Tecnologia em Análise e Desenvolvimento de Sistemas - TADS

Estrutura de Dados

Prof. ***Luciano Vargas Gonçalves***
E-mail: luciano.goncalves@riogrande.ifrs.edu.br

tads
Análise e Desenvolvimento de Sistemas
INSTITUTO FEDERAL
Rio Grande do Sul
Campus Rio Grande

Sumário

Estrutura de Dados

- ORDENAÇÃO

Sumário

- **Estrutura de Dados**
  - **Ordenação**
    - Força Bruta
    - Algoritmos
      - Ordenação por Seleção
      - Ordenação por Inserção
      - Ordenação por Seleção e Troca (Buble Sort)
      - Ordenação por Particionamento (QuickSort)

Métodos de Ordenação - SORT

- **Ordenação (SORT):**
  - Corresponde ao método de rearranjar um conjunto de elementos (objetos) em uma ordem crescente ou decrescente;
  - Tem como objetivo facilitar a recuperação dos itens do conjunto;
    - Exemplo:
      - Recuperação de nomes em um lista telefônica;
      - Produtos em estoque;
      - Entre outros;
  - Atividade relevante e fundamental em processamento de dados.

CHAVES para Ordenação

- ***Chaves para ordenação:***
  - A comparação é feita através de uma determinada ***chave*** escolhida, que faz parte de um registro ou elemento (Estrutura);
  - Os registros são usados para representar os elementos a serem ordenados.

```
TipoItem = record
        chave: TipoChave;
        {outras declarações
                desejadas...}
end;
```

Classificação

- **Quanto à Estabilidade**
  - ***Métodos Instáveis:*** a ordem relativa dos itens com chaves iguais é *alterada durante* o processo de ordenação;
  - ***Métodos Estáveis:*** se a ordem relativa dos itens com chaves iguais *mantém-se inalterada* durante o processo;
    - Ex.: Se a lista dos funcionários é ordenada pelo campo “Salário”, um método estável produz uma lista em que os funcionários com o mesmo salário aparecem em ordem alfabética.
    - Alguns dos métodos de ordenação mais eficientes não são estáveis.

Estabilidade - Exemplo

- Lista Original

10	20	30	40	50	60	70	80	90
R$ 100,00	R$ 100,00	R$ 200,00	R$ 400,00	R$ 500,00	R$ 600,00	R$ 600,00	R$ 500,00	R$ 400,00

Estável:

- A. Estável

10	20	30	40	90	50	80	60	70
R$ 100,00	R$ 100,00	R$ 200,00	R$ 400,00	R$ 400,00	R$ 500,00	R$ 500,00	R$ 600,00	R$ 600,00

Instável:

- A. Instável

Ordem invertida

20	10	30	90	40	50	80	70	60
R$ 100,00	R$ 100,00	R$ 200,00	R$ 400,00	R$ 400,00	R$ 500,00	R$ 500,00	R$ 600,00	R$ 600,00

Ordenação

- **Medidas de complexidade levam em conta:**
  - O número de comparação entre as chaves;
  - O número de trocas entre os itens;
  - São classificados em dois tipos:
    - **Métodos Simples:** mais recomendados para conjuntos pequenos de dados. Usam mais comparações, mas produzem códigos menores e mais simples;
    - **Métodos Eficientes ou Sofisticados:** adequados para conjuntos maiores de dados. Usam menos comparações, porém produzem códigos mais complexos e com muitos detalhes.

Algoritmos de Ordenação

- Ordenação por Seleção (Selection Sort)
- Ordenação por Inserção (Insertion Sort)
- Ordenação por Seleção e Troca (Bubble Sort)

Método Simples

- Ordenação por Inserção através de incrementos decrescentes (ShellSort)
- Ordenação por Particionamento (QuickSort)
- Ordenação de Árvores (HeapSort) não será visto ED1

