UNIVERSIDADE CATÓLICA DE PERNAMBUCO

Estrutura de Dados II

Métodos de Ordenação
Parte 2

Profª Márcio Bueno
ed2tarde@marciobueno.com / ed2noite@marciobueno.com

Material baseado nos materiais da
Profª Ana Eliza e Prof. Robson Lins

Introdução

- Os três métodos de ordenação analisados (*insertionSort*, *selectionSort* e *bubbleSort*) não são muito eficientes – $O(n^2)$.

- Pode-se esperar algum nível melhor de eficiência para um algoritmo de ordenação?
  - ✓ O limite $O(n^2)$ precisa ser quebrado para melhorar a eficiência e o tempo de execução.
    - Como isso pode ser feito?

Classificação por Inserção

- **Shellsort - Método dos Incrementos Descendentes:**
  - ✓Proposto por Donald L. Shell em 1959
  - ✓Explora o fato de que o método de inserção direta apresenta desempenho aceitável quando o número de chaves é pequeno e/ou estas já estão parcialmente ordenadas.
  - ✓Consiste em dividir o vetor em **H** segmentos, de tal forma que cada um possua aproximadamente **N/H** chaves e classificarmos cada segmento separadamente.

Classificação por Inserção

- **Shellsort - Algoritmo**
  - ✓Para implementar o método, o vetor **V[1..N]** é dividido em segmentos:

    Segmento 1: V[1], V[H+1], V[2H+1], ...
    Segmento 2: V[2], V[H+2], V[2H+2], ...
    Segmento 3: V[3], V[H+3], V[2H+3], ...
    ...
    Segmento H: V[H], V[H+H], V[2H+H], ...

  - ✓Onde **H** é o incremento.
  - ✓Mas qual seria um valor adequado para H?

Classificação por Inserção

- **Shellsort - Algoritmo**
  - ✓Extensivos estudos sugerem escolher incrementos que satisfaçam às seguintes condições:

    $h_1 = 1$

    $h_{i+1} = 3h_i + 1$, ... ,

    h_t, para o qual $h_{t+2} >= N$

    - Para n = 10000, tem-se a seqüência, 1,4,13,40,121,364,1093,3280
  - ✓Por simplificação, utilizaremos incrementos iguais a **potências** inteiras **de 2**.

Classificação por Inserção

- **Shellsort - Algoritmo**

  1°) Num primeiro passo, para um certo H inicial, os segmentos formados são classificados por inserção direta;

  2°) Num segundo passo, o incremento H é diminuído (a metade do valor anterior), dando origem a novos segmentos, os quais também serão classificados por inserção direta;

  3°) O processo se repete até que H = 1. Quando for feita a classificação com H = 1, o vetor estará classificado.

Shellsort - Exemplo

Vetor inicial: (N = 16)

17 25 49 12 18 23 45 38 53 42 27 13 11 28 10 14

✓**Primeiro passo: H = 4**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	⇦ **Índice**
17	25	49	12	18	23	45	38	53	42	27	13	11	28	10	14	
1	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	⇦**Segmento**

✓ Quatro segmentos antes da ordenação (inserção direta):

17 18 53 11	**25 23 42 28**	**49 45 27 10**	**12 38 13 14**

Shellsort - Exemplo

- Quatro segmentos após a ordenação:

11 17 18 53	23 25 28 42	10 27 45 49	12 13 14 38
seg. 1	seg. 2	seg. 3	seg. 4

- Segmento original após H = 4:

11 23 10 12 17 25 27 13 18 28 45 14 53 42 49 38

Shellsort - Exemplo

- ✓ **Segundo passo: H = 2**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	⇦ Índice
11	23	10	12	17	25	27	13	18	28	45	14	53	42	49	38	
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	⇦ Segmento

- ✓ Dois segmentos antes da ordenação:

11 10 17 27 18 45 53 49	23 12 25 13 28 14 42 38
seg. 1	seg. 2

- ✓ Dois segmentos após a ordenação:

10 11 17 18 27 45 49 53	12 13 14 23 25 28 38 42
seg. 1	seg. 2

Shellsort - Exemplo

✓Segundo passo: H = 2

✓Segmento original após H = 2:

10 12 11 13 17 14 18 23 27 25 45 28 49 38 53 42

✓Terceiro passo: H = 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	⇦ Índice
10	12	11	13	17	14	18	23	27	25	45	28	49	38	53	42	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	⇦ Segmento

✓Vetor final:

10 11 12 13 14 17 18 23 25 27 28 38 42 45 49 53

Classificação por Inserção

- **Shellsort – Comentários e Análise**
  - ✓Como o primeiro incremento usado é grande, os segmentos individuais são pequenos e as ordenações com inserção direta são razoavelmente velozes.
  - ✓Cada ordenação parcial dos segmentos favorece o desempenho dos passos seguintes, uma vez que a inserção direta é acelerada quando o vetor já se encontra parcialmente ordenado.

