Análise e Técnicas de Algoritmos

Jorge Figueiredo

Visão Geral do Curso

Agenda

- Introdução Informal
- Motivação

Introdução Informal

- O nosso curso é sobre técnicas e análise de algoritmos (computacionais).
- O que é um algoritmo?
  - Procedimento computacional que toma algum valor (conjunto) como entrada e produz um valor (conjunto) como saída.
- Qual a diferença entre algoritmos e programas?
  - Idéia x texto descritivo
  - humanos x computadores

Problemas Computacionais

Especifica a relação entre a entrada e a saída desejada

Ordenação
Entrada: Uma seqüência de n números $\langle a_1, a_2, ..., a_n \rangle$.
Saída: Uma reordenação da seqüência de entrada $\langle a'_1, a'_2, ..., a'_n \rangle$, onde $a'_1 \leq a'_2 \leq ... \leq a'_n$.

Número Primo
Entrada: Uma número natural q.
Saída: *sim* ou *não*, dependendo se q é primo.

Problemas Computacionais

- Um instância de um problema computacional é um possível valor para a entrada.
  - ‹45, 7, 13, 23, 2› é uma instância para o problema da ordenação.
  - 29 é uma instância para o problema dos números primos.

Como Descrever Algoritmos?

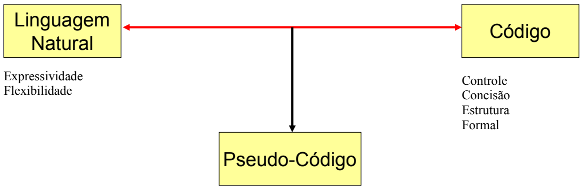
- Utilizar uma linguagem. Qual?
  - Linguagem natural (português ou inglês).
  - Linguagem de programação.
- Problemas:
  - Ambiguidade
  - Estruturação
  - Prolixidade
  - Inadequação para programação

Como descrever algoritmos?

Expressividade
Flexibilidade

Controle
Concisão
Estrutura
Formal

Como descrever algoritmos?

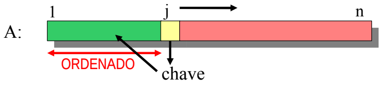


Expressividade
Flexibilidade

Controle
Concisão
Estrutura
Formal

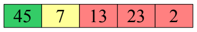
Reduzir ambiguidade
Abstrair detalhes não importantes
Fácil de ser assimilado por humanos

Insertion Sort



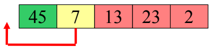
```
InsertionSort(A, n)
for j← 2 to n do
        chave ← A[j]
        ► insere A[j] na parte ordenada A[1..j-1]
        i ← j – 1
        while i > 0 e A[i] > chave do
                A[i + 1] ← A[i]
                i ← i – 1
        A[i + 1] ← chave
```

