Classificação por Intercalação

Este é um bom exemplo de abordagem *top down*, ou de aplicação do princípio da *divisão e conquista*, associado à recursividade.

Ao se observar o andamento do processo sobre a lista, nota-se que a intercalação só começa quando as sublistas tornam-se unitárias. Até lá, *a posição relativa dos elementos não muda*. Ou seja: atinge-se uma situação de n sublistas unitárias, cuja a concatenação é a lista original.

Classificação por Intercalação

E a intercalação começa juntando pares de elementos, a partir das listas unitárias. Ora, bem, no caso de um vetor, os elementos individuais já estão acessíveis. Logo, pode-se começar a ordenação com meio caminho andado (em relação às listas), intercalando trechos de *um* elemento, depois de *dois, quatro* e assim por diante. Cada trecho sob intercalação é dito "cadeia" ou "cordão" de intercalação. Veja o esquema (K é o comprimento da cadeia).

Classificação por Intercalação

Exemplo: Ordenação de vetor por intercalação

1º passo K = 1 75 | 25 | 95 | 87 | 64 | 59 | 86 | 40 | 16 | 49 | 12

[0..0] e [1..1], [2..2] e [3..3], [4..4] e [5..5], [6..6] e [7..7], [8..8] e [9..9]

2º passo K = 2 25 75 | 87 95 | 59 64 | 40 86 | 16 49 | 12

[0..1] e [2..3], [4..5] e [6..7], [8..9] e [10..10]

3º passo K = 4 25 75 87 95 | 40 59 64 86 | 12 16 49

[0..3] e [4..7]

4º passo K = 8 25 40 59 64 75 86 87 95 | 12 16 49

[0..7] e [8..10]

12 16 25 40 49 59 64 75 86 87 95

Classificação por Intercalação

Logo, o problema divide-se, então, em duas partes:

i. delimitar as partições do vetor que deves ser intercaladas;

ii. intercalar as partições.

Classificação por Intercalação

Para estabelecer as cadeias a intercalar, começa-se com tamanho k = 1. Na primeira passagem, formam-se cadeias de tamanho 2, depois de tamanho 4, 8, etc. Assim na primeira passagem, são intercaladas as partições V[0..0] e V[1..1], V[2..2] e V[3..3], V[4..4] e V[5..5] etc. Na segunda V[0..1] e V[2..3], V[4..5] e V[6..7], etc. Na terceira, V[0..3] e V[4..7], V[8..11] e V[12..15], etc.

Regra geral, na passagem i, em que o tamanho da cadeia é $k = 2^{i-1}$, são intercalados os trechos V[j..j+k-1] e V[j+k..j+2*k-1].

Classificação por Intercalação

O problema da intercalação tem solução simples baseada no processo conhecido como *balance line*: percorre-se as duas cadeias a intercalar, usando um cursor para cada uma, copiando para um vetor-resposta sempre o menor elemento dentre os iniciais, avançando-se o cursor apenas da cadeia fornecedora. Ao se esgotar uma das cadeias, a outra é percorrida até o fim, preenchendo-se o vetor-resposta.

Classificação por Intercalação

Uma questão adicional que se coloca é:

Como salvar o resultado a cada passagem?

Só o poderíamos fazer sobre o próprio vetor se fosse adotada uma solução recursiva, como no caso da lista. Mas isto de fato replicaria muitas vezes a área original, pelo salvamento cumulativo de versões parcialmente ordenadas. Melhor intercalar um vetor auxiliar, contendo uma cópia do vetor a ordenar, colocando o resultado no vetor original (vetor-resposta).

Classificação por Intercalação

Com base no que foi discutido, codifique uma função que receba um vetor (de inteiros) e o número de elementos no mesmo e através do método *merge sort* ordene de forma crescente os elementos do vetor.

```
void merge_sort (int *v, int n)
{
  int v_aux[n], tam_cadeia=1, esq, i;
  while (tam_cadeia<n)
  {
    for (i=0; i<n; v_aux[i]=v[i++]);
    esq=0;
    while (esq<n-tam_cadeia)
    {
       intercala (v, v_aux, esq, esq+tam_cadeia-1,
       esq+tam_cadeia, ((esq+2*tam_cadeia-1)<n)?
       (esq+2*tam_cadeia-1):(n-1));
       esq=esq+2*tam_cadeia-1<n?esq+2*tam_cadeia:n;
    }
    tam_cadeia*=2;
  }
}
```

```
void intercala (int *v, int *v_aux, int limesqesq, int limesqdir,
int limdiresq, int limdirdir) {
  int deve_continuar=1, esq_menor, IND=limesqesq;
  while (deve_continuar)   {
    esq_menor=v_aux[limesqesq]<v_aux[limdiresq];
    v[IND++]=esq_menor?v_aux[limesqesq++]:
    v_aux[limdiresq++];
    deve_continuar=limesqesq<=limesqdir&&
    limdiresq<=limdirdir;
  }
  while (limesqesq<=limesqdir)
    v[IND++]=v_aux[limesqesq++];
  while (limdiresq<=limdirdir)
    v[IND++]=v_aux[limdiresq++];
}
```

Classificação por Intercalação

Quanto à complexidade do algoritmo apresentado nos slides anteriores, em uma análise superficial, pode ser determinada se considerarmos o seguinte: tam_cadeia, atualizada por duplicações sucessivas, assume valores do conjunto [1,2,4,8,16...$\lfloor n/2 \rfloor$], sendo a repetição principal controlada pela condição tam_cadeia <= n/2, o que a qualifica como O (log n). Em cada passagem, cada elemento do vetor é copiado uma vez e intercalando uma vez (na função intercala).

Classificação por Intercalação

O *enquanto* intermediário, i.e, o segundo *enquanto* do algoritmo principal, apenas distribui o processamento sobre os sucessivos subvetores. Isto acarreta na máximo 2n movimentos de dados em cada fase. Logo, o procedimento todo é da ordem de 2n log n, ou seja, O (n log n). Como uma análise mais profunda fugiria do escopo desta disciplina, ficaremos apenas neste nível de analise.

Classificação

É importante perceber que, quando o tamanho de uma lista *n* é pequeno, uma classificação **O**(n^2) é em geral mais eficiente do que uma classificação **O**(n log n). Isto acontece porque usualmente as classificações **O**(n^2) são muito simples de programar e exigem bem poucas ações além de comparações e trocas em cada passagem. Por causa dessa baixa sobrecarga, a constante de proporcionalidade é bem pequena. Em geral, uma classificação **O**(n log n) é muito complexa e emprega um grande número de operações adicionais em cada passagem para diminuir o número das passagens subseqüentes. Sendo assim, sua constante de proporcionalidade é maior. Quando *n* é grande, n^2 supera n log n, de modo que as constantes de proporcionalidade não desempenham um papel importante na determinação da classificação mais veloz. Entretanto, quando *n* é pequeno, n^2 não é muito maior que n log n de modo que uma grande diferença nessas constantes freqüentemente faz com que a classificação **O**(n^2) seja mais rápida.