EDUCAÇÃO e TECNOLOGIA

Revista do Instituto Politécnico da Guarda

"EDUCAÇÃO E TECNOLOGIA"
Revista do Instituto Politecnico da Guarda

DIRECTOR: João Bento Raimundo

REDACÇÃO: Rua Comandante Salvador do Nascimento
Telef. 21634 6300 GUARDA

PROPRIEDADE: Instituto Politécnico da Guarda

EXECUÇÃO GRÁFICA: Secção de Reprografia do IPG

Depósito Legal N.° 17.891/87

Nº 4 / Março / 89

"É muito melhor saber um pouco de tudo do que saber tudo de uma só coisa; esta universalidade é a mais bela"

B. Pascal

Continuamos o nosso esforço de, através da Educação e Tecnologia, dar notícia do que mais se vai experimentando, descobrindo, sabendo, enfim, no Instituto Politécnico da Guarda.

Conscientes da inexistência de um saber acabado, do fluir e refluir das mais variadas teses, antíteses e sínteses, o espaço aberto que sempre pretendemos fosse, esta revista granjeou já uma implantação sólida.

Constitui, diríamos, uma amostra do que é o próprio IPG, em termos do seu alargamento e da sua aceitação.

Diremos que o todo que é o Instituto, (que não cremos seja a simples soma das partes, mas a interpretação de todas elas), continua em crescimento e em afirmação.

Os novos cursos lançados no presente ano lectivo - Engenharia de Construção Civil e Engenharia de Manutenção Industrial - vieram alargar o âmbito do intercâmbio científico, tecnológico e pedagógico-didáctico.

Contribuir para o desenvolvimento sócio-cultural e económico desta região tão carenciada é, também, e muito especialmente formar os seus filhos, abrindo todo um leque de opções que lhes venha a permitir uma inserção na vida activa em conformidade com potencialidades pessoais e do meio ainda não exploradas.

Efectivamente no IPG não se faz tudo, nem - muito menos - de tudo se sabe tudo.

Continuaremos a tentar fazer o melhor, que de muito se saiba muito e, desse tudo, se testemunhe o máximo.

João Bento Raimundo

Presidente da C. I. do
Instituto Politécnico da Guarda

VARIANTES DO ALGORITMO DE ORDENAÇÃO POR INSERÇÃO

Álvaro Bento Leal - Prof. Coordenador do I.P.G.

Resumo: São apresentadas diversas variantes do algoritmo de ordenação por inserção simples, como sugestões para a investigação de novos métodos de abordagem do problema da ordenação de conjuntos de dimensão média.

O processo de ordenação é sem dúvida o problema lógico para o qual maior número de algoritmos baseados em ideias distintas têm sido desenvolvidos. A descrição de um grande número desses algoritmos pode encontrar-se em [1]

O problema central nesta classe de algoritmos tem sido o da eficiência assintótica, isto é, o número de comparações entre elementos do conjunto a ordenar quando a dimensão atinge valores muito elevados.

Demonstra-se que no caso mais desfavorável o número de comparações é no mínimo da ordem *n. log n*, onde *n* é o número de elementos a ordenar. O algoritmo conhecido como "heapsort" satisfaz esta condição, no entanto, na práctica, verifica-se que o algoritmo "quicksort" de ordem *n. log n* no caso médio, consome, regra geral, menor tempo de processamento no caso de conjuntos de grande dimensão.

A eficiência medida pelo número de comparações entre elementos é o factor dominante quando se trata de avaliar algoritmos a aplicar a grandes conjuntos, podendo mesmo afirmar-se que sob este aspecto não se espera que surjam algoritmos que destronem de forma significativa a preferência que é dada a alguns dos conhecidos actualmente, isto no que se refere ao processamento sequêncial que é ainda dominante nos computadores actuais. Entrando em linha de conta com o processamento paralelo, hoje em dia em franco desenvol-

vimento, então há que esperar progressos significativos na concepção dos algoritmos de ordenação.

