- La solucion se puede encontrar en el archivo: **isHeap.py**

Running time:
El running time de este programa es O(n).
Esto se debe a que se atraveisa el arbol binario comparando los "roots" o padres con sus hijos de nodo en nodo. Parece que solamente se recorre (n-2)/2 veces. Sin embargo estamos comparando a cada "root" con sus hijos en todos los nodos. Lo que nos resulta en una comparacion con todos los elementos del arreglo o arbol binario en este caso.

3 PROBLEMA

El codigo para MAX-HEAPIFY es muy eficiente en terminos de factores constantes, posiblemente exceptuando la linea 10, que podria causar que algunos compiladores produzcan codigo ineficiente. Escriban un algoritmo de MAX-HEAPIFY que utilize un constructo de control iterativo (loop) en lugar de recursion.

```
MAX-HEAPIFY(As,i)
.     while (true)
.         l=LeftChild(i)
.         r=RightChild(i)
.         if (l <= As.size ) and (As[l] > As[i])
.             biggest = l
.         else biggest = i
.         if (r <= As.size ) and (As[r] > As[biggest])
.             biggest = r
.         if biggest == i
.             jump
.         swap As[i]:As[biggest]
.         i = biggest
.     End Program
```