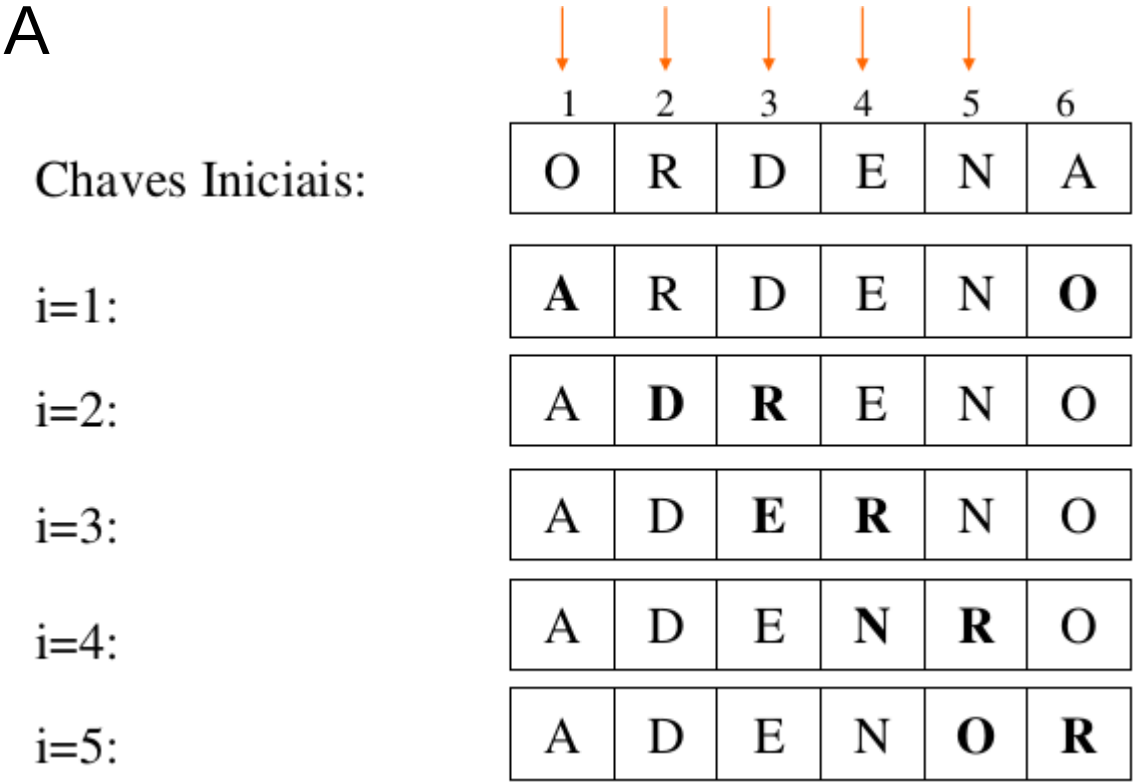
Método Eficientes

Ordenação por Seleção (Selection Sort)

- Selection Sort
  - Um dos algoritmos mais simples (visto em lógica programação)
  - Recomendado para conjuntos pequenos de elementos;
  - Procedimento:
    - Selecione o menor item do conjunto e troque-o com o item que está na posição i (posição em avaliação);
    - Repita essas operações com os demais itens até que reste apenas um elemento.

Ordenação por Seleção (Selection Sort)

- Palavra inicial – ORDENA
- Ordenação
  - Alfabética Crescente
  - ***ORDENA <> ADENOR***



Troca da posição de 1 até 6 pela menor

Ordenação por Seleção (Selection Sort)

Busca o item menor e troca com o elemento da posição (i)

5 3 4 1 2

Selection Sort

https://algorithms.tutorialhorizon.com/selection-sort

Ordenação por Seleção (Selection Sort)

- Algoritmo
  - Pseudocódigo

```
PROCEDURE SelectionSort(vet: Vetor);
VAR i, j, min : integer; aux: TipoItem;
BEGIN
   FOR i:=1 to TamConjunto DO
     BEGIN
        min := i; {Posição a ser ordenada}
        FOR j:=i+1 do TamConjunto DO
                IF (vet[j].chave < vet[min].chave) THEN
                        min := j; {Posição a ser trocada}
                aux := vet[min];
                vet[min] := vet[i];   } Troca
                vet[i] := aux;
        END;
END;
```

Ordenação por Seleção (Selection Sort)

