Análise Projeto de Algoritmos

Loana Tito Nogueira

09-Abril-2012

Projeto de Algoritmos por Divisão e Conquista

- **Dividir para Conquistar:** tática de guerra aplicada ao projeto de algoritmos

- Um algoritmo de divisão e conquista é aquele que resolve o problema desejado combinando as soluções parciais de (um ou mais) subproblemas, obtidas recursivamente.

Projeto de Algoritmos por Divisão e Conquista

- Seja H um problema algorítmico. O método da divisão e conquista consiste em tentar resolver H, através dos seguintes passos:
  - Decompor H em subproblemas $H_1, H_2, ..., H_k$

Menores que H

Projeto de Algoritmos por Divisão e Conquista

- Seja H um problema algorítmico. O método da divisão e conquista consiste em tentar resolver H, através dos seguintes passos:
  - Decompor H em subproblemas $H_1, H_2, ..., H_k$

Menores que H

  - Resolver cada subproblema H_i, através de uma aplicação recursiva desse método

Projeto de Algoritmos por Divisão e Conquista

- Seja H um problema algorítmico. O método da divisão e conquista consiste em tentar resolver H, através dos seguintes passos:
  - Decompor H em subproblemas $H_1, H_2, ..., H_k$

Menores que H

  - Resolver cada subproblema H_i, através de uma aplicação recursiva desse método
  - Compor a solução de H, a partir das soluções de H_1, H_2, ..., H_k

Algoritmo Genérico – Divisão e Conquista

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

Entrada: A instância x

Saída: Solução y da instância x

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

  Entrada: A instância x

  Saída: Solução y da instância x

  Se x é suficientemente pequeno **então**

    Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

  Entrada: A instância x

  Saída: Solução y da instância x

  Se x é suficientemente pequeno **então**

  Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

  Senão

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

  Entrada: A instância x

  Saída: Solução y da instância x

  Se x é suficientemente pequeno **então**

    Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

  Senão

    Divisão

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

  Entrada: A instância x

  Saída: Solução y da instância x

  Se x é suficientemente pequeno **então**

    Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

  Senão

    Divisão

      Decomponha x em instâncias menores x_1, ..., x_k

      Para i=1 até k **faça** y_i=**DivConq**(x_i)

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

  Entrada: A instância x

  Saída: Solução y da instância x

  Se x é suficientemente pequeno **então**

    Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

  Senão

    Divisão

      Decomponha x em instâncias menores x_1, ..., x_k

      Para i=1 até k **faça** y_i=**DivConq**(x_i)

    Conquista

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

  Entrada: A instância x

  Saída: Solução y da instância x

  Se x é suficientemente pequeno **então**

    Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

  Senão

    Divisão

      Decomponha x em instâncias menores x_1, ..., x_k

      Para i=1 até k **faça** y_i=**DivConq**(x_i)

    Conquista

      Combine as soluções y_i para obter a solução y de x

Algoritmo Genérico – Divisão e Conquista

- **DivConq**(x)

  Entrada: A instância x

  Saída: Solução y da instância x

  Se x é suficientemente pequeno **então**

    Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

  Senão

    Divisão

      Decomponha x em instâncias menores x_1, ..., x_k

      Para i=1 até k **faça** y_i=**DivConq**(x_i)

    Conquista

      Combine as soluções y_i para obter a solução y de x

  Retorne(y)

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n>0$ e real a, sei calcular a^{n-1}

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n>0$ e real a, sei calcular a^{n-1}

Passo da Indução: Queremos provar que conseguimos calcular a^n, para $n>0$.

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n>0$ e real a, sei calcular a^{n-1}

Passo da Indução: Queremos provar que conseguimos calcular a^n, para $n>0$.

