**Integrative Task#1 Computation and Discrete Structures 1**

**By: Juan Pablo Parra, Stick Martinez, and Daniel Gonzalez**

**Temporal analysis:**

**First Algorithm:**

Hash\_Insert(T,k)

j=0 → O(1) = 1

repeat

p = h(k,j) → O(1) = 1

if T[p] == NIL → O(1) = 1

T[p] = k → O(1) = 1

return j → O(1) = 1

else

j = j +1 → O(1) = 1

until j == m → O(n) = n

error “desbordamiento de tabla hash”

This is the temporal analysis of the algorithm of inserting an object into a hash table. In it you can see that its cost in terms of time complexity is O(n). Nevertheless, we can see that most of the cost per line is O(1) in fact in the best case that would be the cost, but as we always work with the worst case we must repeat the block until we can insert it, that is O(n) the size of the hash table.

**Second Algorithm:**

Max\_Heapify(A,i)

I = Izq(i) → O(1) = 1

D = Der(i) → O(1) = 1

If I <= A.heap-size and A[I] > A[i] → O(1) = 1

masGrande = I → O(1) = 1

else masGrande = I → O(1) = 1

if D I <= A.heap-size and A[D] > A[masGrande] → O(1) = 1

masGrande = D → O(1) = 1

if masGrande ≠ I → O(1) = 1

exchange A[i] with A[masGrande] → O(1) = 1

Max\_Heapify(A,masGrande) → O(n) = n

This is the temporal analysis of the algorithm of max heapify. In it you can see that its cost in terms of time complexity is O(log n). However, we can see that most of the cost per line is O(1) but being a recursive algorithm which can be repeated up to the stop case we say that it is O(n) however investigating in the book Introduction to Algorithms we are told that by the master theorem O(n) is the same as O(log n).

**Spatial Analysis**

**First Algorithm:**

Hash\_Insert(T,k)

j=0

repeat

p = h(k,j)

         if T[p] == NIL

                     T[p] = k

                     return p

         else

         j = j +1

until j == m

error “desbordamiento de tabla hash”

|  |  |  |
| --- | --- | --- |
| Tipo | Variable | Cantidad de Valores Atómicos |
| Entrada | T  k | 1  1 |
| Auxiliar | m | n |
| Salida | p | 1 |

Spatial Complexity Total = Input + Auxiliary + Output = n + 3 = O(n)

Spatial Complexity Auxiliary = n = O(n)

Spatial Complexity Auxiliary + Output = n +1 = O(n)

**Second Algorithm:**

Max\_Heapify(A,i)

I = Izq(i)

D = Der(i)

If I <= A.heap-size and A[I] > A[i]

         masGrande = I

else masGrande = I

if D I <= A.heap-size and A[D] > A[masGrande]

         masGrande = D

if masGrande ≠ I

         exchange A[i] with A[masGrande]

Max\_Heapify(A,masGrande)

|  |  |  |
| --- | --- | --- |
| Tipo | Variable | Cantidad de Valores Atómicos |
| Entrada | A  i | n  1 |
| Auxiliar | I  D  masGrande | 1  1  1 |
| Salida |  |  |

Spatial Complexity Total = Input + Auxiliary + Output = n + 4 = O(n)

Spatial Complexity Auxiliary = 3 = O(1)

Spatial Complexity Auxiliary + Output = 3 = O(1)