Para o caso de pequenos (< ~ 30) e médios (< ~ 200) conjuntos o número de comparações por si só não é suficiente para a avaliação do algoritmo, o número de movimentos entre posições de memória e a simplicidade do programa influenciam significativamente a velocidade de processamento.

Na práctica verifica-se que o algoritmo de inserção, dada a sua simplicidade e, não obstante ocasionar um número médio de comparações da ordem de n^2, supera os restantes, quando o número de elementos do conjunto não ultrapassa um valor da ordem dos 20 a 30.

Têm surgido, ver [2], várias ideias para melhorar o algoritmo de inserção. Assim é possivel recorrer a uma cadeia ordenada onde os elementos vão sendo sucessivamente inseridos. Também foi sugerida a utilização do algoritmo de pesquisa binária para encontrar a posição no subconjunto ordenado onde um novo elemento deve ser inserido. Estas sugestões não deram os resultados prácticos esperados, a perda de simplicidade não compensa a melhoria nos restantes aspectos.

Há no entanto uma variante do algoritmo da inserção com indiscutível interesse práctico que é o algoritmo "shellsort" que, para conjuntos de dimensão média, supera os restantes.

O autor pensa que a pesquisa de variantes do algoritmo de inserção com interesse práctico não está esgotada e recorre a este artigo para a sugestão de um conjunto de caminhos que parecem promissores

No que se segue considera-se que se trata do problema da ordenação ascendente dos elementos representados num vector ("array") de números (ou cadeias de caracteres), x[i] (i=1,2...n).

Lógica de inserção:

Admita-se que os elementos x [i] ($i=i_0, i_0+1,...i_1$) estão ordenados e, se pretende inserir o elemento y nesse conjunto de forma a obter um conjunto ordenado com mais um elemento. O objectivo é realizado pelos seguintes algoritmos:

```
Algoritmo  INSD (x, i0, i1, y):
    Para  j = i1, i1-1 ... i0 enquanto y < x [j] fazer:
      ⌊   x [j+1] : = x [j];
          x [j+1] : = y,
    fim;
```

ou

```
Algoritmo INSE (x, i0, i1, y):
    Para  j=i0, i0+1,...i1 enquanto y > x [j] fazer:
      ⌊   x[j-1]: = x[j];
          x[j-1]: = y;
    fim;
```

para a direita, no caso de INSD, de todos os elementos maiores que y de forma a libertar a posição que corresponde a y no conjunto ordenado, onde esse elemento é inserido. O conjunto ordenado passa a ser formado pelos elementos x[i] ($i = i_0, i_0+1, \ldots i_1+1$).

No caso do algoritmo INSE o deslocamento é efectuado para a esquerda dos elementos menores do que y e o conjunto ordenado passa a ser x[i] ($i = i_0-1, i_0, \ldots i_1$).

Admitindo que y tem um valor aleatório no conjunto inicial, o número médio de elementos que sofrem deslocamento é $1/2.(i_1-i_0+1)$, isto é, o número médio de comparações, necessário para a execução de INSD e INSE, é dado por este valor.

Algoritmo de ordenação por inserção simples:

Para a ordenação completa do conjunto x[i] (i=1,2,... n), começa por se considerar o elemento x[1] como subconjunto ordenado ao qual se adicionam sucessivamente os elementos x[i] (i=2,3 ... n), isto é:

Algoritmo SD(x,n):

Para i=2,3, ... n *fazer*
  Executar INSD (x, 1, i-1, x[i]);
fim;

Ou então, utilizando INSE

Algoritmo SE (x, n):
Para i = n - 1, n - 2, ... 1 *fazer:*
  Executar INSE (x, i + 1, n, x[i]);
fim;

Para o caso de SD o número de comparações no caso médio é:

$$\sum_{i=2} 1/2.(1-1+1) + \sum_{i=3} 1/2.(2-1+1) + \ldots \sum_{i=n} 1/2.(n-1-1+1) = 1/4 .n(n-1)$$

Para o caso de SE o valor que se obtem é o mesmo.