Sei calcular a^{n-1} ➡ calculo a^n multiplicando a^{n-1} por a

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

```
Se n = 0 então
        Retorne(1) {caso base}
Senão
        an' := Expon(a, n-1)
        an := an'*a
Retorne(an)
```

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

```
Se n = 0 então
        Retorne(1) {caso base}
Senão
        an' := Expon(a, n-1)
        an := an'*a
Retorne(an)
```

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

```
Se n = 0 então
        Retorne(1) {caso base}
Senão
        an' := Expon(a, n-1)
        an := an'*a
Retorne(an)
```

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

```
Se n = 0 então
        Retorne(1) {caso base}
Senão
        an' := Expon(a, n-1)
        an := an'*a
Retorne(an)
```

Complexidade: Expon(a,n)

- T(n): nº de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1, & n=0 \\ T(n-1)+ 1, & n>0 \end{cases}$$

Complexidade: Expon(a,n)

- T(n): nº de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1, & n=0 \\ T(n-1)+ 1, & n>0 \end{cases}$$

$T(n)= O(n)$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: n=0; $a^0 = 1$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n>0$ e real a, sei calcular a^k, para todo $k<n$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n>0$ e real a, sei calcular a^k, para todo $k<n$

Passo da Indução: Queremos provar que conseguimos calcular a^n, para $n>0$.

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n>0$ e real a, sei calcular a^k, para todo $k<n$

Passo da Indução: Queremos provar que conseguimos calcular a^n, para $n>0$.

Sei calcular $a^{\lfloor n/2 \rfloor}$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
  - **Problema:** Calcular a^n, para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

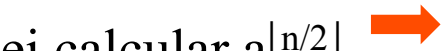
Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n>0$ e real a, sei calcular a^k, para todo $k<n$

Passo da Indução: Queremo

$$\begin{cases} (a^{\lfloor n/2 \rfloor})^2, \text{ se n é par} \\ a^n \\ a.\ (a^{\lfloor n/2 \rfloor})^2, \text{ se n é ímpar} \end{cases}$$

para $n>0$.

Sei calcular $a^{\lfloor n/2 \rfloor}$ 

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n
Saída: O valor de a^n

Se n = 0 **então**

Retorne(1) {caso base}

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n
Saída: O valor de a^n

Se n = 0 **então**

    Retorne(1) {caso base}

Senão

    divisão

        an' := Exp(a, n div 2)

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n
Saída: O valor de a^n

Se n = 0 **então**
    Retorne(1) {caso base}
Senão
    divisão
        an' := Exp(a, n div 2)
    conquista
        an := an'*an'
        Se (n mod 2)=1 **então** an:= an* a

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n
Saída: O valor de a^n

Se n = 0 **então**
    Retorne(1) {caso base}
Senão
    divisão
        an' := Exp(a, n div 2)
    conquista
        an := an'*an'
        Se (n mod 2)=1 **então** an:= an* a
Retorne(an)

Complexidade

- T(n): nº de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1 & , n=0 \\ T(\lfloor n/2 \rfloor) + c & , n>0 \end{cases}$$

Complexidade

- T(n): nº de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1 & , n=0 \\ T(\lfloor n/2 \rfloor) + c & , n>0 \end{cases}$$

$T(n) = O(\log n)$

Min e Max de S

- **Problema:** Dado conjunto S com n elementos, determinar o mínimo e o máximo de S

Min e Max de S

- **Problema:** Dado conjunto S com n elementos, determinar o mínimo e o máximo de S

- Solução: Aplicar a técnica de Divisão e Conquista

Min e Max de S

- **Problema:** Dado conjunto S com n elementos, determinar o mínimo e o máximo de S

- Solução: Aplicar a técnica de Divisão e Conquista

- Idéia: Dividir S em 2 subconjuntos S_1 e S_2.

- $S_1 = \{s_1, s_2, ..., s_{n/2}\}$
- $S_2 = \{s_{n/2+1}, ..., s_n\}$, $n > 2$

Min e Max de S

- Idéia: Dividir S em 2 subconjuntos S_1 e S_2.
- $S_1 = \{s_1, s_2, ..., s_{n/2}\}$
- $S_2 = \{s_{n/2+1}, ..., s_n\}$, $n > 2$

- Resolver o problema para S_1 e S_2.