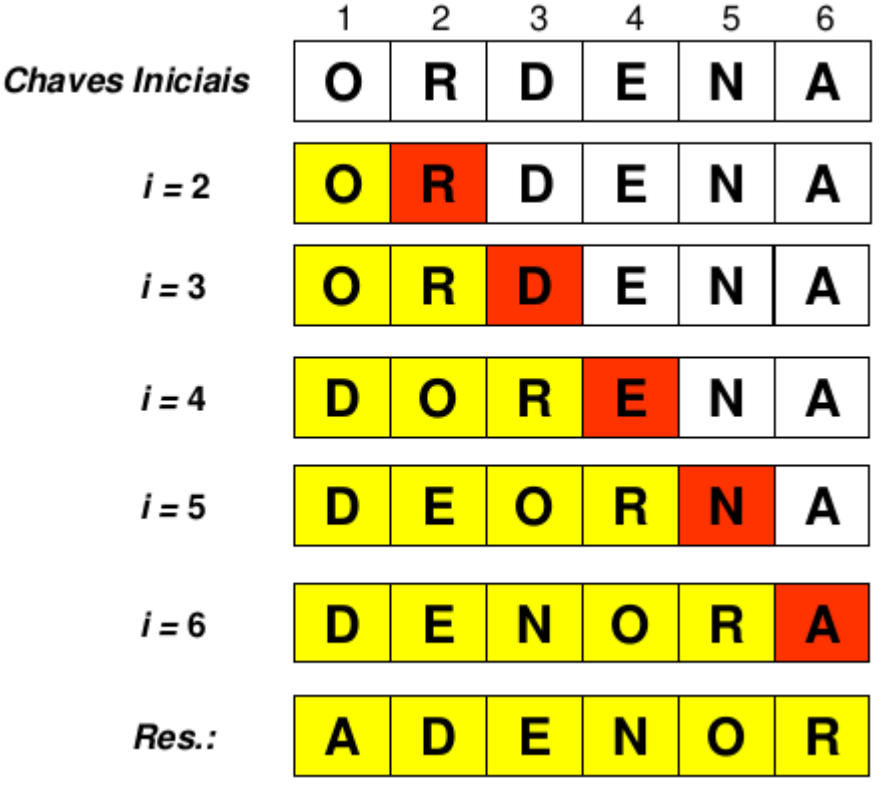
- Desvantagens
  - O fato do conjunto já estar ordenado não ajuda em nada (o número de comparações continuará o mesmo).
  - O algoritmo ***não é estável,*** isto é, os registros com chaves iguais nem sempre irão manter a mesma posição relativa de antes do início da ordenação.

Ordenação por Inserção (Insertion Sort)

- INSERTION SORT
  - Também um algoritmo simples
  - Procedimento
    1) Os elementos são divididos em uma sequência de ***destino*** a_1 , ..., a_{i-1} e em uma sequência ***fonte*** a_i , ..., a_n .
    2) Em cada passo, a partir de i =2, o i-ésimo item da sequência fonte é retirado e transferido para a sequência destino sendo inserido na posição adequada.

Ordenação por Inserção (Insertion Sort)

- Insertion Sort
  - A inserção do item em uma posição adequada na sequência de destino é realizada com a movimentação das chaves **maiores para a direita (deslocamento)** e então é feita a inserção do item na posição vazia



Ordenação por Inserção (Insertion Sort)

```
1: for j = 2 to A.length do
2:     key = A[j]
3:     i = j − 1
4:     while i > 0 and A[i] > key d
5:         A[i + 1] = A[i]
6:         i = i − 1
7:     end while
8:     A[i + 1] = key
9: end for
```

6 5 3 1 8 7 2 4

Retira na posição i+1 e insere Ordenado (j<i

https://commons.wikimedia.org/wiki/

Ordenação por Inserção (Insertion Sort)

Vantagens:

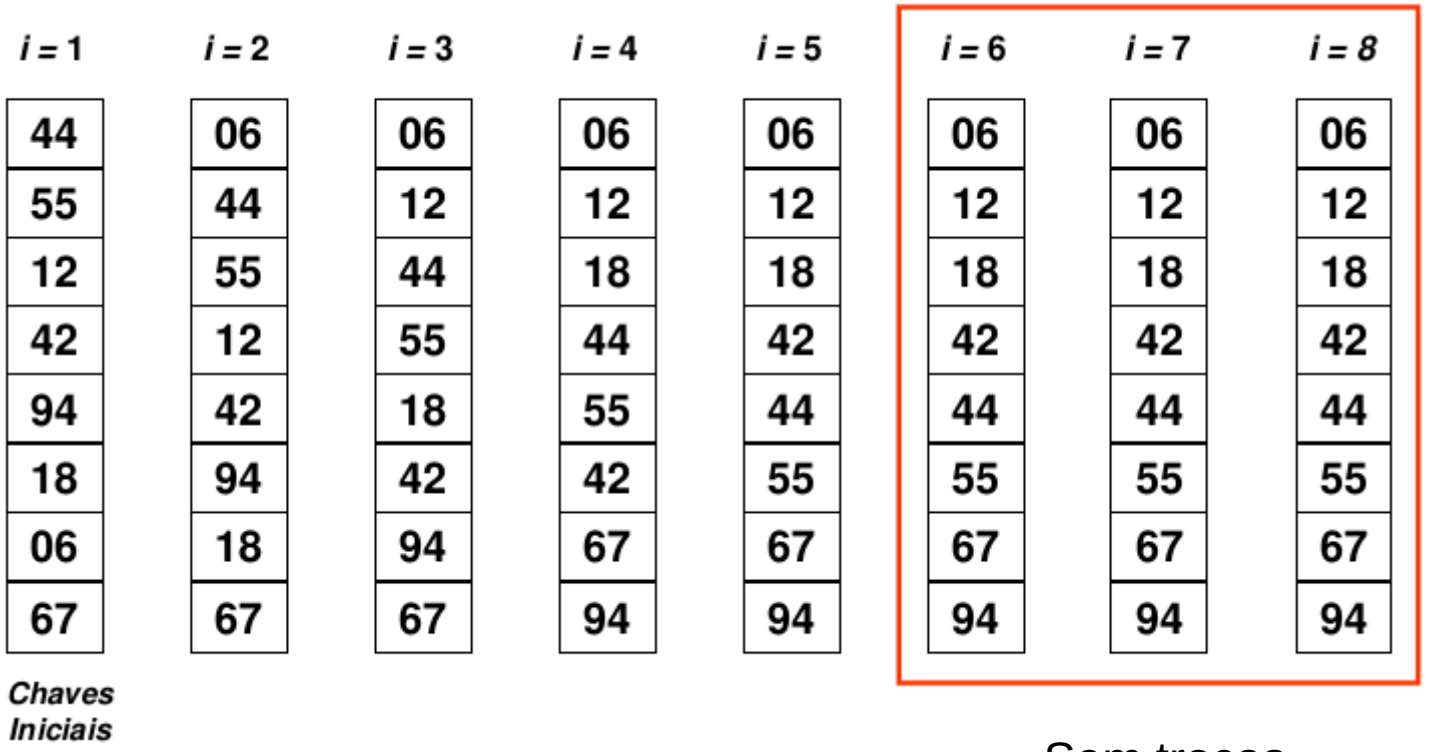
- O número mínimo de comparações e movimentos ocorre quando os itens já estão originalmente ordenados;
- O número máximo ocorre quando os itens estão originalmente em ordem reversa, o que indica um comportamento natural para o algoritmo.

Ordenação por Seleção e Troca (Bubblesort)

- **BubbleSort**
  - Um dos algoritmos mais simples
  - Princípio:
    - 1. As chaves Item[1].Chave e Item[2].Chave são comparadas e trocadas se estiverem fora de ordem;
    - 2. Repete-se o processo de comparação e troca com Item[2] e Item[3], Item[3] e Item[4], ...
  - Por que Bolha?
    - Se o vetor a ser ordenado for colocado na vertical, com Item[n] em cima e Item[1] embaixo, durante cada passo o menor elemento “sobe” até encontrar um elemento maior ainda, como se uma bolha subisse dentro de um tubo de acordo com sua densidade

Ordenação por Seleção e Troca (Bubblesort)

- Lista nos diferentes tempos



Ordenação por Seleção e Troca (Bubblesort)

```
procedure Bolha(var L: Lista, n: integer);
   var i,j: Índice;
        x: TipoItem;
begin
   for i:= 2 to n do
      for j:= n to i do begin
        if L.Item[j-1].Chave > L.Item[j].Chave then
          begin
            x:= L.Item[j-1];
             L.Item[j-1] := L.Item[j];
            L.Item[j] := x;
          end;
       end; {for}
end;
```

Ordenação por Seleção e Troca (Bubblesort)

Cada rodada o maior sobe e o menor desce

6 5 3 1 8 7 2 4

https://commons.wikimedia.org/wiki/File:Bubble-sort.gif

Ordenação por Seleção e Troca (Bubblesort)

- Bubblesort
  - Note que no exemplo, as três últimas iterações não afetam a ordem do vetor; assim o algoritmo pode ser melhorado!
  - Técnica óbvia: manter uma indicação para saber se houve ou não troca na última iteração: se não houve, o vetor já está ordenado.