Variante 1

No algoritmo básico SD o conjunto ordenado vai sendo acrescentado sucessivamente de um elemento à direita e vai ocupando a extremidade esquerda do conjunto inicial. De uma forma semelhante no algoritmo SE o subconjunto ordenado ocupa a extremidade direita do conjunto inicial.

Variante de SD e SE é o caso em que o subconjunto ordenado é iniciado em posições centrais do conjunto dado. Tal pode conseguir-se por:

Algoritmo V1 (x, n, k): $\{1< k < n\}$

Para i = k+1, k+2, ... n *fazer*:
  Executar INSD (x, k, i-1, x[i]);
Para i = k-1 , k-2, ... 1 *fazer*:
  Executar INSE (x, i+1, n, x[i]);
fim;

O número de comparações no caso médio é:

$$\overbrace{1/2[(k-k+1)+(k+1-k+1)+...+(n-1-k+1)]}^{1^{o}\ ciclo} + \overbrace{1/2\,[(n-k+1)+(n-k+1+1)+...(n-2+1)]}^{2^{o}\ ciclo}$$

$$\underbrace{\qquad\qquad}_{n-k\ termos} \qquad\qquad \underbrace{\qquad\qquad}_{k-1\ termos}$$

$$= 1/2\,.\,[1/2(n-k+1)\,(n-k)] + 1/2\,[1/2(k-1)\,(n-k+2)] = 1/4\ n(n-1)$$

Conclui-se que V1 é equivalente sob este aspecto a SE e SD.
Equivalente é também o algoritmo que se obtem permutando os dois ciclos em V1.

Variante 2

Dentro da mesma ordem de ideias, uma nova variante é obtida, considerando que os elementos são inseridos alternadamente à direita e à esquerda.

Algoritmo V2 (x, n, k): $\{1 < k \leq n/2\}$

Para j = 1,2, ... k-1 *fazer*:
  Executar INDS (x, k-j+1, k+j-1, x[k+j]);
  Executar INSE (x, k-j+1, k+j, x[k-j]);
Para j = 2k, 2k+1, ... n *fazer*:
  Executar INSD (x, 1, i-1, x[i]);
fim;

O número de comparações para um j qualquer no 1º ciclo é

$$(k+j-1) - (k-j+1)+1 + (k+j) - (k-j+1)+1 = 4j-1$$

O número total de comparações médio no 1º ciclo é:

$$1/2 \sum_{j=1}^{k-1} (4j-1) = 1/2 \left(\frac{4k-2}{2}\right) . (k-1) = 1/2.(2k-1)\,(k-1)$$

$$1/2.\ \frac{(2k-1)+(n-1)}{2}.\ (n-2k+1) = 1/2\ (n/2+k-1)\ (n-2k+1)$$

O total para os 2 ciclos vem:

$$1/2[(2k-1)\ (k-1) + (n/2+k-1)\ (n-2k+1)] = 1/4\ n(n-1)$$

Conclui-se que, sob este aspecto, há equivalência com as variantes anteriores.

Variante 3

Considere-se $1 \leq k \leq n$, a ordenação do conjunto pode processar-se por:

Algoritmo V3 (x, n, k):

Para i = 2, 3, ... k *fazer*:
  Executar INSD (x, 1, i-1, x[i]);
Para i = k+2, k+3, ... n *fazer*:
  Executar INSD (x, k+1, i-1, x[i]);
Para i = k+1, k+2, ... n *fazer*:
  Executar INSD (x, 1, i-1, x[i]);
fim;

O 3º ciclo pode ser condicionado por x[i] < x[i-1], mas o benefício daí resultante não parece, em princípio, compensar a introdução dessa condição.