- $Min(S) = \min\{\min(S_1), \min(S_2)\}$
- $Max(S) = \max\{\max(S_1), \max(S_2)\}$

Algoritmo MinMax

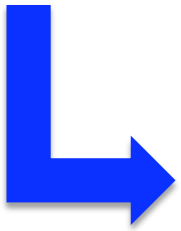
- Se n=2 então
  - Seja S={a,b}
  - Retornar (min{a,b}, max{a,b})
  - Senão
    - S_1= subconjunto dos n/2 elementos inicias de S
    - S_2= subconjunto dos n/2 elementos finais de S
    - (a,b)= MinMax(S_1)
    - (c,d)= MinMax(S_2)
  - Retornar (min{a,c}, max{b,d})

Número de Comparações

$$T(n) = \begin{cases} 1, \text{ se } n = 2 \\ 2T(n/2) + 2, \text{ se } n > 2 \end{cases}$$

Número de Comparações

$$T(n) = \begin{cases} 1, \text{ se } n = 2 \\ 2T(n/2) + 2, \text{ se } n > 2 \end{cases}$$



$$T(n) = 3n/2 - 2$$

Teorema: O problema MinMax requer pelo menos 3n/2 – 2 coparações

O Algoritmo MinMax é ótimo

Busca Binária

- **Problema:** Dado um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que A[i]=x, ou que não existe tal i.

Busca Binária

- **Problema:** Dado um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que A[i]=x, ou que não existe tal i.

- **INDUÇÃO FORTE!!**

Busca Binária

- **Problema:** Dado um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que A[i]=x, ou que não existe tal i.

- **INDUÇÃO FORTE!!**

  - **Como o vetor está ordenado, conseguimos determinar, com apenas uma comparação, que metade das posições do vetor não podem conter o valor x**

Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que A[j]= x ou j=0

```
Se i=f então se A[i] =x então retorne(i)
      senão retorne(0)
senão
      j:=(i+f) div 2
      se A[j]=x retorne(j)
      senão se A[j]>x então
                  j:=BuscaBinaria(A, i,f-1, x)
                  senão
                              j:= BuscaBinaria(A, i+1, f, x)
retorne(j)
```

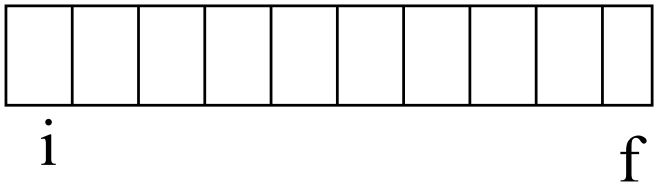
Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice 1≤ j ≤ n tal que A[j]= x ou j=0

Se i=f **então se** A[i] =x **então retorne**(i)

senão retorne(0)



senão

j:=(i+f) div 2

se A[j]=x **retorne**(j)

senão se A[j]>x **então**

j:=**BuscaBinaria**(A, i,f-1, x)

senão

j:= **BuscaBinaria**(A, i+1, f, x)

retorne(j)

Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice 1≤ j ≤ n tal que A[j]= x ou j=0

Se i=f **então se** A[i] =x **então retorne**(i)

senão retorne(0)

senão

j:=(i+f) div 2

se A[j]=x **retorne**(j)

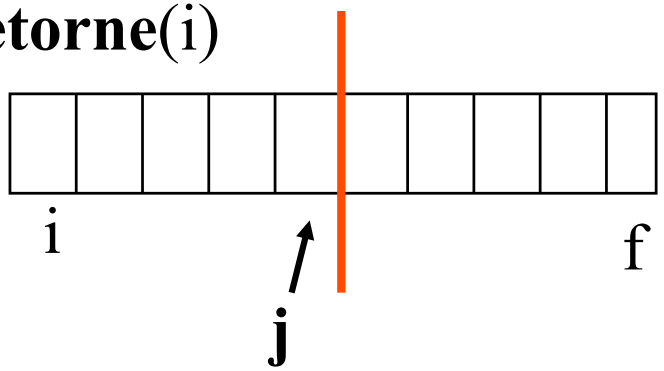
senão se A[j]>x **então**

j:=**BuscaBinaria**(A, i,f-1, x)

senão

j:= **BuscaBinaria**(A, i+1, f, x)

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice 1≤ j ≤ n tal que A[j]= x ou j=0

Se i=f **então se** A[i] =x **então retorne**(i)

senão retorne(0)

senão

j:=(i+f) div 2

se A[j]=x **retorne**(j)