Ordenação por Seleção e Troca (Bubblesort)

```
procedure Bolha2 (var A: Vetor; var n: Indice);
var i, j : Indice;
    temp: Item; troca: boolean;
begin
i := n; troca := TRUE;
while (i >= 2) and (troca = TRUE) do begin
    troca := FALSE;
      for j:= 1 to (i-1) do begin
          if A[j].chave < A[j+1].chave then begin
            temp := A[j].chave;
            A[j].chave := A[j+1].chave;
            A[j+1].chave := temp;
            troca := TRUE;
          end;
      end; (for)
    i := i – 1;
end;
end;
```

Informa se teve alguma troca

Ordenação por Seleção e Troca (Bubblesort)

- Método extremamente lento: só faz comparações entre posições adjacentes
  - É o método mais ineficiente entre os simples
    - Melhor caso: vetor já ordenado
    - Pior caso: vetor de entrada em ordem reversa
  - Cada passo aproveita muito pouco do que foi “descoberto” em relação à ordem das chaves no passo anterior (exibe informações redundantes)

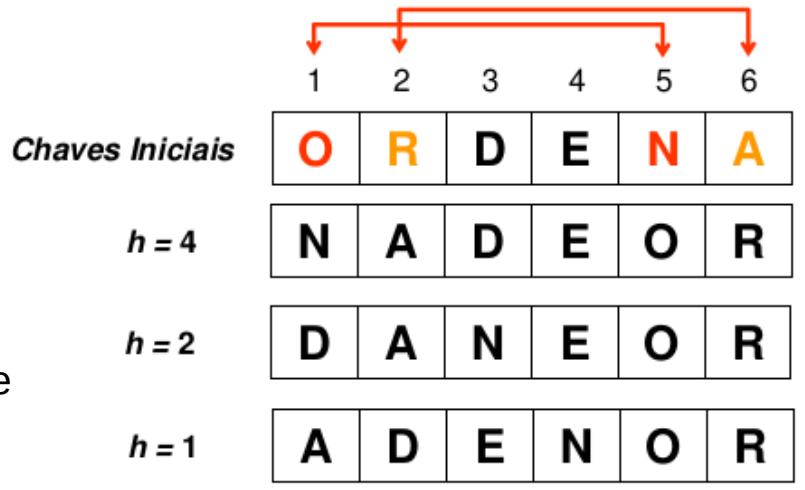
Métodos Eficientes de Ordenação

ShellSort

- Método proposto por Shell em 1959
  - É uma extensão da ordenação por inserção
    - O Método de Inserção troca itens adjacentes quando procura o ponto de inserção na sequência destino
    - Se o menor item estiver na posição mais a direita no vetor, então o número de comparações e movimentações é igual a n – 1 para encontrar o seu ponto de inserção
  - O Shellsort contorna o problema permitindo trocas de registros que estão distantes um do outro. Os itens que estão separados ***h*** posições são rearranjados de forma que todo ***h-ésimo*** item leva a uma sequência ordenada.

ShellSort

1) Na primeira passada (h=4):
   - O item O é comparado com N (posições 1 e 5) e trocados
   - O item R é a seguir comparado e trocado com A (posições 2 e 6)
2) Na segunda passada (h=2):
   - N, D e O, nas posições 1, 3 e 5 são rearrumados para resultar em D, N e O nestas mesmas posições; da mesma forma, A, E e R, nas posições 2, 4 e 6 são comparados e mantidos nos seus lugares.
3) A última passada (h=1):
   - corresponde ao algoritmo de inserção, mas apenas trocas locais serão executadas



