Lista Exercícios

1. Usando como referência a explicação do algoritmo a seguir, faça uma função que ordene um *array* com 50 elementos inteiros. Informe a complexidade de pior caso do algoritmo desenvolvido.

O algoritmo percorre o *array* e a cada comparação entre os elementos, verifica e, caso necessário, troca o maior elemento com o menor. Ao chegar na última posição do *array* e após realizar a verificação final, o menor elemento estará na primeira posição da lista. Depois, percorre o array, realiza as comparações e caso necessário faz as trocas. Ao alcançar o final desta rodada, o segundo menor valor estará na segunda posição do *array.* Este procedimento é realizado sucessivamente, até que a lista esteja toda classificada em ordem crescente.

2. Usando como referência a explicação do algoritmo a seguir, faça uma função que ordene um *array* com 20 elementos inteiros. Informe a complexidade de pior caso do algoritmo desenvolvido.

O algoritmo percorre todo o array e a cada comparação entre os elementos, verifica e, caso necessário, marca o menor valor da lista. Ao chegar no último elemento e após realizar a verificação final, coloca o menor elemento selecionado na primeira posição da lista. Depois, percorre o array e seleciona o segundo menor valor da lista seguindo a mesma regra, para colocá-lo na segunda posição. Faz isso com o terceiro menor valor da lista e assim sucessivamente até que todos os elementos da lista sejam usados e que a lista fique classificada em ordem crescente.

3. Considere um algoritmo de ordenação que percorra um *array* de números inteiros da esquerda para a direita conforme o esquema de funcionamento a seguir.

1. | 3 | 5 | 4 | 2 | 1 |
2. | 3 | 5 | 4 | 2 | 1 |
3. | 3 | 4 | 5 | 2 | 1 |
4. | 2 | 3 | 4 | 5 | 1 |
5. | 1 | 2 | 3 | 4 | 5 |

- Verifica-se se o elemento que está na 2ª posição é menor do que o que está na 1ª posição, ou seja, se o 5 é menor do que o 3. Como não é, então não há troca.
- Verifica-se se o elemento que está na 3ª posição é menor do que o que está na 2ª posição e na 1ª posição. Como o 4 é menor do que o 5 mas não é menor do que o 3, então o 5 troca de lugar com o 4.
- Verifica-se se o elemento que está na 4ª posição é menor do que os elementos anteriores, ou seja, se o 2 é menor do que o 5, do que o 4 e do que o 3. Como ele é menor do que 3, então o 5 passa a ocupar a posição do 2, o 4 ocupa a posição do 5 e o 3 ocupa a posição do 4, assim a posição do 3 fica vazia e o 2 passa para essa posição.
- Este processo se repete até que todos os elementos do *array* estejam ordenados.

Usando como referência a explicação anterior do algoritmo de ordenação, que parece com o embaralhamento de cartas, faça uma função que ordene um *array* com 10 elementos inteiros. Informe também a complexidade de pior caso do algoritmo criado.

4. Relacione V (verdadeiro) ou F (falso) para cada alternativa abaixo. **Caso a alternativa seja falsa, justifique o erro. Não serão consideradas as alternativas sem justificativa ou com justificativa incorreta**.

( ) Os algoritmos de classificação *InsertionSort*, *MergeSort* e *QuickSort* têm uma eficiência melhor se comparados aos algoritmos *SelectionSort* e *BubbleSort*. Essa afirmação é válida, pois a complexidade de pior caso dos três primeiros é O(n log n) enquanto dos dois últimos é $O(n^2)$.

( ) Apesar de ser mais eficiente, o algoritmo QuickSort possui a mesma complexidade de pior caso dos algoritmos BubbleSort e InsertionSort, ou seja, $O(n^2)$.

( ) Tanto o algoritmo *MergeSort* quanto o *QuickSort* utilizam a técnica chamada divisão e conquista para subdividir os elementos do array que será ordenado. A principal diferença entre eles é que o *MergeSort* concatena as soluções obtidas para chegar à solução final, enquanto que o *QuickSort* faz intercalações.