Classificação por Inserção

- **Shellsort – Comentários e Análise**
  - ✓Embora a cada passo se use incrementos menores, e portanto, segmentos maiores, estes se encontram praticamente ordenados.
  - ✓A análise de eficiência é da ordem de $O(n^{1,2})$ que ainda é melhor do que $O(n^2)$

Classificação por Inserção

- **Shellsort – Considerações Gerais**
  - ✓ Características que variam entre diferentes implementações:
    1. Seqüência de incrementos;
    2. Escolha do algoritmo de ordenação simples aplicado a cada segmento, exceto no último passo;
    3. Escolha do algoritmo de ordenação simples aplicado ao último segmento, H = 1.

Classificação por Inserção

- **Shellsort – Considerações Gerais**
  - ✓ Características que variam entre diferentes implementações:
    - Por exemplo, Dobosiewicz usa ordenação por inserção para H != 1 e o método da bolha para H = 1.
    - Incerpi e Sedgewick usam método da bolha para H != 1 e inserção direta para H = 1.
    - Apesar de todas essas variações, todas essas versões têm desempenho melhor do que qualquer método de ordenação simples.

Shellsort - Implementação

```
void shellSort(int data[], int n, int inc[], int num){
    int j, hCnt, h, k, tmp;
    for (num--; num >= 0; num--){
        h = inc[num];
          for (hCnt = h; hCnt < 2*h; hCnt++){
              for (j = hCnt; j < n; j += h){
                    tmp = data[j];
                    k = j;
                    while (k-h >= 0 && tmp < data[k-h]){
                        data[k] = data[k-h];
                        k -= h;
                    }
                    data[k] = tmp;

              }
          }
    }
}
```

Classificação por Inserção

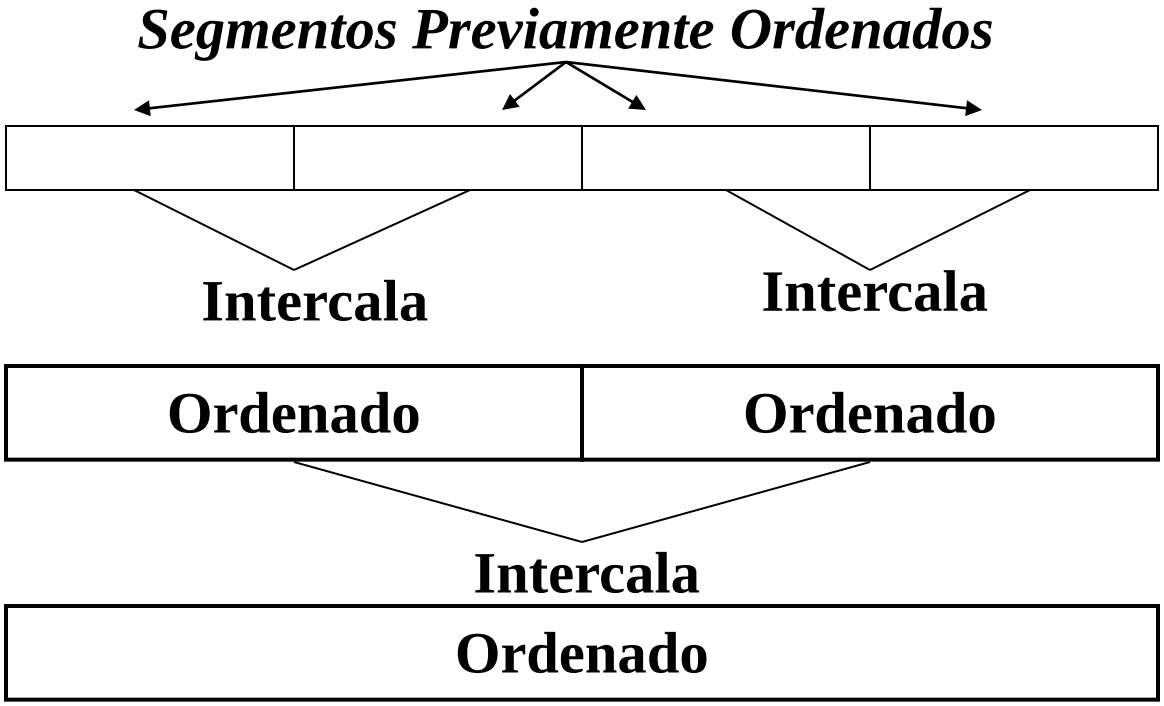
- **Shellsort – Exercícios:**
  - ✓ Mostre o passo a passo de executar shellsort para o seguinte conjunto de dados {9,8,7,6,5,4,3,2,1} usando os incrementos {1,3,7}.
  - ✓ Faça o mesmo para o conjunto {81,94,11,96,12,35,17,95,28,58,41,75,15} usando os incrementos {1,3,5}.

