Eficiência da busca

Ordenação – insertion sort e selection sort

Instituto de Computação – Unicamp

1 de Abril de 2014

Roteiro

Eficiência dos Algoritmos

Podemos medir a eficiência de qualquer algoritmo analisando a quantidade de recursos (tempo, memória, banda de rede, etc.) que o algoritmo usa para resolver o problema para o qual foi proposto.

- É comum medir a eficiência em relação ao tempo. Para isso, analisamos quantas instruções um algoritmo usa para resolver o problema.
- Podemos fazer uma análise simplificada dos algoritmos de busca analisando a quantidade de vezes que os algoritmos **acessam** uma posição do vetor.

Eficiência dos Algoritmos

No caso da busca sequencial existem três possibilidades:

- Na melhor das hipóteses a chave de busca estará na posição 0. Portanto teremos um único acesso em **vet[0]**.
- Na pior das hipóteses, a chave é o último elemento ou não pertence ao vetor, e portanto acessaremos todas as *tam* posições do vetor.
- É possível mostrar que se uma chave qualquer pode ser requisitada com a mesma probabilidade, então o número de acessos será

$$(\text{tam} + 1)/2$$

na média.

Eficiência dos Algoritmos

No caso da busca binária temos as três possibilidades:

- Na melhor das hipóteses a chave de busca estará na posição do meio. Portanto teremos um único acesso.
- Na pior das hipóteses, teremos $(\log \text{tam})$ acessos.
  - Para ver isso note que a cada verificação de uma posição do vetor, o tamanho do vetor considerado é dividido pela metade. No pior caso repetimos a busca até o vetor considerado ter tamanho 1. Se você pensar um pouco, o número de acessos x pode ser encontrado resolvendo-se a equação:

    $$\frac{\text{tam}}{2^x} = 1$$

    cuja solução é $x = (\log_2 \text{tam})$.
- É possível mostrar que se uma chave qualquer pode ser requisitada com a mesma probabilidade, então o número de acessos será

  $$(\log_2 \text{tam}) - 1$$

  na média.

Eficiência dos Algoritmos

Para se ter uma idéia da diferença de eficiência dos dois, considere que temos um cadastro com 10^6 (um milhão) de itens.

- Com a busca sequencial, a procura de um item qualquer gastará na média

$$(10^6 + 1)/2 \approx 500000 \text{ acessos.}$$

- Com a busca binária teremos

$$(\log_2 10^6) - 1 \approx 20 \text{ acessos.}$$

Eficiência dos Algoritmos

Mas uma ressalva deve ser feita: Para utilizar a busca binária, o vetor precisa estar ordenado!

- Se você tiver um cadastro onde vários itens são removidos e inseridos com frequência, e a busca deve ser feita intercalada com estas operações, então a busca binária pode não ser a melhor opção, já que você precisará ficar mantendo o vetor ordenado.
- Caso o número de buscas feitas seja muito maior quando comparada com outras operações, então a busca binária é uma boa opção.

Ordenação

- Continuamos com o estudo de algoritmos para o problema de ordenação:

Dada uma coleção de elementos com uma relação de ordem entre si, devemos gerar uma saída com os elementos ordenados.

- Novamente usaremos um vetor de inteiros como exemplo de coleção a ser ordenada.

Insertion-Sort

- Seja **vet** um vetor contendo números inteiros, que devemos deixar ordenado.
- A idéia do algoritmo é a seguinte:
  - A cada passo, uma porção de 0 até $i - 1$ do vetor já está ordenada.
  - Devemos inserir o item da posição i na posição correta para deixar o vetor ordenado até a posição i.
  - No passo seguinte consideramos que o vetor está ordenado até i.

Insertion-Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i

$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion-Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion-Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion-Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i

$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion-Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i

$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion-Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i

$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion-Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i

$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion-Sort

- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
j=i;
while(j>0){  //neste laço vamos trocando v[i] com posições
             //anteriores até acharmos a posição correta de v[i]
  if(vet[j] < vet[j-1]){
     aux = vet[j-1];
     vet[j-1] = vet[j];
     vet[j] = aux;
     j--;
  }else{
      break;
  }
}
```

Insertion-Sort

- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
j=i;
while(j>0){  //neste laço vamos trocando v[i] com posições
             //anteriores até acharmos a posição correta de v[i]
  if(vet[j] < vet[j-1]){
     aux = vet[j-1];
     vet[j-1] = vet[j];
     vet[j] = aux;
     j--;
  }else{
      break;
  }
}
```

Insertion-Sort

- Vamos apresentar uma forma alternativa de fazer a mesma coisa do slide passado.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];  //inserir aux na posição correta
j = i - 1; //analisar elementos das j posições anteriores

while( ( j >=0 ) && ( vet[j] > aux) ){
    vet[ j+1 ] = vet[ j ];    // enquanto vet[j] > aux empurra
     j --;                    // vet[j] para frente
}

//Quando terminar o laço:
//    OU j == -1, significando que você empurrou v[0] para frente
//    OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
vet[ j+1 ] = aux;
}
```

Insertion-Sort

- Vamos apresentar uma forma alternativa de fazer a mesma coisa do slide passado.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];  //inserir aux na posição correta
j = i - 1; //analisar elementos das j posições anteriores

while( ( j >=0 ) && ( vet[j] > aux) ){
    vet[ j+1 ] = vet[ j ];    // enquanto vet[j] > aux empurra
     j --;                    // vet[j] para frente
}

//Quando terminar o laço:
//    OU j == -1, significando que você empurrou v[0] para frente
//    OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
vet[ j+1 ] = aux;
}
```

