GSI011 – Estrutura de Dados 2

Profª: Christiane Regina Soares Brasil

GSI011 – Estrutura de Dados 2 – Aula 5

- Ordenação
  - Tipos:
    - Numérica;
    - Lexicográfica.
  - Pode ser:
    - Crescente;
    - Não decrescente;
    - Decrescente;
    - Não crescente.

- Pode ser:
  - **Estável**: a ordem dos elementos com a mesma chave não será mudada;
  - **Não estável**: a ordem dos elementos com a mesma chave é mudada;

Exemplo: Seja a seguinte instância: {32(a), 2, 10, 32(b)}

  - Ordenação estável: {2, 10, 32(a), 32(b)}
  - Ordenação não estável: {2, 10, 32(b), 32(a)}

- Podem ser divididos em:
  - **Básicos**: implementação mais simples, menos eficientes.
  - **Sofisticados**: implementação não trivial, mais eficientes.

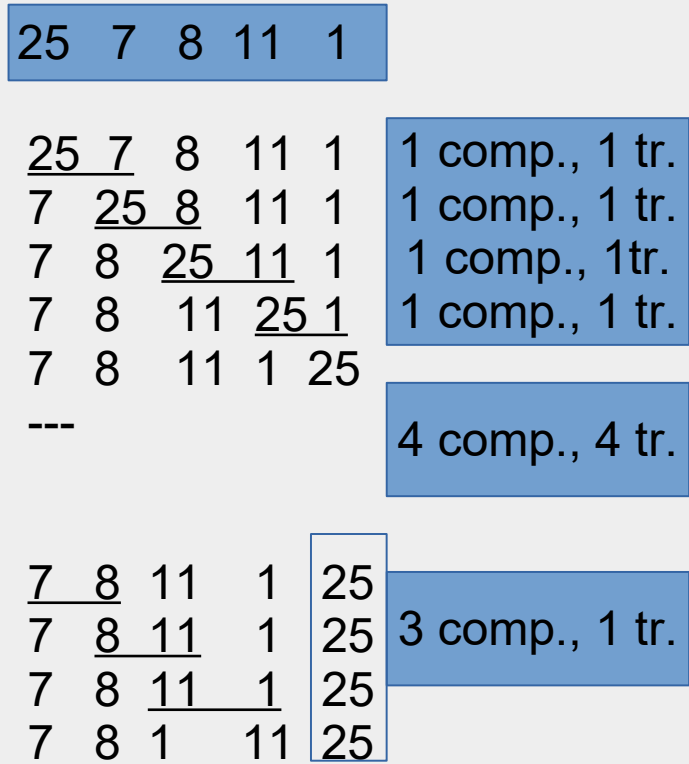
- Algoritmos de Ordenação Básicos: *Bubble Sort*

```
void bubbleSort(int* v, int n)
{
    int i, aux, fim, troca;
    fim = n;
    do{
    troca = -1;
      for(i=0;i<fim - 1;i++)
      {    //troca sempre que o elemento da esquerda for maior
            if(v[i]>v[i+1])//realiza a troca
            {
                  aux = v[i]; v[i] = v[i+1]; v[i+1] = aux;
                  troca = i;//marca se houve a troca
            }
      }
      fim--;
    }while(troca!=-1);//enquanto tiver acontecido troca
}
```

Carrega o maior elemento para a última posição, depois o segundo maior elemento para a penúl-tima posição, e assim por diante.

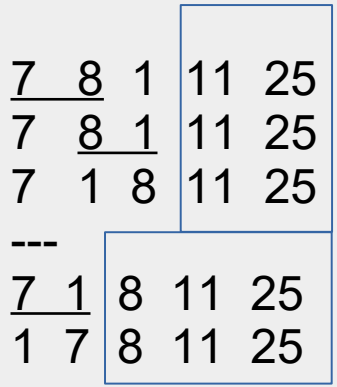
- Algoritmos de Ordenação Básicos: *Bubble Sort*

```
void bubbleSort(int* v, int n)
{
    int i, aux, fim, troca;
    fim = n;
    do{
    troca = -1;
      for(i=0;i<fim - 1;i++)
      {
              if(v[i]>v[i+1])
            {
                  aux = v[i]; v[i] = v[i+1]; v[i+1] = aux;
                  troca = i;
            }
      }
      fim--;
    }while(troca!=-1);
}
```



- Algoritmos de Ordenação Básicos: *Bubble Sort*

```
void bubbleSort(int* v, int n)
{
    int i, aux, fim, troca;
    fim = n;
    do{
    troca = -1;
      for(i=0;i<fim - 1;i++)
      {
              if(v[i]>v[i+1])
            {
                  aux = v[i]; v[i] = v[i+1]; v[i+1] = aux;
                  troca = i;
            }
      }
      fim--;
    }while(troca!=-1);
}
```



2 comp., 1 tr.

1 comp., 1 tr.