ShellSort

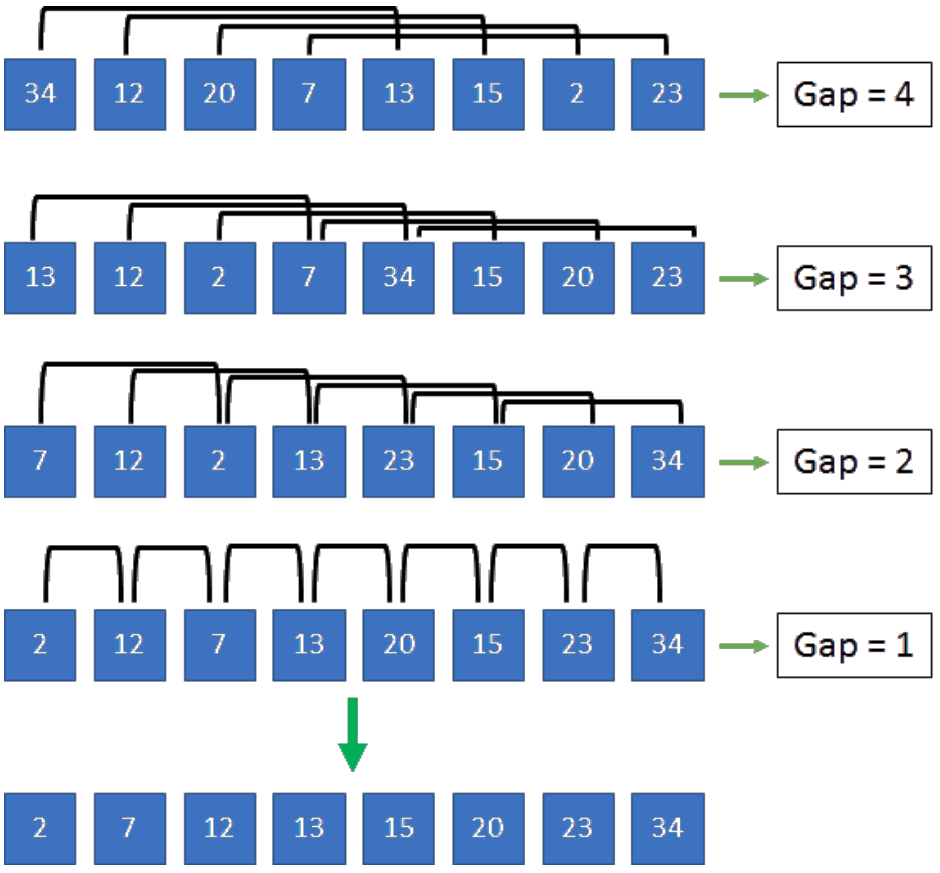
- A razão pela qual este método é mais eficiente ainda não é conhecida porque ninguém ainda foi capaz de analisar o algoritmo!
- A sua análise envolve problemas matemáticos muito difíceis, como definir qual a sequência de incrementos deve fornecer os melhores resultado.
- A sequência apresentada foi obtida de maneira empírica e uma análise matemática indica que o esforço computacional para ordenar n elementos é proporcional a $n^{1/2}$ com o Shellsort.

ShellSort

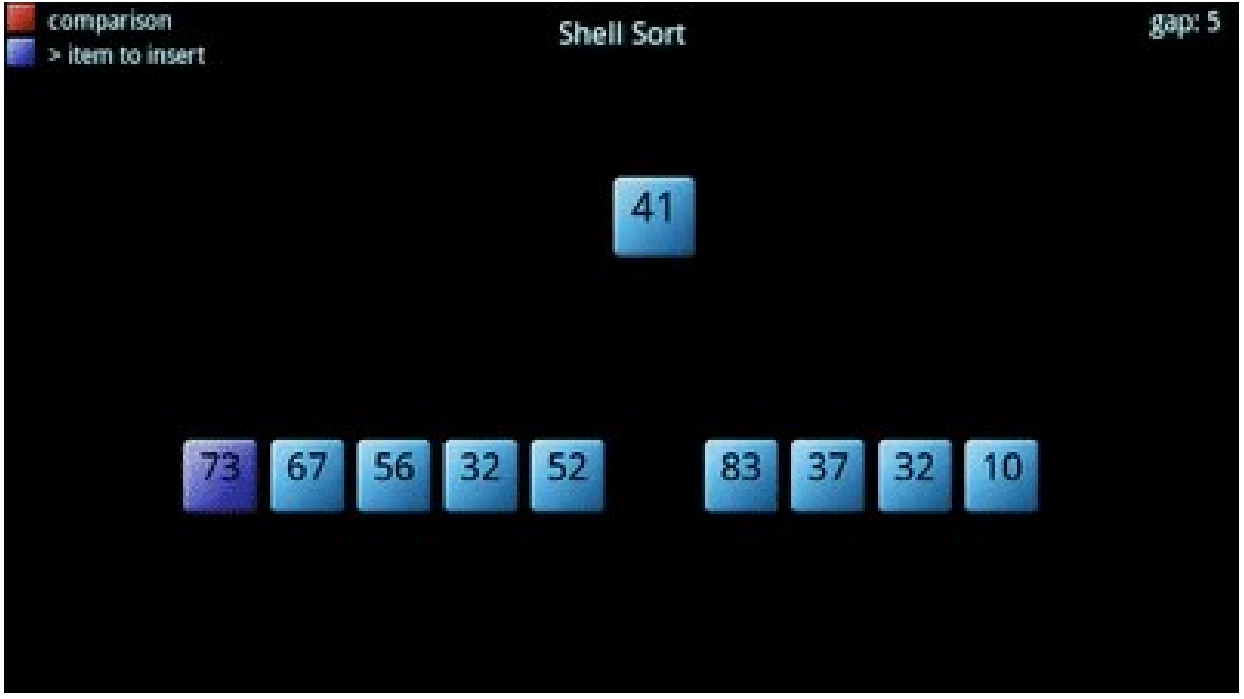
- Algoritmo Shell Sort

```
Função ShellSort(A, n)
        aux, i, j, h = n/2;
        Enquanto h > 0
                i = h;
                Enquanto i < n
                        aux = A[i]
                        j = i;
                Enquanto j >= h && aux < A[j - h]
                                A[j] = A[j - h];
                                j = j - h;
                        A[j] = aux;
                        i = i +1;
        h = h/2;
```

