Lista 02 de ATC

Em aula foram dados dois algoritmos de ordenação, o Insertion Sort e o Merge Sort.

O Insertion Sort, um algoritmo mais simples, supõe que o vetor está ordenado da posição inicial até uma certa posição n. O procedimento então pega o elemento da próxima posição e coloca em seu lugar devido na parte ordenada do vetor, deixando o vetor ordenado até a posição $n+1$. Repetindo este procedimento com n variando de 1 até o tamanho do vetor, o resultado é o vetor todo ordenado.

O Merge Sorte usa a técnica de divisão-e-conquista, o algoritmo divide o vetor ao meio, em dois vetores de tamanho igual, ordena cada um deles, depois junta os dois, de forma ordenada, em um terceiro vetor.

Os dois algoritmos são mostrados a seguir, na forma de um pseudo-código (O algoritmo Merge é usado pelo Merge-Sort). Encontre o tempo de execução de pior caso do Insertion-Sort e do Merge-Sort, em termos da notação O, em relação ao tamanho do vetor A. **Escreva o raciocínio que você usou.**

Algoritmo 1: Merge-sort(A, p, r)

Entrada: Um vetor A, a posição do início do vetor p e a posição final r.
Saída: O vetor A ordenado da posição p até a posição r.

```
1.1 se r − p < 2 então
1.2     return
1.3 q = ⌊(p + r)/2⌋
1.4 Merge-Sort(A, p, q)
1.5 Merge-Sort(A, q + 1, r)
1.6 Merge(A, p, q, r)
```

Algoritmo 2: Merge(A, p, q, r)

Entrada: Um vetor A, a posição do início do vetor p, a posição do meio q e a posição final r.

Saída: O vetor A ordenado da posição p até a posição r.

```
2.1  n_1 = q − p + 1
2.2  n_2 = r − q
2.3  Seja L[1..n_1 + 1] e R[1..n_2 + 1] novos vetores
2.4  para i = 1 até n_1 faça
2.5      L[i] = A[p + i − 1]
2.6  para j = 1 até n_2 faça
2.7      R[j] = A[q + j]
2.8  L[n_1 + 1] = ∞
2.9  R[n_2 + 1] = ∞
2.10 i = 1
2.11 j = 1
2.12 para k = p até r faça
2.13     se L[i] <= R[j] então
2.14         A[k] = L[i]
2.15         i = i + 1
2.16     senão
2.17         A[k] = R[j]
2.18         j = j + 1
```

Algoritmo 3: Insertion-Sort(A)

Entrada: Um vetor A de tamanho n.

Saída: O vetor A ordenado.

```
3.1  para j = 2 até n faça
3.2      key = A[j]
3.3      i = j − 1
3.4      enquanto i > 0 e A[i] > key faça
3.5          A[i + 1] = A[i]
3.6          i = i − 1
3.7      A[i + 1] = key
```