Exemplo do Insertion Sort



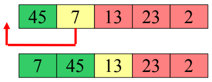
45	7	13	23	2

Exemplo do Insertion Sort

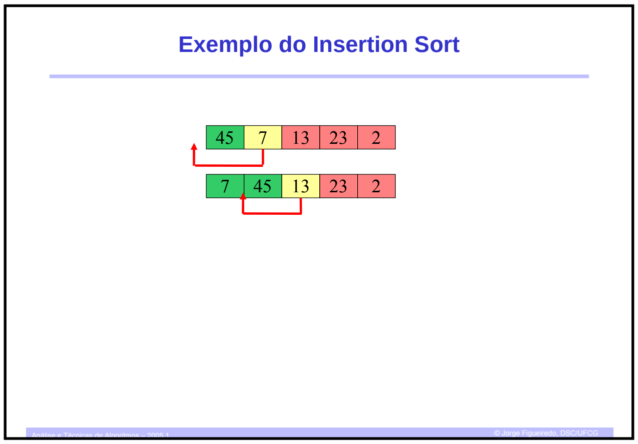


45	7	13	23	2

Exemplo do Insertion Sort



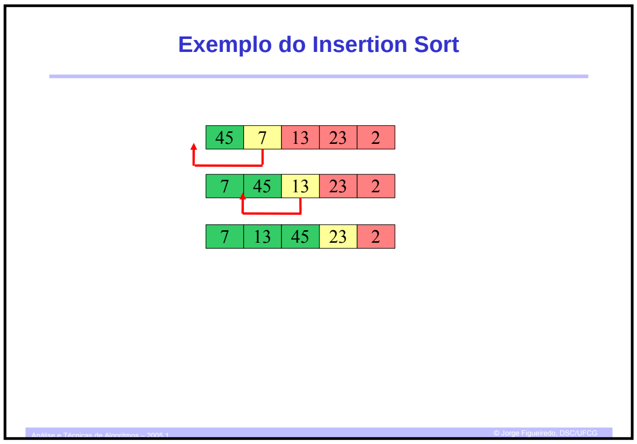
45	7	13	23	2
7	45	13	23	2



Exemplo do Insertion Sort

45 7 13 23 2

7 45 13 23 2

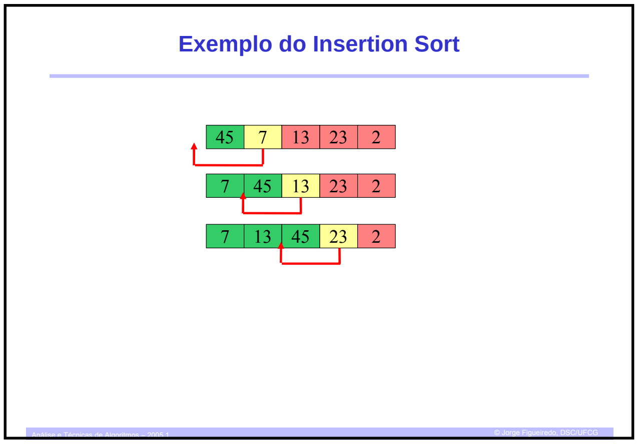


Exemplo do Insertion Sort

45 7 13 23 2

7 45 13 23 2

7 13 45 23 2

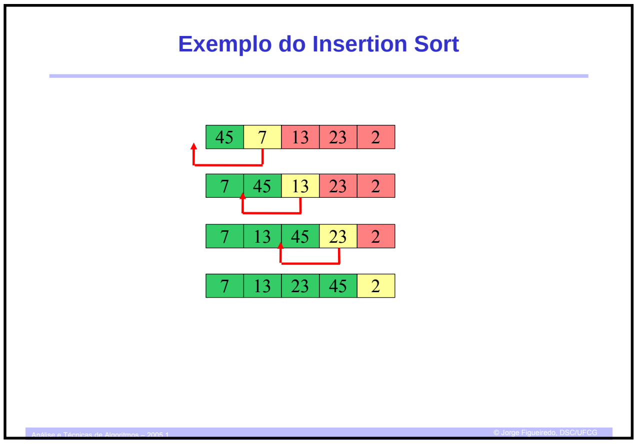


Exemplo do Insertion Sort

45 7 13 23 2

7 45 13 23 2

7 13 45 23 2



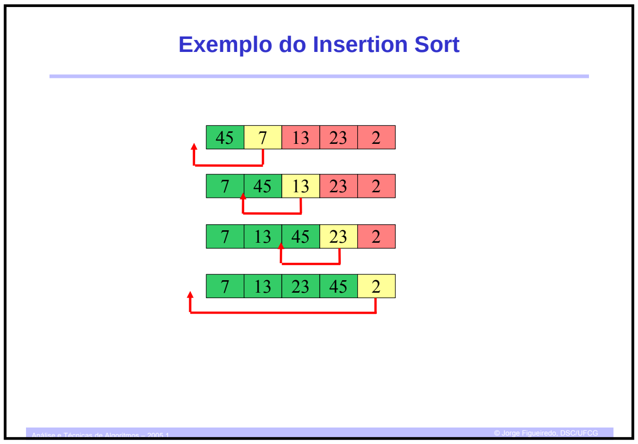
Exemplo do Insertion Sort

45 7 13 23 2

7 45 13 23 2

7 13 45 23 2

7 13 23 45 2



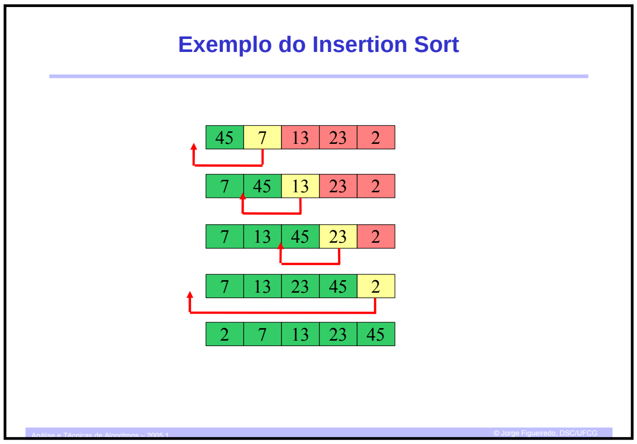
Exemplo do Insertion Sort

45 7 13 23 2

7 45 13 23 2

7 13 45 23 2

7 13 23 45 2



Exemplo do Insertion Sort

45 7 13 23 2

7 45 13 23 2

7 13 45 23 2

7 13 23 45 2

2 7 13 23 45

Avaliação de Algoritmos

- Propriedades:
  - Corretude
  - Simplicidade
  - Eficiência

Avaliação de Algoritmos

- Corretude:
  - Um algoritmo está correto se para toda entrada (legal) ele produz a saída correta.

Avaliação de Algoritmos

- Simplicidade:
  - Benefícios pragmáticos:
    - Fácil de ser entendido
    - Fácil de implementar
    - Fácil de manter

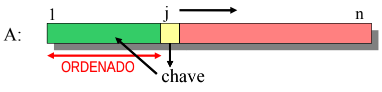
Avaliação de Algoritmos

- Eficiência (em função do tamanho do problema/entrada):
  - Tempo
    - Quanto tempo leva para produzir a saída correta?
  - Espaço
    - Quanto espaço de memória é necessário?

Como provar a corretude de um algoritmo?

- Não é uma tarefa simples
- Testes não são suficientes: servem apenas para mostrar que um algoritmo tem erros.
- Ferramentas utilizadas:
  - Invariantes de laço
  - Função recursiva
  - Prova por indução

Corretude do Insertion Sort



```
InsertionSort(A, n)
for j ← 2 to n do
    chave ← A[j]
    ► insere A[j] na parte ordenada A[1..j-1]
    i ← j – 1
    while i > 0 e A[i] > chave do
        A[i + 1] ← A[i]
        i ← i – 1
    A[i + 1] ← chave
```

Corretude do Insertion Sort

- Invariante de laço:
  No início de cada iteração do laço, o subarray A[1..*j*-1] contém os elementos originais de A[1..*j*-1] mas, ordenados.
- Prova:
  Inicialização : *j*=2, A[1..*j*-1]=A[1..1]=A[1], ordenado.
  Manutenção: cada iteração preserva o invariante.
  Término: *j*=*n*+1, então A[1..*j*-1]=A[1..*n*], ordenado.

Eficiência do Insertion Sort

- O algoritmo é eficiente?
- O tempo de execução de um algoritmo depende da *cara* da entrada.
- Na análise do tempo de execução, a parametrização é baseada no tamanho da entrada.
- A análise do tempo de execução, é uma forma de definir eficiência.
- *Random-Access-Model* (*RAM*)

Independência da Máquina

- Para comparar os diferentes algoritmos, de forma justa, é necessário definir um modelo abstrato de máquina.
- Máquina de Acesso Aleatório:
  - Um único processador genérico.
  - Instruções executadas sequencialmente, sem operações concorrentes.
  - Memória ilimitada.
  - Instrução básica toma uma unidade de tempo.

Tempo de Execução – Tipos de Análise

- Pior caso:
  - Maior tempo de execução de um algoritmo para qualquer entrada de tamanho n.
  - É o tipo mais utilizado. Todos gostam de garantia.
- Caso médio:
  - Tempo esperado de um algoritmo sobre todas as entradas de tamanho n.
  - Necessidade de usar distribuição estatística.
- Melhor caso:
  - Raramente é feita.