ShellSort


34 12 20 7 13 15 2 23
Gap = 4
13 12 2 7 34 15 20 23
Gap = 3
7 12 2 13 23 15 20 34
Gap = 2
2 12 7 13 20 15 23 34
Gap = 1
2 7 12 13 15 20 23 34

ShellSort



https://makeagif.com/gif/shell-sort-algorithm-mKGEkd

ShellSort

- Shellsort : ótima escolha para arquivos de tamanho moderado;
- Implementação simples e quantidade pequena de código;
- Melhor método para conjuntos de dados pequenos e médios;
- O método também não é estável.

QuickSort

- Ordenação por Particionamento ou Quicksort
  - O Quicksort é o algoritmo mais rápido para ordenação interna conhecido para uma grande quantidade de situações, sendo por isso o mais utilizado entre todos os algoritmos de ordenação.
  - Princípio
    - Dividir o problema de ordenar um conjunto de ***n*** itens em dois problemas menores para ordenar;
    - Ordenar independentemente os problemas menores;
    - Combinar os resultados para produzir a solução do problema maior.

Ordenação por Particionamento - QuickSort

- Quicksort
  - A parte mais delicada desse método se refere à divisão da partição:
    - Deve-se rearranjar o vetor na forma A[Esq..Dir] através da escolha arbitrária de um item ***x*** do vetor chamado ***pivo;***
    - Ao final, o vetor A deverá ter duas partes, uma esquerda com chaves menores ou iguais que ***x*** e a direita com valores de chaves maiores ou iguais que ***x.***

QuickSort

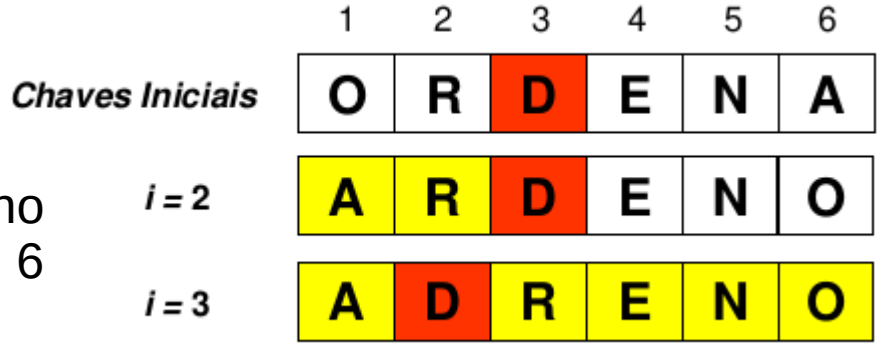
- Procedimento Algoritmo QuickSort
  - Escolher arbitrariamente um item do vetor e colocar este valor em x
  - Percorrer o vetor a partir da esquerda até que um item A[i] >= x é encontrado; da mesma maneira, percorrer o vetor a partir da direita até que um item A[j] <= x é encontrado;
  - Como os itens A[i] e A[j] não estão na ordem correta no vetor final, eles devem ser trocados
  - Continuar o processo até que os índice i e j se cruzem em algum ponto do vetor

QuickSort

- Funcionamento
  - Ao final do processo, o vetor A[Esq..Dir] está particionado de tal forma que:
    - Os itens em A[Esq], A[Esq+1],..., A[j] são menores ou iguais a x
    - Os itens em A[i], A[i+1],..., A[Dir] são maiores ou iguais a x

