Lab3

Fernando González

16 de agosto de 2018

Problema 1

Algorithm 1 HeapSort

```
1: procedure HeapiSort(A)
2:     Initialization:
3:     NewHeap(A)
4:     i = 0
5:     j = i + 1
6:     for i in range len(A) do
7:         if A[i] < A[j] then
8:             Swap A[i] − > A[j]
9:             i = i + 1
10:        else if A[i] > A[j] then
11:            i = i + 1
12:        else if A[i] == A[j] then
13:            i = i + 1
14:        MaxHepify(A,i,j)
```

Mi algoritmo es menos ineficiente que el HeapSort, ya que trate de implementar el Algoritmo *QuickSort* para poder ordenar los nodos, en lugar de arruinar el Heap. Luego de pasar por todos los nodos, mi algoritmo hace un $MaxHeapify$ para verificar que mi árbol este bien ordenado, y de cierto modo *"perfeccionar"* el Heap.

Problema 2

1. El *running time* del QuickSort cuando todos los valores son iguales es $O(n) = n^2$, ya que sin importar el pivote que sea asignado, este algoritmo tendrá que pasar por todos los elementos de array.

3. El QuickSort es mas utilizado que el HeapSort, aunque su *runningtime* sea menos eficiente, ya que, como en el primer problema, el HepSort debe de hacer *Swaps ïnnecesarios"*, el QuickSort no. Y esto se da aun cuando los elementos ya están totalmente ordenados, el HeapSort tendrá que hacer estos *Swaps ïnnecesarios"* al ejecutarse.

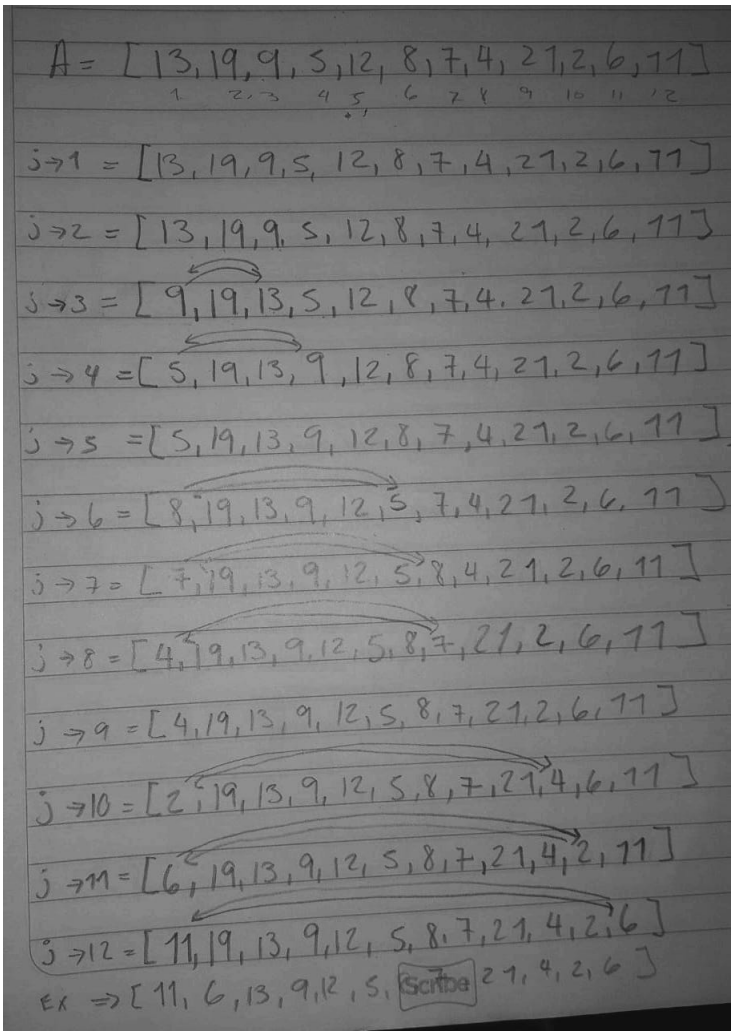

Figura 1: 2. Trace Partiton: $A_{final} = [9, 5, 8, 7, 4, 2, 6, 11, 12, 13, 19, 21]$