Insertion-Sort

- Vamos apresentar uma forma alternativa de fazer a mesma coisa do slide passado.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];  //inserir aux na posição correta
j = i - 1; //analisar elementos das j posições anteriores

while( ( j >=0 ) && ( vet[j] > aux) ){
    vet[ j+1 ] = vet[ j ];    // enquanto vet[j] > aux empurra
     j --;                    // vet[j] para frente
}

//Quando terminar o laço:
//    OU j == -1, significando que você empurrou v[0] para frente
//    OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
 vet[ j+1 ] = aux;
 }
```

Insertion-Sort

- Vamos apresentar uma forma alternativa de fazer a mesma coisa do slide passado.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];  //inserir aux na posição correta
j = i - 1; //analisar elementos das j posições anteriores

while( ( j >=0 ) && ( vet[j] > aux) ){
    vet[ j+1 ] = vet[ j ];    // enquanto vet[j] > aux empurra
     j --;                    // vet[j] para frente
}

//Quando terminar o laço:
//   OU j == -1, significando que você empurrou v[0] para frente
//   OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
 vet[ j+1 ] = aux;
 }
```

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4)$: $aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4)$: $aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4)$: $aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4)$: $aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4) : aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4) : aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4) : aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4) : aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4)$: $aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4)$: $aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4)$: $aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4)$: $aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4)$: $aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4)$: $aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4)$: $aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4)$: $aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4)$: $aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4)$: $aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4)$: $aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4)$: $aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4)$: $aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4)$: $aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4)$: $aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4)$: $aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.
$(1, 3, 5, 10, \underline{20}, 2, 4)$: $aux = 2; j = 4;$
$(1, 3, 5, \underline{10}, 20, 20, 4)$: $aux = 2; j = 3;$
$(1, 3, \underline{5}, 10, 10, 20, 4)$: $aux = 2; j = 2;$
$(1, \underline{3}, 5, 5, 10, 20, 4)$: $aux = 2; j = 1;$
$(\underline{1}, 3, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$
Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$
$(1, 2, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

```
int main(){
    int i,j, aux, vet[6]={3,2,5,1,90,6},tam=6;

    for(i=1; i<tam; i++){

        aux = vet[i];
        j=i-1;

        while( (j>=0) && (vet[j] > aux)  ){
            vet[j+1] = vet[j];
            j--;
        }
        vet[j+1] = aux;
    }
}
```

```
int main(){
    int i,j, aux, vet[6]={3,2,5,1,90,6},tam=6;

    for(i=1; i<tam; i++){

        aux = vet[i];
        j=i-1;

        while( (j>=0) && (vet[j] > aux)  ){
            vet[j+1] = vet[j];
            j--;
        }
        vet[j+1] = aux;
    }
}
```

```
int main(){
   int i,j, aux, vet[6]={3,2,5,1,90,6},tam=6;

   for(i=1; i<tam; i++){

      aux = vet[i];
      j=i-1;

      while( (j>=0) && (vet[j] > aux)  ){
         vet[j+1] = vet[j];
         j--;
      }
      vet[j+1] = aux;
   }
}
```

```
int main(){
   int i,j, aux, vet[6]={3,2,5,1,90,6},tam=6;

   for(i=1; i<tam; i++){

      aux = vet[i];
      j=i-1;

      while( (j>=0) && (vet[j] > aux)  ){
         vet[j+1] = vet[j];
         j--;
      }
      vet[j+1] = aux;
   }
}
```

Selection-Sort

- Seja **vet** um vetor contendo números inteiros.
- Devemos deixar **vet** em ordem crescente.
- A idéia do algoritmo é a seguinte:
  - Ache o menor elemento a partir da posição 0. Troque então este elemento com o elemento da posição 0.
  - Ache o menor elemento a partir da posição 1. Troque então este elemento com o elemento da posição 1.
  - Ache o menor elemento a partir da posição 2. Troque então este elemento com o elemento da posição 2.
  - E assim sucessivamente...

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

- Como achar o menor elemento a partir de uma posição inicial?
- Vamos achar **o índice** do menor elemento em um vetor, a partir de uma posição inicial:

```
int min = inicio, j;
for(j=inicio+1; j<tam; j++){
  if(vet[min] > vet[j])
    min = j;
}
```

Selection-Sort

- Como achar o menor elemento a partir de uma posição inicial?
- Vamos achar **o índice** do menor elemento em um vetor, a partir de uma posição inicial:

```
int min = inicio, j;
for(j=inicio+1; j<tam; j++){
  if(vet[min] > vet[j])
    min = j;
}
```

Selection-Sort

- Dado o código anterior para achar o índice do menor elemento, como implementar o algoritmo de ordenação?
- Ache o menor elemento a partir da posição 0, e troque com o elemento da posição 0.
- Ache o menor elemento a partir da posição 1, e troque com o elemento da posição 1.
- Ache o menor elemento a partir da posição 2, e troque com o elemento da posição 2.
- E assim sucessivamente...

```
int main(){
  int vetor[10]={14,7,8,34,56,4,0,9,-8,100};
  int i, min, aux, tam=10;
  for(i=0;i<tam;i++)
      printf("%d, ",vetor[i]);
  printf(")");

  for(i=0; i<tam; i++){
    min=i;
    for(j=i+1; j<tam; j++){
       if(vet[min] > vet[j])
       min = j;
    }
    aux = vet[i];
    vet[i] = vet[min];
    vet[min] = aux;
  }
  for(i=0;i<tam;i++)
    printf("%d, ",vetor[i]);
  printf(")\n");  return 0;
}
```

Exercício

- Altere os algoritmos de ordenação vistos nesta aula para que estes ordenem um vetor de inteiros em ordem decrescente ao invés de ordem crescente.