Atividade B3-1-Cálculo tempo de execução-Insertion Sort

Nome: Vinícius Batista Crozato

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted
              sequence A[1 .. j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

Esse algoritmo "Insertion-Sort" poderá haver dois casos, um caso em que o array está ordenado e outro caso onde o array está em ordem decrescente.

Ordenado:

Começando com o loop *for* que executa 'n-1' vezes, sendo 'n' o tamanho do array.

No comando '*key = A[j]*' são executadas 'n-1' vezes.

Nesse caso, o algoritmo não executará o loop *while* , dessa forma, pulando direto para '*A[i + 1] = key*'.

Sendo assim contabilizando o número total de operações, temos:

T(n) = 2(n - 1) = *O*(n).

Ordem Crescente:

Neste caso, a cada passo do *for* o *while* executa o máximo de iterações possíveis.

No comando '*key = A[j]*' são executadas '*n-1*' vezes.

No *while* para cada '*j*', o *while* executa '*j-1*' vezes.

E para cada iteração do 'while', tem uma comparação '*A[i] > key*' e uma atribuição '*A[i + 1] = A[i]*'.

Dessa forma, contando as iterações, tanto do *for* quando do *while*, temos:

$T(n)=\sum^{n}_{j=2}(j-1)$

Podendo usar a fórmula da soma de uma série aritmética:

$\sum^{n}_{j=2}(j-1) = \sum^{n-1}_{j=1} = (n-1) * n / 2$

Simplificando:

$T(n)=n^2 - n / 2$

Com esse termo dominante, temos uma complexidade de tempo de:

$T(n)=O(n^2)$

Sendo assim, o Insertion-Sort, é eficiente para arrays pequenos ou ordenados minimamente, mas possivelmente será ineficiente para arrays grande em ordem inversa.