Os 2 primeiros ciclos são a ordenação pelo SD de cada um dos subconjuntos em que foi particionado o conjunto dado. O 3º ciclo pode interpretar-se como um "merge" dos dois subconjuntos ordenados.

O número de comparações médio nos 1º e 2º ciclo é:

$$\underbrace{1/4\ k(k-1)}_{1º\ ciclo} + \underbrace{1/4\ (n-k)\ (n-k-1)}_{2º\ ciclo}$$

A análise do número de comparações no 3º ciclo necessita de um comentário: o número de elementos inseridos no subconjunto x[i] (i=1, 2 ... k) é n-k, o deslocamento médio dos elementos inseridos é (k+1)/2, porque os elementos do conjunto x[i] (i=k+1, k+2 ... n) que, até dado momento foram inseridos, estão necessáriamente à esquerda do elemento em causa, dada a ordenação efectuada pelo 2º ciclo, assim, para o 3º ciclo, ter-se-á:

$$1/4\ (n-k)\ (k+1)$$

O número de comparações para a ordenação completa é:

$1/4[k(k-1) + (n-k)(n-k-1) + (k+1)(n-k)] = 1/4(n^2-nk+k^2-k)$

O valor mínimo é obtido para $k = n/2$ e é dado por

$1/4[n(n-1) - 1/4(n^2-2n)]$

Pelo que comparando com as variantes anteriores consegue-se uma economia de $1/16(n^2-2n)$ comparações.

Variante 4

Pense-se na partição do conjunto dado em 3 subconjuntos (i=1, 2, ... k), (i=k+1, k+2, ... m) e (i=m+1, m+2, ... n). A ordenação pode obter-se por:

Algoritmo V4 (x, n, k, m):

Para i = 2, 3, ... k *fazer*:
  Executar INSD (x, 1, i-1, x[i]);
Para i = k+2, k+3, ... m *fazer*:
  Executar INSD (x, k+1, i-1, x[i]);
Para i = m+2, m+3, ... n *fazer*:
  Executar INSD (x, m+1, i-1, x[i]);
Para i = k+1, k+2, ... n *fazer*:
  Executar INSD (x, 1, i-1, x[i]);
fm;

O número médio de comparações nos 3 primeiros ciclos é:

$1/4[k(k-1) + (m-k)(m-k-1) + (n-m)(n-m-1)]$

O 4º ciclo, pelas mesmas razões apresentadas na análise de V3, consome um número médio de comparações

$1/4(m-k)(k+1)$ ---> para a inserção dos elementos i = k+1, k+2...m

$1/4(n-m)(m+1)$ ---> para a inserção dos elementos i = m+1, m+2...n

A ordenação completa utilizará um número médio de comparações igual a:

$1/4[k(k-1)+(m-k-1).(m-k)+(n-m).(n-m-1)+(m-k).(k+1)+(n-m)(m+1)]$
$= 1/4(n^2+m^2+k^2-mn-mk-k)$

Para $k=1/3n$ e $m=2/3n$ vem:

$1/4[n(n-1) - 4/9(n^2-n/3)]$

1/9(n²-n/3) comparaçoes.

Nesta linha de raciocínio pode pensar-se na subdivisão em 4,5... intervalos.

Variante 5

Como na variante 4 considere-se o conjunto particionado em 3: (i=1, 2 ... k), (i=k+1, k+2, ...m), (i=m+1, m+2, ...n).

Faça-se a ordenação prévia do 2º conjunto, seguidamente a do conjunto formado pela reúnião do 1º com o 3º e por fim o "merge" de todos eles.

O algoritmo pode desenvolver-se com 3 ciclos iniciais idênticos aos de V4. Para obter o 1º e o 3º subconjuntos ordenados como um só, há que considerar uma alteração de forma no algoritmo INSD.

