

# Análise e Otimização de Algoritmos de Ordenação: Bubble Sort, Selection Sort e Insertion Sort

## 1. Implementação dos Algoritmos:

- Implemente os seguintes algoritmos de ordenação:
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
- Cada algoritmo deve ser implementado em sua versão não otimizada inicialmente.

## 2. Otimização dos Algoritmos:

- **Bubble Sort:** Implemente uma versão otimizada, onde o algoritmo verifica se houve trocas durante uma passagem pela lista. Caso não haja trocas, o algoritmo deve ser interrompido, já que a lista estará ordenada.
- **Selection Sort:** Otimize a busca pelo menor valor. Em vez de buscar o menor valor de todo o restante da lista para cada iteração, faça isso de forma eficiente para reduzir o número de comparações.
- **Insertion Sort:** Implemente uma versão otimizada utilizando a técnica de "binary insertion", que reduz a complexidade de encontrar a posição de inserção.

## 3. Testando e Comparando os Algoritmos:

- Gere uma lista de números aleatórios para ser ordenada por cada algoritmo. Use listas com diferentes tamanhos (ex: 100, 1000, 10000) e verifique o tempo de execução para cada uma das versões (não otimizada e otimizada).
- Utilize a função time para medir o tempo de execução de cada algoritmo.

## 4. Análise de Complexidade:

- **Bubble Sort:** Discuta a complexidade do algoritmo no pior caso, melhor caso e caso médio. Compare o desempenho da versão otimizada com a versão não otimizada.
- **Selection Sort:** Discuta a complexidade do algoritmo e como a otimização impacta a quantidade de trocas realizadas.
- **Insertion Sort:** Comente sobre o comportamento do algoritmo em listas parcialmente ordenadas e discuta como a otimização pode melhorar o tempo de execução.

## 5. Relatório:

- Faça um relatório que inclua:
  - O código dos algoritmos implementados (Utilizar [Carbon](#) para formatar).
  - A linguagem de programação utilizada.
  - A descrição da otimização aplicada em cada algoritmo.
  - Uma tabela ou gráfico comparando os tempos de execução dos algoritmos não otimizados e otimizados para diferentes tamanhos de listas.
  - Uma análise detalhada das complexidades (teórica e prática) de cada algoritmo e como as otimizações impactaram os resultados.

### Algoritmos não otimizados:

#### Selection Sort


```
ALGORITMO SelectionSort(array)
  n ← tamanho do array

  PARA i DE 0 ATÉ n - 1 FAÇA
    min_index ← i

    PARA j DE i + 1 ATÉ n - 1 FAÇA
      SE array[j] < array[min_index] ENTÃO
        min_index ← j
      FIM SE
    FIM PARA

    trocar array[i] e array[min_index]
  FIM PARA
FIM ALGORITMO
```


#### Bubble Sort



```
ALGORITMO BubbleSort(array)
  n ← tamanho do array

  PARA i DE 0 ATÉ n - 2 FAÇA
    PARA j DE 0 ATÉ n - 2 - i FAÇA
      SE array[j] > array[j + 1] ENTÃO
        trocar array[j] e array[j + 1]
      FIM SE
    FIM PARA
  FIM PARA
FIM ALGORITMO
```

## Insertion Sort



```
ALGORITMO InsertionSort(array)
  n ← tamanho do array

  PARA i DE 1 ATÉ n - 1 FAÇA
    chave ← array[i]
    j ← i - 1

    ENQUANTO j ≥ 0 E array[j] > chave FAÇA
      array[j + 1] ← array[j]
      j ← j - 1
    FIM ENQUANTO

    array[j + 1] ← chave
  FIM PARA
FIM ALGORITMO
```