Classificação por Intercalação

- **Definição**
  - ✓ Este método consiste em dividir o vetor a ser classificado em dois ou mais segmentos, ordená-los separadamente e, depois, intercalá-los dois a dois, formando, cada par intercalado, novos segmentos ordenados, os quais serão intercalados entre si, repetindo-se o processo até que resulte apenas um único segmento ordenado.
  - ✓ Para obter a ordenação dos segmentos iniciais, pode ser usado qualquer método de classificação.

Classificação por Intercalação

- **Esquema do Processo de Intercalação**



Classificação por Intercalação

- **Método Mergesort**
  - ✓ Dividir recursivamente o vetor a ser ordenado em dois vetores até obter n vetores de um único elemento.
  - ✓ Aplicar o algoritmo de *merge* tendo como entrada 2 vetores de um elemento e formando um vetor ordenado de dois elementos.
  - ✓ Repetir este processo formando vetores ordenados cada vez maiores até que todo o vetor esteja ordenado.

Método Mergesort - Exemplo

Vetor inicial: (23 17 8 15 9 12 19 7)

Passo 1 : (23 17 8 15 | 9 12 19 7)

Passo 2 : (23 17 | 8 15 | 9 12 7 19)

Passo 3 : (23 | 17 | 8 15 | 9 12 7 19)

Passo 4 : (17 23 | 8 15 | 9 12 7 19)

Passo 5 : (17 23 | 8 | 15 | 9 12 7 19)

Passo 6 : (17 23 | 8 15 | 9 12 7 19)

Passo 7 : (8 15 17 23 | 9 12 7 19)

Passo 8 : (8 15 17 23 | 9 12 | 7 19)

Passo 9 : (8 15 17 23 | 9 | 12 | 7 19)

Passo 10 : (8 15 17 23 | 9 12 | 7 19)

Passo 11 : (8 15 17 23 | 9 12 | 7 | 19)