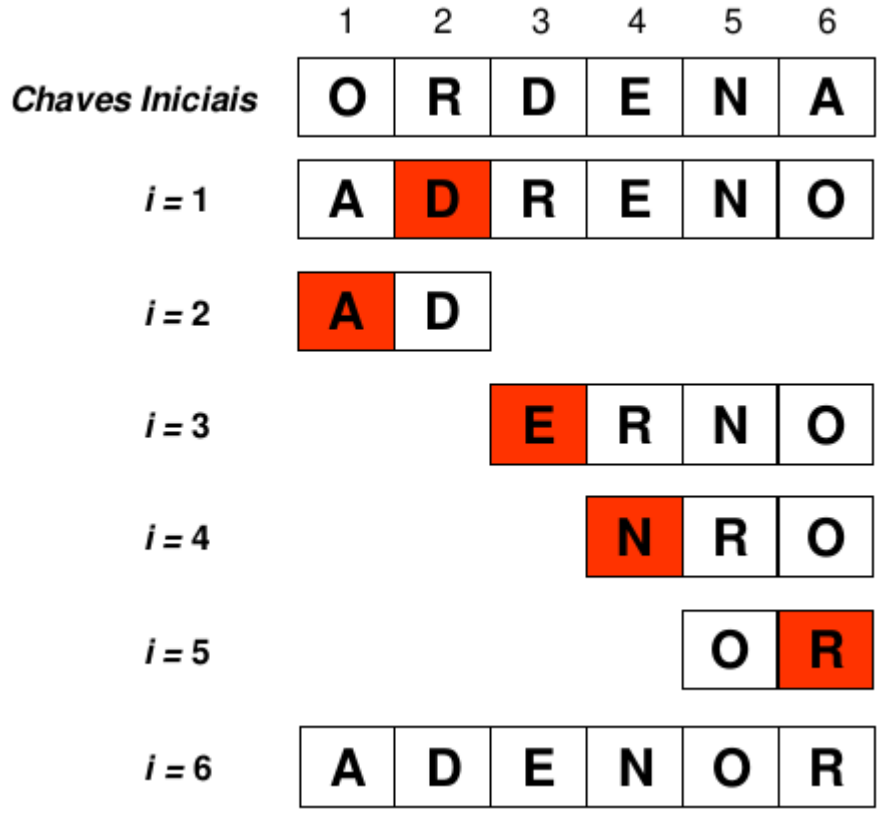
QuickSort

- Exemplo
  - O pivô é escolhido como sendo A[(i+j) div 2]
  - Inicialmente, i=1 e j=6, e então x=A[3] = D
  - A varredura a partir da posição 1 pára no item O e a varredura a partir da posição 6 pára em A, sendo os dois itens trocados
  - A varredura a partir da posição 2 pára em R e a varredura a partir da posição 5 pára no item D, e então os dois itens são trocados
  - Neste instante i e j se cruzam (i=3 e j=2), o que encerra o processo de partição



QuickSort

- Exemplo
  - Fim da partição – ordenar as partes



QuickSort

```
procedure Particao(Esq,Dir: Índice; var i,j:
  Índice);
   var pivo, x: Item;
begin
 i := Esq;
 j:= Dir;
 pivo:= A[(i+j) div 2)];  {obtencao do pivo}
 repeat;
    while pivo.Chave > A[i].Chave do i:=i+1;
    while pivo.Chave < A[j].Chave do j:=j-1;
    if I <= j then begin
       x := A[i]; A[i]:=A[j];A[j]:=x;
       i:=i+1; j:=j-1;
    end;
  until i>j;
end;
```

QuickSort

- Esq e Dir são índices para definir os sub-vetores do vetor original A a ser particionado
- i e j retornam as posições finais das partições, onde:
  - A[Esq], A[Esq+1],..., A[j] são menores ou iguais a x
  - A[i], A[i+1],..., A[Dir] são maiores ou iguais a x
- O vetor é uma variável global ao procedimentoPartição

QuickSort

Parte 2 do algoritmo

```
procedure Quicksort(var A: Vetor);

{*** A definição do procedimento partição
  entra aqui ***}

procedure Ordena(Esq,Dir:Indice);
  var i,j: Indice;
begin
 Particao(Esq,Dir,i,j);
 if Esq < j then Ordena(Esq,j);
 if i < Dir then Ordena(i,Dir);
 i := Esq;
 j:= Dir;
end;
```

QuickSort

- Melhor caso: quando cada partição divide o arquivo em duas partes iguais
- Pontos fracos:
  - A implementação do algoritmo é muito delicada e difícil;
  - O método não é estável;

• Entretanto, desde que se tenha uma implementação robusta o suficiente, o Quicksort deve ser o algoritmo preferido para as aplicações.

QuickSort

6 5 3 1 8 7 2 4

https://commons.wikimedia.org/wiki/File:Quicksort-example.gif