Algoritmo INSDM (x, k, m, y, t):

Se y < x [k] *então*: x[m+1]: = x[k],
t: = 1;
senão: t: = 0,
fim;

Para j = k-1, k-2, ...1 *enquanto* y < x[j] *fazer*:
x [j+1]: = x[j];
x [j+1]: = y,
fim;

Que é o mesmo que INSD com a diferença de o elemento acrescentado ao conjunto ocupar a posição m+1. A variável *t* indica se houve ou não necessidade de inserir o elemento no subconjunto x[i] (i=1, 2 ... k).

A variante 5 pode então formular-se, por:

Algoritmo V5 (k, m):
Para i = 2, 3, ...k *fazer*:
Executar INSD (x, 1, i-1, x[i]);
Para i = k+2, k+3, ...m *fazer*:
Executar INSD (x, k+1, i-1, x[i]);
Para i = m+2, m+3 ... n *fazer*:
Executar INSD (x, m+1, i-1, x[i]);
t: = 1,
Para i = 1, 2, ... *enquanto* t = 1 *fazer*:
Executar INSDM (x, k, m, x[m+1], t),
Executar INSE (x, m+2, n, x[m+1];
Para i = k+1, k+2, ...n *fazer*:
Executar INSD (x, 1, i-1, x[i]);
fim;

Comparando com V4, foi acrescentado mais um ciclo, o 4º, pelo que aparentemente não há economia de comparações. Na realidade tal não acontece porque o 5º ciclo de V5, embora igual na forma ao 4º de V4 é agora mais eficiente, pelo seguinte: o 1º subconjunto, após a execução de 4º ciclo, está ordenado e contem os menores elementos do conjunto formado pelo 1º e 3º subconjuntos, admitindo uma distribuição inicial aleatória, estes elementos ao serem inseridos no 2º subconjunto terão um percurso médio de 1/4 da dimensão desse conjunto; o mesmo se passa com os elementos do 3º subconjunto.

Veja-se então o cálculo do número médio total de comparações:

$$1/4(m-k)\,(m-k-1) \qquad \text{para o } 2º \text{ ciclo.}$$

O 1º, 3º e 4º ciclos correspondem à ordenação pelo algoritmo V3 de um conjunto com n-m+k elementos, pelo que substituindo n por n-m+k na expressão do número de comparações aí obtido, vem:

$$1/4[(n-m+k)^2 - (n-m+k)\,k+k^2-k]$$

O 5º ciclo pelo que se disse atrás, ocasiona um número médio de comparações igual a:

$$\underbrace{1/4(m-k)k}_{1º \text{ subconjunto}} + \underbrace{1/4(m-k)\,(n-m)}_{3º \text{ subconjunto}}$$

Para o algoritmo completo ter-se-á

$$1/4(n^2+m^2+k^2-mk-mn+nk-m)$$

Uma sugestão será fazer k = n/3 e m = 2/3n o que conduz ao valor de

$$1/4\ n(n-1) - (2/9.n^2 - n/3)$$

Conseguindo-se uma economia de ordem de $2n^2/9$ comparações relativamente a SD ou SE.

Pode objectar-se que a utilização do algoritmo INSDM introduz mais comparações, que é um facto, mas esse número é da ordem de n-m o que comparado com a economia que é de ordem n^2, não invalida a conclusão.