Eficiência do Insertion Sort

- Como determinar o tempo de execução?

```
InsertionSort(A, n)                              custo    vezes
for j ← 2 to n do                                 $c_1$    $n$
    chave ← A[j]                                  $c_2$    $n-1$
    //insere A[j] na parte ordenada A[1..j-1]     0        $n-1$
    i ← j-1                                       $c_4$    $n-1$
    while i >0 e A[i]>chave do                    $c_5$    $\sum_{j=2}^{n} t_j$
        A[i+1] ← A[i]                             $c_6$    $\sum_{j=2}^{n}(t_j-1)$
        i ← i-1                                   $c_7$    $\sum_{j=2}^{n}(t_j-1)$
    A[i+1] ← chave                                $c_8$    $n-1$
```

(t_j é o número de vezes que o teste do laço na linha 5 é executado para o valor de j)
O custo total é $T(n)$ = soma de *custo* × *vezes* em cada linha

$$=c_1n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$$

Eficiência do Insertion Sort (cont.)

- Melhor caso: números já ordenados
  - $tj=1$, e linhas 6 e 7 serão executadas 0 vezes
  - $T(n) = c1n + c2(n-1) + c4(n-1) + c5(n-1) + c8(n-1)$
    $=(c1 + c2 + c4 + c5 + c8)n - (c2 + c4 + c5 + c8) = cn + c'$
- Pior caso: números em ordem inversa
  - $tj=j$
  - $\sum_{j=2}^{n} tj = n(n+1)/2-1$ e $\sum_{j=2}^{n}(tj-1) = n(n-1)/2$,
  - $T(n) = c1n + c2(n-1) + c4(n-1) + c5(n(n+1)/2 -1) + c6(n(n-1)/2 -1) + c7(n(n-1)/2)+ c8(n-1)$
    $=((c5 + c6 + c7)/2)n2 +(c1 + c2 + c4 +c5/2-c6/2-c7/2+c8)n-(c2 + c4 + c5 + c8) = an^2+bn+c$

Eficiência do Insertion Sort (cont.)

- Caso médio: números aleatórios
  - Na média, $t_j = j/2$
  - $T(n)$ será da ordem de n^2, mesmo que no pior caso.

Ainda Sobre Análise de Algoritmos

- A análise de algoritmos permite o estudo teórico de programas computacionais:
  - Desempenho.
  - Utilização de recursos.
  - Corretude.
- Estudo de métodos, técnicas, idéias, dicas para desenvolver algoritmos (eficientes).

- Importância do estudo e análise de algoritmos:
  - Ajuda no entendimento de escalabilidade.
  - Permite definir o que é viável e o que é impossível.
  - A matemática utilizada serve como uma linguagem para lidar com o comportamento de um programa.
  - Provê meios para comparar diferentes soluções de um mesmo problema.

Objetivos do Curso

1. Aprender algoritmos clássicos
2. Aprender como identificar/desenvolver algoritmos corretos e eficientes para resolver um determinado
3. Aprender como expressar algoritmos
4. Aprender como validar/verificar algoritmos
5. Aprender como analisar
6. Aprender como reutilizar algoritmos
7. Aprender como aplicar algoritmos bastante conhecidos.

Razões para Estudar Algoritmos

- Evitar reinventar a roda:
  - Existem bons algoritmos que solucionam problemas importantes.
- Ajudar no desenvolvimento de seus algoritmos:
  - Nem sempre existe um algoritmo de prateleira que sirva para resolver o seu problema.
  - O conhecimento de algoritmos bem estabelecidos é fonte de inspiração.
  - Muitos dos princípios de projetos de algoritmos são úteis em todos os problemas de programação.

Razões para Estudar Algoritmos

- Ajudar a entender ferramentas que utilizam algoritmos particulares.
  - Por exemplo, ferramentas de compressão de dados.
- Útil conhecer as técnicas de algoritmos empregadas para resolver determinadas classes de problemas.