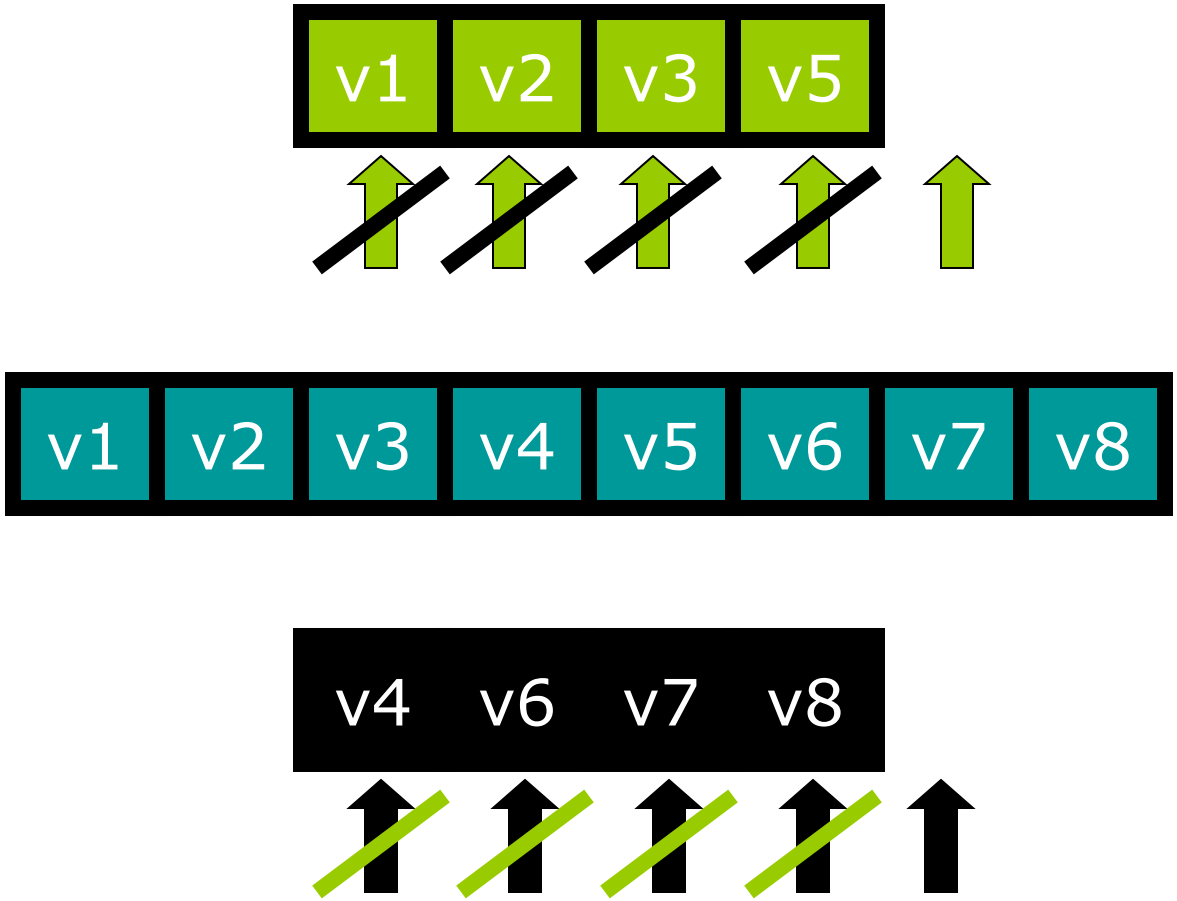
Passo 12 : (8 15 17 23 | 9 12 | 7 19)

Passo 13 : (8 15 17 23 | 7 9 12 19)

Passo 14 : (7 8 9 12 15 17 19 23)

Vetor final: (7 8 9 12 15 17 19 23)

Como intercalar (fazer o *merge*)?



Classificação por Intercalação

- **Mergesort - Implementação**

```
void mergeSort(int v[], int inicio, int fim) {
    int meio;

  if (inicio == fim) { // tamanho = 1
   return;
  } else {
   // encontra o meio
   meio = (inicio + fim) / 2;
   // ordena 1a metade
   mergeSort(v, inicio, meio);
   // ordena 2a metade
   mergeSort(v, meio + 1, fim);
   // intercala
   merge(v, inicio, meio, fim);
  }
}
```

Classificação por Intercalação

- MergeSort - Implementação

```
void merge(int vetor[], int inicio, int meio, int fim) {
    int prim = inicio;
    int seg = meio + 1;
    int aux[fim+1], i = inicio;
    // enquanto tiver elementos nos dois conjuntos
    while ((prim <= meio) && (seg <= fim)) {
        // insere elemento do primeiro
        if (vetor[prim] <= vetor[seg]) {
            aux[i] = vetor[prim];
            prim++;
        } else { // insere elemento do segundo
            aux[i] = vetor[seg];
            seg++;
        }
        i++;
     }
```

Continua

Classificação por Intercalação

- MergeSort - Implementação

```
void merge(int vetor[], int inicio, int meio, int fim) {
    // sobrou elementos do segundo
    if (prim > meio) {
        while (seg <= fim) {
            aux[i] = vetor[seg];
            seg++;
            i++;
        }
    // sobrou elementos do primeiro
    } else {
        while (prim <= meio) {
            aux[i] = vetor[prim];
            prim++;
            i++;
        }
    }
```

Continua

Classificação por Intercalação

- *MergeSort* - Implementação

```
void merge(int vetor[], int inicio, int meio, int fim) {

    // copia o vetor auxiliar no vetor principal
    for (i = inicio; i <= fim; i++) {
        vetor[i] = aux[i];
    }
}
```

Classificação por Intercalação

- MergeSort – Exercícios:
  - ✓Utilizando o mergeSort ordene o seguinte conjunto de elementos {85,99,98,97,96,95,94,93,92,91,90,89, 87,86, 74}.