Variante 6

Volte-se de novo à ideia da variante 2 e, para simplificar, considere-se em V2, k = (n-1)/2 e n impar. Neste caso o 2º ciclo de V2 pode retirar-se .

subconjunto ordenado, que ocupa a parte central, os elementos x[k+j] e x[k-j]. Introduza-se uma nova operação antes de inserir estes elementos e que consiste em permutá-los se não estiverem na ordem relativa.

```
Algoritmo  V6 (x, n):

    k: = (n-1)/2,
Para  j = 1, 2, ... k-1 fazer:
  | Se  x[k+j] < x[k-j]
  |     então:  y: = x[k+j], x[k+j]: = x[k-j], x[k-j]: = y;
  |     senão:;
  | Executar  INSD (x, k-j+1, k+j-1, x[k+j]);
  | Executar  INSE (x, k-j+1, k+j, x[k-j]);
fim;
```

Se se admitir que as k-1 comparações e as eventuais permutas dos elementos não pesam na eficiência, conclui-se que há melhoria relativamente a V2, porque o maior dos dois elementos é introduzido à direita e o menor à esquerda, pelo que os percursos dentro do subconjunto ordenado são menores em média, do que no caso em que essa precaução não foi tomada.

A análise neste caso é bastante mais complicada. Uma estimativa pode obter-se pelos seguintes argumentos, para uma distribuição aleatória.

- No caso de V4, em 1/2 das inserções efectuadas os elementos estão na ordem correcta, pelo que a melhoria introduzida em V6 só afecta metade dos elementos.
- No caso de ter havido permuta ela pode ter ocorrido entre elementos que pertençam ambos à parte direita do conjunto ordenado ou ambos à parte esquerda desse mesmo conjunto, a probabilidade de que, no caso de haver permuta ela seja realmente efectiva é de 1/2.

Para os casos em que a permuta foi efectiva o percurso dos elementos no conjunto ordenado é em média metade do caso geral. Pelo que atendendo a que para V2 se obteve-se um número médio de comparações dado por 1/4 n(n-1), no caso de V6 ter-se-á

$$\underset{\text{não efectivas}}{3/4 \,.\, 1/4\, n(n-1)} + \underset{\text{efectivas}}{1/4 \,.\, 1/8\, n(n-1)} = 7/8 \,.\, 1/4\, n(n-1)$$

isto é, 12,5% de economia, ao que há a deduzir as 1/2.(n-1)-1 comparações adicionais do algoritmo V6.

Variante 7

A mesma ideia da variante 6 mas com a ordenação prévia dos subconjuntos x[i] (i = 1, 2, ... k-1) e x[i] (i = k+1, k+2, ... n):

```
Algoritmo  V7 (x, n):

    k: = (n-1)/2,
  Para  i = 2, 3, ... k-1  fazer:
   ⌊ Executar INSD (x, 1, i-1, x[i]);
  Para  i = k+2, k+3, ... n  fazer:
   ⌊ Executar  INSD [x, k+1, i-1, x[i]);
  Para  j = 1, 2, ... k-1  fazer:
   | Se  x[k+j] < x[k-j]
   |    então:  y: = x[k+j], x[k+j]: = x[k-j], x[k-j]: = y;
   |    senão:;
   | Executar  INSD (x, k-j+1, k+j-1, x[k+j]);
   ⌊ Executar  INSE (x, k-j+1, k+j, x[k-j]);
  fim;
```

Relativamente a V6 e pelas mesmas considerações, ter-se--á neste caso, que quando a permuta existe ela é sempre efectiva donde

$$1/2 \,.\, 1/4\, n(n-1) + 1/2 \,.\, 1/8\, n(n-1) = 3/4 \,.\, 1/4\, n(n-1)$$

a que há que adicionar $2.\ 1/4\ k(k-1)$ comparações do 1º e 2º ciclo de V7. No total ter-se-á:

$$3/4\,.1/4\,n(n-1) + 2\,.\,1/4\,(\frac{n-1}{2})\,(\frac{n-1}{2} - 1) = 7/8\,.\,1/4\,n(n-1) - 3/8(n-1)$$

isto é, V7 é aproximadamente equivalente a V6.

