```
void subdivide (LISTA_ENC *pL, LISTA_ENC *pL1,
    LISTA_ENC *pL2) {
   int cont, k=1;
   for (cont=tam(*pL)/2;cont;cont--) {
      ins(pL1,recup(*pL,1),k++);
      ret(pL,1);
   }
   for (k=1, cont=tam(*pL);cont;cont--)   {
      ins(pL2,recup(*pL,1),k++);
      ret(pL,1);
   }
}
```

```
void intercala (LISTA_ENC *pL1, LISTA_ENC *pL2,
LISTA_ENC *pL)                          {
   int k=1;
   while (*pL1&&*pL2)                   {
      if ((*pL1)->inf>(*pL2)->inf)      {
         ins(pL,(*pL2)->inf,k++);
         ret(pL2, 1);                   }
      else                              {
         ins(pL,(*pL1)->inf,k++);
         ret(pL1, 1);                   } }
   while (*pL1)                         {
      ins(pL,(*pL1)->inf,k++);
      ret(pL1, 1);                      }
   while (*pL2)                         {
      ins(pL,(*pL2)->inf,k++);
      ret(pL2, 1);                      } }
```

Classificação por Intercalação

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

A implementação que vimos é estável?

Sim.

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.

UNIVASF

Classificação por Intercalação

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 a seguir, onde **K** é o comprimento da cadeia.

UNIVASF

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

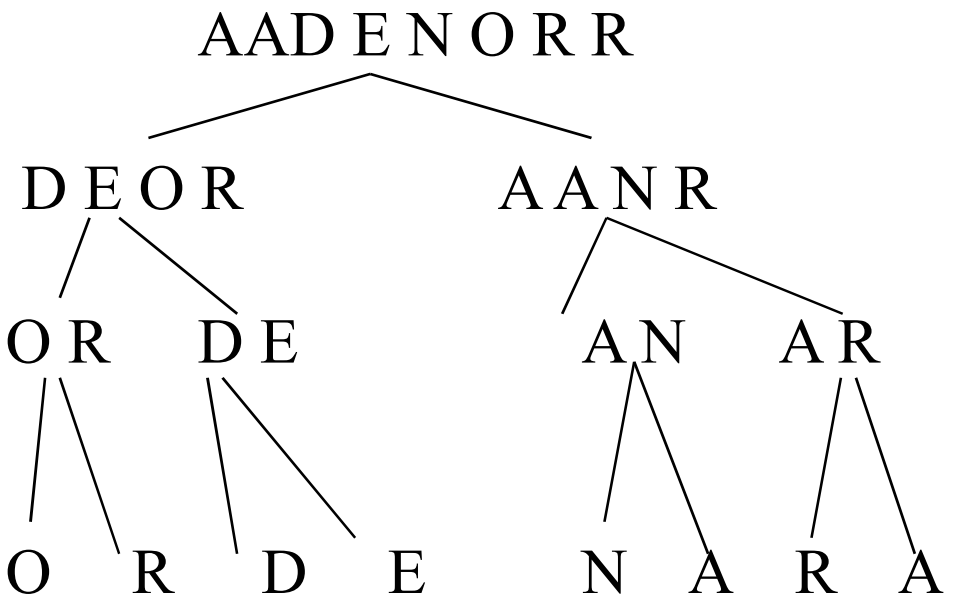
[0..7] e [8..10]

12 16 25 40 49 59 64 75 86 87 95

Classificação por Intercalação

Outro exemplo:

Vetor = {O R D E N A R A}



Classificação por Intercalação

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

i. delimitar as partições do vetor que devem 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.

A 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.

UNIVASF

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. Torna-se 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.

UNIVASF

```
void merge_sort (int *v, int n)
{
  int *v_aux, tam_cadeia=1, j, i;
  if (!(v_aux=(int *)malloc(n*sizeof(int)))) exit(1);
  while (tam_cadeia<n)
  {
    for (i=0; i<n; v_aux[i]=v[i++]);
    j=0;
    while (j<n-tam_cadeia)
    {
       intercala (v, v_aux, j, j+tam_cadeia-1,
       j+tam_cadeia, ((j+2*tam_cadeia-1)<n)?
       (j+2*tam_cadeia-1):(n-1));
       j=j+2*tam_cadeia-1<n?j+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...⌊n/2.0⌋], 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 no 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.

UNIVASF

Métodos de Ordenação que utilizam o Princípio de Distribuição

Exemplo: considere o problema de ordenar um baralho com 52 cartas não ordenadas. Suponha que ordenar o baralho implica em colocar as cartas de acordo com a ordem

A < 2 < 3 < ... < 10 < J < Q < K < ♣ < ♦ < ♥ < ♠

Para ordenar por distribuição, basta seguir os passos abaixo:

1. Distribuir as cartas em 13 montes, colocando em cada monte todos os ases, todos os dois, todos os três, ..., todos os reis.
2. Colete os montes na ordem acima (*as* no fundo, depois os *dois*, etc.), até o rei ficar no topo.

Métodos de Ordenação que utilizam o Princípio de Distribuição

3. Distribua novamente as cartas abertas em 4 montes, colocando em cada monte todas as cartas de paus, todas as cartas de ouros, todas as cartas de copas e todas as cartas de espadas.
4. Colete os montes na ordem indicada (paus, ouros, copas e espadas).

Métodos baseados no princípio de distribuição são também conhecidos como ordenação digital, radixsort ou bucketsort. Neste caso não existe comparação entre chaves.

UNIVASF

Métodos de Ordenação que utilizam o Princípio de Distribuição

Outro exemplo: As antigas classificadoras de cartões perfurados também utilizam o princípio da distribuição para ordenar uma massa de cartões.

Dificuldades de implementar este método:

- Problema de lidar com cada monte.
- Se para cada monte nós reservarmos uma área, então a demanda por memória extra pode se tornar proibitiva.

UNIVASF