Laboratorio 1 — Juan Barillas

August 2, 2018

1 Problema 1

A continuación se encuentra mi pseudocódigo para realizar una búsqueda lineal

Algorithm 1 Búsqueda Lineal

```
Def busqueda (lista,x)
i = 1
for j in lista do

  if j == x: then
    return i
  end if
  i = i + 1
end for
return Null

Ejemplo de función
busqueda([4,8,25,21,10,42],21)
Outuput = 4
```

Mi loop invariant se encuentra en la posición de i pues esta indica que los índices anteriores no son igual al valor que buscamos, esto no cambia hasta encontrar el valor o hasta el siguiente loop invariant el cual indicaría que el número que buscamos no se encuentra en el array, en tal punto retorno un null y el este indica mi otro loop invariant la permanencia de no retornar null a menos que no exista el número.

2 Problema 2

Algorithm 2 Problema dos

```
Input: matriz A (n * m) y matriz B (m * p)
Output: matriz C (n * p)
for i from 1 to n:  do

  for j from 1 to p: do
    Let sum = 0

    for k from 1 to m: do
      Set sum ¡- sum + A[i][k] * B[k][j]
    end for
  end for
end for
```

El running time para una matriz es un tanto complicada para expresar. La multiplicación de dos matrices dan como resultado una nueva matriz que contiene la conexión de estas dos anteriores, es decir la progresión de las multiplicaciones. En el algoritmo 2 tenemos la matriz $A(n * m)$ y la matriz $B(m * p)$ durante el algoritmo en cada ciclo se repite una n cantidad de veces, p cantidad de veces y una m cantidad de veces en el peor de los casos en los que el algoritmo debe recorrerse, es decir se hace una conexión entre la multiplicación de la matriz A con la de B para poder multiplicar sus factores de $(n * p)$ Debido a su multiplicidad su running time se puede escribir como un $O(n * m * p)$ y siendo estas matrices perfectas (todas cuadradas) pueden tener un running time expresado por $O(n^3)$

3 Problema 3

Algorithm 1 Bubble sort algorithm

```
S is an array of integer
for i in 1 : length(S) − 1) do
   for j in (i + 1) : length(S)  do
      if S[i] > S[j] then
         swap S[i] and S[j]
      end if
   end for
end for
```

1) El worst-case running time con el algoritmo de Bubble sort es un $O(n^2)$ esto se debe a que el algoritmo compara con el entero siguiente una n cantidad de veces por el n números que se encuentran en el array. Esta descripción podemos verla desde el inicio en el primer ciclo que se recorrera $n$veces todo el tiempo al igual que su segundo ciclo que recorre n veces debido a que debe comparar siempre con su siguiente de $i + 1$ durante cada ciclo anterior. Por esta razón Bubble sort tiene un worst-case running time de $O(n^2)$

2) Insertion sort y bubble sort son diferentes en su algoritmo más sin embargo comparten un mismo worst-case scenario en el cuál ambos toman $O(n^2)$ el worst-case de Bubble sort fue explicado en el inciso 1 de este problema y el de insertion sort se da cuando los números a ordenar se encuentran completamente al revés debido a su algoritmo esto impide la optimización del proceso ya que debe hacer las comparaciones n^2 de veces como total justo al momento que terminar de ordenar la lista.
Por otra parte Insertion sort provee un mejor best-case scenario por su algoritmo que completaria en un running time de $O(n)$ debido a que estaria completamente ordenado o lo mayor ordenado posible.