Algoritmos de Ordenação

Ordenação por Inserção e Seleção

ORDENAÇÃO POR INSERÇÃO

Ordenação por Inserção

- Ordenação por inserção é um algoritmo simples que organiza a lista final um item de cada vez;
- Percorre o vetor da esquerda para a direita.
- Cada elemento é comparado com os anteriores e colocado na posição correta.
- Começa no segundo elemento, compara com o primeiro e insere na posição certa.
- Continua com o terceiro elemento, compara com os dois primeiros, e assim por diante.
- Eficiente para listas pequenas ou quase ordenadas.
- Usado em algoritmos mais complexos, como o Timsort.

Ordenação por Inserção

Algoritmo - VISUALG

```
algoritmo "Ordenação por Inserção"
var
    i, j, chave: inteiro
    vetor: vetor[1..5] de inteiro

inicio
    vetor[1] := 2
    vetor[2] := 90
    vetor[3] := 9
    vetor[4] := 18
    vetor[5] := 46

    para i de 2 ate 5 faca
        chave := vetor[i]
        j := i - 1

        enquanto (j >= 1) e (vetor[j] > chave) faca
            vetor[j + 1] := vetor[j]
            j := j - 1
        fimenquanto

        vetor[j + 1] := chave
    fimpara

    para i de 1 ate 5 faca
        escreva(vetor[i], " ")
    fimpara
fimalgoritmo
```

Código em GO

```
import "fmt"

func insertionSort(vetor []int) {
    for i := 1; i < len(vetor); i++ {
        chave := vetor[i]
        j := i - 1

        for j >= 0 && vetor[j] > chave {
            vetor[j+1] = vetor[j]
            j--
        }

        vetor[j+1] = chave
    }
}

func main() {
    vetor := []int{12, 61, 33, 5, 1}

    insertionSort(vetor)

    for _, valor := range vetor {
        fmt.Printf("%d ", valor)
    }
}
```

Ordenação por Inserção

- Se o vetor não está ordenado, há um par de elementos consecutivos fora de ordem.
- Trocamos esses pares invertidos, percorrendo do fim para o começo do vetor.
- Uma única passada pode não ser suficiente.
- Após cada passada, o menor elemento estará em sua posição correta.
- Realizamos n - 1 passadas para ordenar o vetor, onde n é o tamanho do vetor.

ORDENAÇÃO POR SELEÇÃO

Ordenação por Seleção

- Percorre os elementos e seleciona o menor.
- Troca esse elemento com o da extremidade esquerda.
- Organiza os elementos à esquerda.
- Número de trocas reduzido de N^2 para N, mas comparações permanecem em N^2.
- Mais adequado para pequenos conjuntos de dados.
- Menos eficiente para grandes conjuntos.

Ordenação por Seleção

Algoritmo - VISUALG

```
algoritmo "OrdenacaoPorSelecao"
var
   vetor[5]: inteiro
   i, j, minIndex, temp: inteiro
inicio
   vetor <- [53, 29, 15, 21, 3]

   para(i <- 0 ate 4 faca
      minIndex <- i

      para(j <- i+1 ate 4 faca
         se(vetor[j] < vetor[minIndex]) entao
            minIndex <- j
         fimse
      fimpara

      temp <- vetor[i]
      vetor[i] <- vetor[minIndex]
      vetor[minIndex] <- temp
   fimpara

   para(i <- 0 ate 4 faca
      escreva(vetor[i], " ")
   fimpara
fimalgoritmo

```

Código em GO

```
package main

import "fmt"

func selectionSort(vetor []int) {
    tamanho := len(vetor)
    var minIndex, temp int

    for i := 0; i < tamanho-1; i++ {
        minIndex = i
        for j := i + 1; j < tamanho; j++ {
            if vetor[j] < vetor[minIndex] {
                minIndex = j
            }
        }
        temp = vetor[i]
        vetor[i] = vetor[minIndex]
        vetor[minIndex] = temp
    }
}

func main() {
    vetor := []int{69, 23, 11, 5, 45}
    selectionSort(vetor)
    fmt.Println(vetor)
}

```

Ordenação por Seleção

•O código começa com a definição da função “selectionSort”, que recebe um slice de inteiros como entrada.

•Ele inicializa variáveis como “tamanho” para armazenar o tamanho do slice, “minIndex” para o índice do menor elemento e “temp” para a troca temporária de elementos.

•Em um loop “for”, itera sobre o slice, exceto pelo último elemento.

•Dentro desse loop, inicializa “minIndex” como o índice atual.

•Em outro loop “for”, percorre o slice a partir do próximo elemento.

•Se o próximo elemento for menor que o atualmente considerado o menor, atualiza “minIndex” para o índice desse novo menor elemento.

• Após encontrar o menor elemento, realiza a troca entre o elemento atual e o menor encontrado.

• Esse processo é repetido até que todos os elementos estejam ordenados.

• Na função “main”, um slice desordenado é definido.

• A função “selectionSort” é chamada com esse slice como argumento.

• Por fim, o slice ordenado é impresso.

FIM