=> Estável

10 comp., 7 trocas

- Algoritmos de Ordenação Básicos: *Bubble Sort*
  - Complexidade
    - $O(n)$, melhor caso: vetor ordenado crescente
    - $O(n^2)$, pior caso: vetor ordenado decrescente
    - $O(n^2)$, caso médio: desordenado

- Algoritmos de Ordenação Básicos: *Selection Sort*

```
void selectionSort(int* v, int n)
{
    int i, j, menor, aux;

    for(i=0;i<n - 1;i++)
    { menor = i;
      for(j=i+1;j<n;j++)//procurando se tem menor à direita
      {       //atualiza a posição do novo menor
              if(v[j]<v[menor])  menor = j;

      }
      if(i!=menor)//encontrou um elemento menor à direita
      {//realiza a troca
         aux = v[menor]; v[menor] = v[i]; v[i] = aux;
      }
    }
}
```

Procura o menor à direita de i, e atualiza a nova posição do menor, trazendo o menor para primeira posição, e assim sucessivamente.

- Algoritmos de Ordenação Básicos: *Selection Sort*

```
void selectionSort(int* v, int n)
{
    int i, j, menor, aux;

    for(i=0;i<n - 1;i++)
    { menor = i;
      for(j=i+1;j<n;j++)
      {
            if(v[j]<v[menor])  menor = j;

      }
      if(i!=menor)
      {
        aux = v[menor]; v[menor] = v[i]; v[i] = aux;
      }
    }
}
```

25 7 8 11 1

menor = 0 1 4
1 7 8 11 25
4 comp., 1 tr.

---

menor = 1
1 7 8 11 25
3 comp., 0 tr.

---

menor = 2
1 7 8 11 25
2 comp., 0 tr.

---

menor = 3
1 7 8 11 25
1 comp., 0 tr.

=> Estável

10 comp., 1 troca

- Algoritmos de Ordenação Básicos: *Selection Sort*
  - Complexidade
    - $O(n^2)$, sempre.

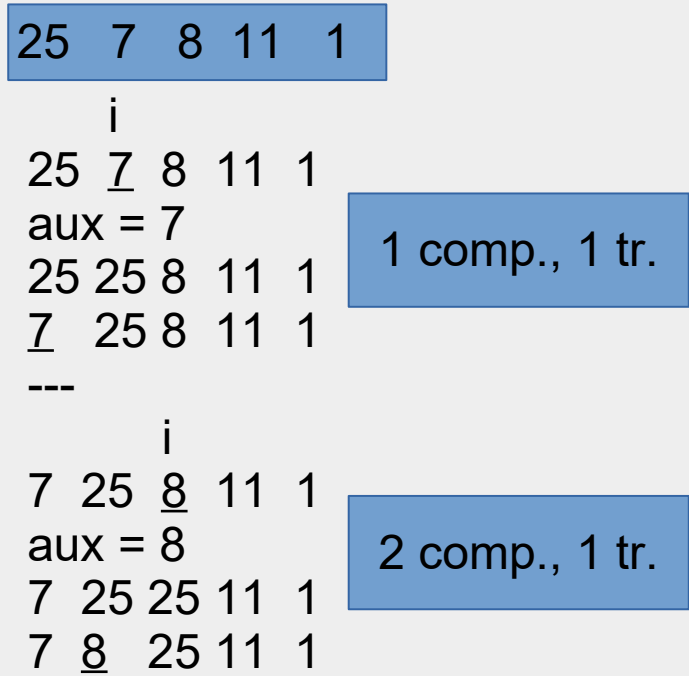
- Algoritmos de Ordenação Básicos: *Insertion Sort*

```
void insertionSort(int* v, int n)
{
    int i, j, aux;
    for(i=1; i<n; i++){//vai da segunda posição em diante
       aux = v[i];
       j = i;
       while(j>0 && v[j-1] > aux)//enqto existe alguém maior
       { //que aux, faz o deslocamento para direita
             v[j] = v[j-1]; j--;//duplicando esse elemento
       }
       v[j] = aux;//local correto em que posso substituir
    }
}
```

Pega o elemento a ser ordenado, e depois procura a posição certa para inseri-lo enquanto o aux for menor que os vizinhos à esquerda.

- Algoritmos de Ordenação Básicos: *Insertion Sort*

```
void insertionSort(int* v, int n)
{
    int i, j, aux;
    for(i=1; i<n; i++){
       aux = v[i];
       j = i;
       while(j>0 && v[j-1] > aux)
       {
              v[j] = v[j-1]; j--;
       }
       v[j] = aux;
    }
}
```



GSI011 – Estrutura de Dados 2 – Aula 5

- Algoritmos de Ordenação Básicos: *Insertion Sort*

```
void insertionSort(int* v, int n)
{
    int i, j, aux;
    for(i=1; i<n; i++){
       aux = v[i];
       j = i;
       while(j>0 && v[j-1] > aux)
       {
              v[j] = v[j-1]; j--;
       }
       v[j] = aux;
    }
}
```

```
                i
7  8  25  11  1
aux = 11
7  8  25  25  1
7  8  11  25  1
---
                   i
7  8  11  25  1
aux = 1
7  8  11  25  25
7  8  11  11  25
7  8   8  11  25
7  7   8  11  25
1  7   8  11  25
```

2 comp., 1 tr.

4 comp., 1 tr.

=> Estável

9 comp., 4 trocas

- Algoritmos de Ordenação Básicos: *Insertion Sort*
  - Complexidade
    - $O(n)$, melhor caso: vetor ordenado crescente
    - $O(n^2)$, pior caso: vetor ordenado decrescente
    - $O(n^2)$, caso médio: desordenado

- Comparação entre os algoritmos

Algoritmos	Trocas	Comparações
BubbleSort	7	10
SelectionSort	1	10
InsertionSort	4	9