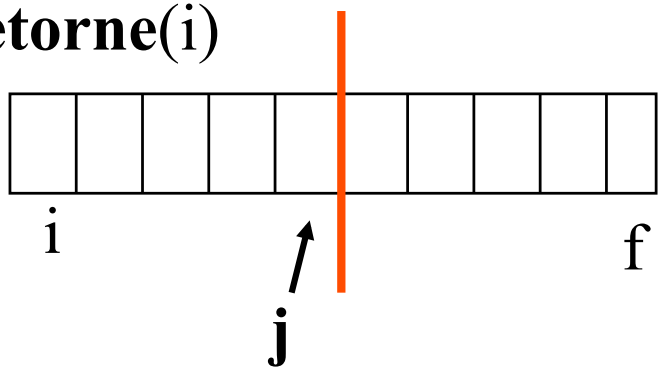
senão se A[j]>x **então**

j:=**BuscaBinaria**(A, i,f-1, x)

senão

j:= **BuscaBinaria**(A, i+1, f, x)

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice 1≤ j ≤ n tal que A[j]= x ou j=0

Se i=f **então se** A[i] =x **então retorne**(i)

senão retorne(0)

senão

j:=(i+f) div 2

se A[j]=x **retorne**(j)

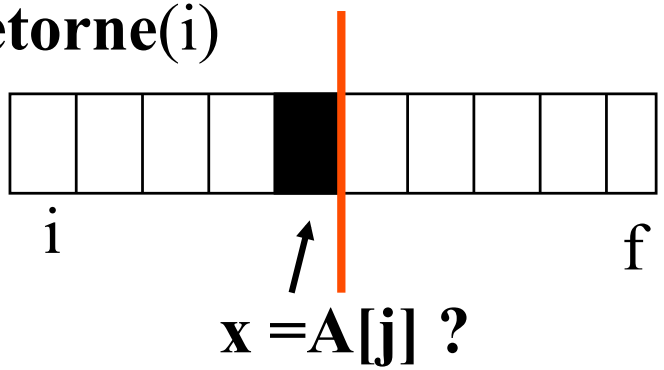
senão se A[j]>x **então**

j:=**BuscaBinaria**(A, i,f-1, x)

senão

j:= **BuscaBinaria**(A, i+1, f, x)

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice 1≤ j ≤ n tal que A[j]= x ou j=0

Se i=f **então se** A[i] =x **então retorne**(i)

senão retorne(0)

senão

j:=(i+f) div 2

se A[j]=x **retorne**(j)

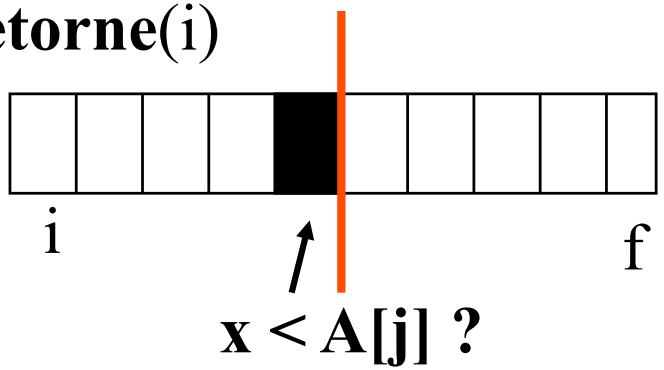
senão se A[j]>x **então**

j:=**BuscaBinaria**(A, i,j-1, x)

senão

j:= **BuscaBinaria**(A, i+1, f, x)

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice 1≤ j ≤ n tal que A[j]= x ou j=0

Se i=f **então se** A[i] =x **então retorne**(i)

senão retorne(0)

senão

j:=(i+f) div 2

se A[j]=x **retorne**(j)

