sort

Bubble sort e um algoritmo simples de facil entendimento e implementacao, no entanto,
nao e um algoritmo eficaz pois sua enficiencia diminui radicalmente conforme aumenta o numero de elementos no array.
Seu melhor caso e quando os elementos ja estao ordenados.
Ja seu pior caso e quando os elementos estao ordenados em ordem inversa.

O algoritmo **Insertion Sort** tem como seu melhor caso os elementos ja ordenados e como
pior caso os elementos ordenados em ordem inversa tendo em vista que eh feito uma analise de elemente em elemento antes de inserir o elemento em uma posicao.

No **Quick Sort** o melhor caso ocorre quando as particoes possue sempre o mesmo tamanho,
ou seja, estao sempre balanceadas (NlogN)
O pior caso ocorre quando a funcao de particao calcula um pivo que divide o array de N elementos em dois:
uma particao com N-1 elementos e outra com 0 elementos.
Nesse caso teremos um particionamento nao balanceado. Quando isso acontece a cada nivel da recursao
temos um tempo de execucao o(N^2)

Selection sort e um algoritmo que divide o array em duas partes:
A parte ordenada que fica a esquerda do elemento analisado e a
parte que ainda nao foi ordenada que fica a direita desse elemento.
Pra cada elemento do array ele procura o menor valor a sua direita
e troca os dois valores de lugar.
Por esse algoritmo sempre comparar um elemento com os outros
a cada iteracao procurando o menor, nao existe um melhor caso
pois mesmo o vetor estando ordenado ou em ordem inversa,
sempre vai ser executado os dois lacos do algoritmo. Tambem
sendo muito lento para vetores grandes.
No entanto, por nao necessitar de um vetor auxiliar pra realizar a

ordenacao, ele ocupa menos memoria e e extremamente rapido
na ordenacao de vetores pequenos.

Bubble Sort
→ MELHOR CASO: array ordenado e com um baixo numero de quantidades.
para iniciantes é um bom método de facil compreenção alem de facilidade para implementação.
→ PIOR CASO: array como foi desenvolvido (1mi de posições) e deserdenado

- → MINHAS OBSERVAÇÕES: dentre os algoritmos de ordenação foi o algoritmo com pior desempenho, tanto em tempo
  como em quantidades de trocas, além de consumir uma quantidade absurda de uso em CPU
- → MEU CODIGO: utilizando apenas os dois laços para conferir a ordenação não estava ordenando corretamente,
  após eu ter imprementado o 'do while' houve uma melhora de 100% em ordenação.

```
Selection Sort
```

- → MELHOR CASO: array pequeno, baixa quantidade de trocas.
- → PIOR CASO: ele faz a mesma quantidade de comparação que o tamanho do vetor
- → MINHAS OBSERVAÇÕES: foi o segundo pior caso em algoritmos de ordenação, mas houve uma melhora sigificativa em quantidades de trocas, se comparar com o Bubble Sort.
- → MEU CODIGO: basico e funcional

```
Insertion Sort
```

- >MELHOR CASO: elementos mais proximos do metodo de ordenação possivel.
  -> PIOR CASO: elementos ordenados de formas ivarsas ao que foi solicitado.
  -> MINHAS OBSERVAÇÕES: foi o algoritmo com menor tempo de execução
  -> MEU CODIGO: basico, não consegui reduzir os gastos computacionais

```
Quick Sort
```

- → O melhor caso ocorre quando as particoes possue sempre o mesmo tamanho, ou seja, estao sempre balanceadas (NlogN)
- → O pior caso ocorre quando a funcao de particao calcula um pivo que divide o array de N elementos em dois:
  uma particao com N-1 elementos e outra com 0 elementos.
  Nesse caso teremos um particionamento nao balanceado. Quando isso acontece a cada nivel da recursao
  temos um tempo de execucao o(N^2)