Variante 8

Seguindo o mesmo princípio da variante anterior, suponha-se o conjunto dividido em 3 subconjuntos, de igual dimensão, para simplificar.

```
Algoritmo  V8 (x, n):          { n múltiplo de 3 }
   k: = n/3
Para   i = 2, 3, ... k  fazer:
 ⌊    Executar   INSD (x, 1, i-1, x[i]);
Para   i = k+2, k+3, ... 2k  fazer:
 ⌊    Executar   INSD (x, k+1, i-1, x[i]);
Para   i = 2k+2, 2k+3, ... n  fazer:
 ⌊    Executar   INSD (x, 2k+1, i-1, x[i]);
Para   j = 1, 2, ... k  fazer:
 | Se  x[2k+j] < x(k-j+1)
 |     então:  y: =x[2k+j], x[2k+j]:= x[k-j+1], x[k-j+1]: = y;
 |     senão:;
 |    Executar  INSD (x, k-j+2, 2k+j-1, x[2k+j]);
 ⌊    Executar  INSE (x, k-j+2, 2k+j, x[k-j+1]);
fim;
```

$3 \cdot 1/4.k(k-1) = 1/3 \cdot 1/4.n(n-3)$ comparações em média.

No 4º ciclo, haverá um número médio de comparações dado por:

INSD INSE

$j=1 \rightarrow 1+1/2 \cdot 1/2(k+2.1-3+1)+1/2 \cdot 1/2(k+2.1-2+1)=k/2+5/4$

Se factor de eficiência resultante da permuta

$j=2 \rightarrow 1+1/2 \cdot 1/2(k+2.2-3+1)+1/2 \cdot 1/2(k+2.2-2+1)=k/2+9/4$

.
.
.

no total para o 4º ciclo é

$$\sum_{j=1}^{k} (1/4+k/2+j) = k/4+k^2/2 + k\left(\frac{k+1}{2}\right) = k^2 + 3/4.k$$

$$= n^2/9+n/4$$

No total do algoritmo V8 ter-se-á:

$$1/3 \cdot 1/4.n(n-3) + n^2/9 + n/4 = 17/108.n^2$$

Relativamente ao algoritmo básico há uma economia da ordem de $5/54.n^2 - n/4$ operações de comparação.

Variante 9

Considere-se um conjunto com algumas dezenas de elementos. Divida-se em subconjuntos de dimensão k, k = n/m (m número de subconjuntos). Seleccione-se k de forma a estar compreendido entre 10 e 20, para o que se sabe ser a dimensão ideal de aplicação do algoritmo de inserção simples. Ordenem--se separadamente pelo algoritmo SD os *m* subconjuntos. Aplique-se aos 3 subconjuntos centrais o algoritmo V8. Seguidamente pelo processo indicado no 4º ciclo de V8 faça-se a inserção no conjunto central, já ordenado, dos dois subconjuntos vizinhos à direita e à esquerda, e assim sucessivamente até todos os subconjuntos terem sido inseridos ordenadamente.

Dada a eficiência de SD para pequenos conjuntos e atendendo à economia que consegue com a variante 8, é de esperar que a ordenação por este processo seja realmente eficaz para uma dimensão do conjunto superior à que é de interesse quando se usa o algoritmo de inserção simples.

Conclusões

Com base em algumas sugestões, que tanto quanto é do conhecimento do autor, não são referidas na literatura, desenvolveram-se várias variantes do algoritmo de ordenação por inserção que a análise da eficiência mostrou serem vantajosas, sem perder a grande virtude do algoritmo de inserção que é a sua simplicidade.

O que foi exposto sugere que muitas mais variantes é possível estabelecer, eventualmente com resultados mais promissores.

A etapa seguinte a este trabalho é uma investigação práctica das sugestões apresentadas de forma a definir com maior exactidão os limites de aplicabilidade eficiente de cada uma delas.

Referências:

[1] - KNUTH: "The Art of Computer Programming - vol 3: Sorting and Searching" - Ed. Addison-Wesley.

[2] - A. TENENBAUM, M. AUGENSTEIN: "Data Structures Using Pascal" - Ed. Prentice-Hall.