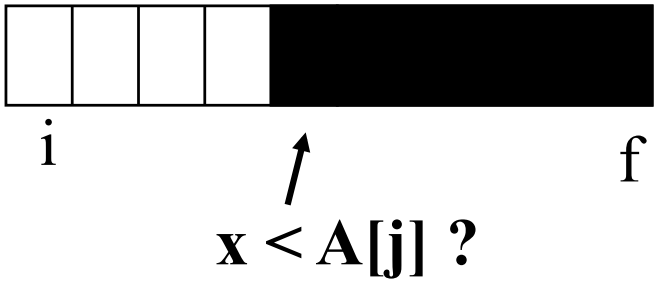
senão se A[j]>x **então**

j:=**BuscaBinaria**(A, i,j-1, x)

senão

j:= **BuscaBinaria**(A, i+1, f, x)

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que A[j]= x ou j=0

```
Se i=f então se A[i] =x então retorne(i)
      senão retorne(0)
senão
      j:=(i+f) div 2
      se A[j]=x retorne(j)
      senão se A[j]>x então
            j:=BuscaBinaria(A, i,f-1, x)
            senão
                  j:= BuscaBinaria(A, i+1, f, x)
retorne(j)
```

Complexidade

- CALCULE!

Ordenação

- **Problema:** Ordenar um conjunto de n ≥ 1 inteiros

  - Diversos algoritmos para o problema de ordenação através de indução

Ordenação

- **Indução Simples:**
  - **Hipótese de Indução:**

    Sabemos ordenar um conjunto de n-1 ≥ 1 inteiros

Ordenação

- **Indução Simples:**
  - **Hipótese de Indução:**

    Sabemos ordenar um conjunto de n-1 ≥ 1 inteiros

    Caso base: n=1. Já está ordenado!!

Ordenação

- **Indução Simples:**
  - **Hipótese de Indução:**

    Sabemos ordenar um conjunto de n-1 ≥ 1 inteiros

    Caso base: n=1. Já está ordenado!!

    Passo da Indução: Seja S um conjunto com n>1 inteiros e x um elemento de S.

Ordenação

- **Indução Simples:**
  - **Hipótese de Indução:**

    Sabemos ordenar um conjunto de n-1 ≥ 1 inteiros

    Caso base: n=1. Já está ordenado!!

    Passo da Indução: Seja S um conjunto com n>1 inteiros e x um elemento de S.

    Por h.i. sabemos ordenar S-x, basta então inserir x na posição correta para obtermos S ordenado

Ordenação

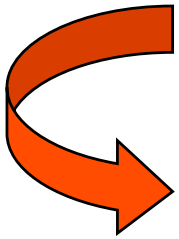
- **Indução Simples:**
  - **Hipótese de Indução:**

    Sabemos ordenar um conjunto de n-1 ≥ 1 inteiros

    Caso base: n=1. Já está ordenado!!

    Passo da Indução: Seja S um conjunto com n>1 inteiros e x um elemento de S.

    Por h.i. sabemos ordenar S-x, basta então inserir x na posição correta para obtermos S ordenado



INSERTION SORT!!!

Algoritmo: OrdenaçãoInserção(A, n)

Entrada: Vetor A e n inteiro
Saída: Vetor A ordenado

```
Se n≥ 2 faça
    OrdenaçãoInserção(A, n-1)
    v:= A[n]
    j:= n
    enquanto (j >1) e (A[j-1]) > v) faça
            A[j]:=A[j-1]
            j:=j-1
    A[j]:=v
```

Algoritmo: OrdenaçãoInserção(A, n)

Entrada: Vetor A e n inteiro
Saída: Vetor A ordenado

```
Se n≥ 2 faça
    OrdenaçãoInserção(A, n-1)
    v:= A[n]
    j:= n
    enquanto (j >1) e (A[j-1]) > v) faça
            A[j]:=A[j-1]
            j:=j-1
    A[j]:=v
```

Versão Recursiva

Complexidade

- Tanto o número de comparações quanto de trocas é dado pela recorrência:

$$T(n) = \begin{cases} 0, & n=1 \\ T(n-1) + n, & n>1 \end{cases}$$

Algoritmo: OrdenaçãoInserção(A, n)

Entrada: Vetor A e n inteiro
Saída: Vetor A ordenado

```
Para i:=2 até n faça
        v:= A[n]
        j:= n
        enquanto (j >1) e (A[j-1]) > v) faça
                A[j]:=A[j-1]
                j:=j-1
        A[j]:=v
```

Versão Iterativa

2a. alternativa

- **Hipótese de Indução:**

  Sabemos ordenar um conjunto de n-1 ≥ 1 inteiros

Caso base: OK!

