```
void shell_sort (int *v, int n) {
   int inc, j, k, y;
   inc=n;
   do{
      inc=inc>1?inc/2:1;
      for (j=inc; j<n; j++)
      {
         y=v[j];
         for (k=j-inc; k>=0 && y<v[k]; k-=inc)
            v[k+inc]=v[k];
         v[k+inc]=y;
      }
   }while(inc>1);  }
```

Método dos Incrementos Decrescentes

A análise da eficiência da classificação de Shell é complicada, em termos matemáticos, e está além do escopo desta disciplina.

No entanto, uma observação se faz importante, o desempenho do processo pode ser prejudicado, se os incrementos forem múltiplos entre si, pois cadeias logicamente adjacentes segundo incrementos múltiplos podem gerar passos redundantes.

Classificação por Intercalação

Assim como a classificação quicksort, veremos agora, mais um método avançado baseado no preceito dividir para conquistar.

Isto é alcançado *a)* pela subdivisão do problema da ordenação em subproblemas menores e/ou *b)* pelo uso de áreas auxiliares, evitando o problema da liberação de casas ocupadas por elementos fora do lugar para receberem chaves relocalizadas (que ocorre nos métodos *in situ* em geral). Consegue-se, nesse grupo, desempenho de ordem **O**(n log n).

Classificação por Intercalação

Analisaremos então, o método de **intercalação** ou ***merge sort***. Uma primeira aproximação ao mesmo é propiciada por um estudo da sua aplicação sobre uma *lista*, em vez de um vetor. Para classificar uma lista, pode-se adotar o seguinte procedimento:

1. **se** a lista tem um só elemento, já está ordenada;
2. **senão,** dividir a lista em duas sublistas, ordená-las e intercalá-las.

Classificação por Intercalação

Os subproblemas que se apresentam são:

i. dividir uma lista;

ii. ordenar uma sublista;

iii. intercalar duas listas.

O subproblema de dividir uma lista pode ser solucionado colocando-se em uma sublista **n/2** elementos da lista original e o restante na outra sublista, onde **n** é o número de elementos na lista original. A ordenação das sublistas é resolvida pela aplicação recursiva da solução do problema principal. A intercalação de duas listas ordenadas consiste apenas em se inserir numa lista resultante cada um dos elementos das listas originais na ordem correta.

Classificação por Intercalação

No processo de intercalação, as listas podem ser vistas como filas: os elementos do início são comparados e o menor deles é retirado da fila correspondente e levado para a lista resposta. Quando uma fila esvaziar, simplesmente copia-se o restante da outra.

Por exemplo, ao se intercalar as listas L1 = (A D E K L P) e L2 = (B C F G) a seqüência de movimentos seria a seguinte:

Classificação por Intercalação

Intercalação de listas ordenadas:

L1	L2	lista resultante
(A D E K L P)	(B C F G)	(A)
(D E K L P)	(B C F G)	(A B)
(D E K L P)	(C F G)	(A B C)
(D E K L P)	(F G)	(A B C D)
(E K L P)	(F G)	(A B C D E)
(K L P)	(F G)	(A B C D E F)
(K L P)	(G)	(A B C D E F G)
(K L P)	( )	(A B C D E F G K L P)

Observe como se aplicariam as duas fases (subdivisão e intercalação) ao se ordenar o vetor exemplo (armazenado numa lista). Em cada fase, os trechos resultantes de subdivisão estão sublinhados e os resultantes de intercalação estão em negrito.

Exemplo: Ordenação de lista por intercalação

Lista original	(75 25 95 87 64 59 86 40 16 49)
Subdivisão	(75 25 95 87 64) (59 86 40 16 49)
Subdivisão	(75 25 95) (87 64) (59 86 40) (16 49)
Subdivisão	(75 25)(95) (87) (64) (59 86) (40) (16) (49)
Subdivisão/**intercalação**	(75) (25) (**64 87**) (59) (86) (**16 49**)
Intercalação	(**25 75**) (**59 86**)
Intercalação	(**25 75 95**) (**40 59 86**)
Intercalação	(**25 64 75 87 95**) (**16 40 49 59 86**)
Intercalação	(**16 25 40 49 59 64 75 86 87 95**)

Tendo como base o que foi discutido, uma possível função, na linguagem C, para implementar a classificação de uma lista através do merge sort é:

```
void merge_sort (LISTA_ENC *pl) {
  if (tam(*pl)>1) {
    LISTA_ENC L1, L2;
    cria_lista(&L1);
    cria_lista(&L2);
    subdivide (pl, &L1, &L2);
    merge_sort (&L1);
    merge_sort (&L2);
    intercala (&L1, &L2, pl); } }
```

```
typedef struct nodo
{
  int inf;
  struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
```

Classificação por Intercalação - Exercício

Como exercício, implemente as funções subdivide e intercala, constantes no algoritmo anterior.

```
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);
   }
   k=1;
   for (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.

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

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.

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 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.

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.