Passo da Indução: Encontrar o mínimo x.

Sabemos ordenar S\x.

Selection Sort

Selection Sort(A, i, n)

Entrada: Vetor A, inicio e fim
Saída: Vetor A ordenado

```
Se i < n faça
    min:=i
    para j:= i+1 até n faça
          se A[j] < A[min] então min:=j
    t:= A[min]
    A[min]:= A[i]
    A[i]:= t
    SelectionSort(a, i+1, n)
```

Complexidade

- Número de comparações:
  - T(n)= 0, se n=1
  - T(n)= T(n-1)+n, se n >1

- Número de trocas:

  - T(n)= 0, se n=1
  - T(n)= T(n-1) + 1, n > 1

Complexidade

- Número de comparações:
  - T(n)= 0, se n=1
  - T(n)= T(n-1)+n, se n >1

$O(n^2)$

- Número de trocas:
  - T(n)= 0, se n=1
  - T(n)= T(n-1) + 1, n > 1

$O(n)$

Versão Iterativa

```
Para i:=1 até n-1 faça
    min:=i
    para j:= i+1 até n faça
          se A[j] < A[min] então min:=j
    t:= A[min]
    A[min]:= A[i]
    A[i]:= t
```

Indução Forte

- **Hipótese de Indução:**

  Sabemos ordenar um conjunto de $1 \leq k \leq n$

Indução Forte

- **Hipótese de Indução:**

  Sabemos ordenar um conjunto de $1 \leq k \leq n$
- **Caso Base:** n=1. OK!
- **Passo da Indução:** Seja S um conjunto com n>1 inteiros.

  Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

Indução Forte

- **Hipótese de Indução:**

  Sabemos ordenar um conjunto de $1 \leq k \leq n$
- **Caso Base:** n=1. OK!
- **Passo da Indução:** Seja S um conjunto com n>1 inteiros.

  Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

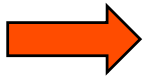
$< n$

Indução Forte

- **Hipótese de Indução:**

  Sabemos ordenar um conjunto de $1 \leq k \leq n$
- **Caso Base:** n=1. OK!
- **Passo da Indução:** Seja S um conjunto com n>1 inteiros.

  Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$



Podemos usar a h.i. para ordenar S_1 e S_2

Indução Forte

- **Hipótese de Indução:**

  Sabemos ordenar um conjunto de $1 \leq k \leq n$
- **Caso Base:** n=1. OK!
- **Passo da Indução:** Seja S um conjunto com n>1 inteiros.

  Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

⇒ Podemos usar a h.i. para ordenar S_1 e S_2

COMO OBTER S?

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:
  - SORT(i,j): ordena o subcojunto $\{s_i,..., s_j\}$

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:
  - SORT(i,j): ordena o subcojunto $\{s_i,..., s_j\}$
  - MERGE(S_1, S_2): efetua a intercalação dos subconjuntos ordenados S_1 e S_2

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:
  - SORT(i,j): ordena o subcojunto $\{s_i,..., s_j\}$
  - MERGE(S_1, S_2): efetua a intercalação dos subconjuntos ordenados S_1 e S_2
  - Chamada Externa: **SORT(1,n)**

Algoritmo

Procedimento SORT(i,j)

Se i=j **então**

retornar s_i

Senão

m:= (i+j –1)/2

Retornar MERGE(SORT(i, m), **SORT**(m+1, j))

Algoritmo

Procedimento MERGE(S_1, S_2)

Se $S_1 = \varnothing$ **então**
    Retornar S_2
senão
    se $S_2 = \varnothing$ **então**
        Retornar S_1
    senão
        sejam e_1, e_2 os elementos iniciais de S_1 e S_2 resp.
        k:= **Se** $e_1 > e_2$ **então** 1 **senão** 2
        retornar $\{e_k\}$
        $S_k := S_k - \{e_k\}$
        MERGE(S_1, S_2)

Complexidade

- $T(n)= 2T(n/2) + n-1$