Projeto e Análise de Algoritmos

Algoritmos Iterativos: Projeto e Corretude

Atílio G. Luiz

Primeiro Semestre de 2024

Invariante de laço

Invariante de laço

- Correção de Insertion-Sort

Revendo o Insertion-Sort

```
Insertion-Sort(A, n)
  para j = 2 até n faça
      chave = A[j]
      i = j − 1
      enquanto i ≥ 1 e A[i] > chave faça
          A[i + 1] = A[i]
          i = i − 1
      A[i + 1] = chave
```

Até agora:

- já vimos que o algoritmo termina
- e analisamos sua complexidade de tempo

O que falta fazer?

- Verificar se ele está correto.

Invariante de laço

Definição

Uma invariante laço é uma propriedade de laço que

- expressa uma relação entre as variáveis
- está associada a determinada posição de um laço
- é satisfeita (verdadeira) em toda execução do laço

A posição escolhida é normalmente descrita como

- imediatamente antes ou depois da iteração do laço
- imediatamente antes ou depois de determinada linha

Objetivos

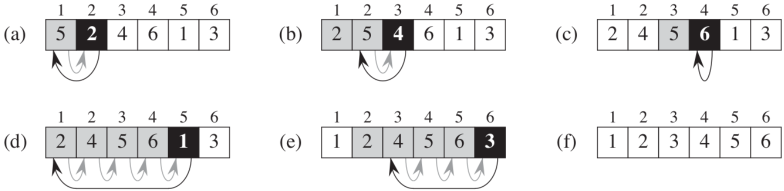
- após o término do laço, deve ser uma propriedade útil para se mostrar a correção do algoritmo
- permite nos concentrar apenas em uma iteração do laço

Exemplo de invariante

```
Insertion-Sort(A, n)
    para j = 2 até n faça
        chave = A[j]
        i = j − 1
        enquanto i ≥ 1 e A[i] > chave faça
            A[i + 1] = A[i]
            i = i − 1
        A[i + 1] = chave
```

- Qual seria um bom invariante para o laço **para**?

Snapshots de um array sob ação do INSERTION-SORT



Em cada j-ésima iteração, o elemento $A[j]$ é guardado em uma variável e comparado com os elementos à sua esquerda (que já estão ordenados) até encontrar a sua posição correta.

Exemplo de invariante

```
Insertion-Sort(A, n)
    para j = 2 até n faça
        chave = A[j]
        i = j − 1
        enquanto i ≥ 1 e A[i] > chave faça
            A[i + 1] = A[i]
            i = i − 1
        A[i + 1] = chave
```

Invariante

Imediatamente antes de cada iteração do laço **para**, o subvetor $A[1 \ldots j-1]$ consiste dos elementos originalmente em $A[1 \ldots j-1]$, ordenados em ordem crescente.

- posição da invariante: antes da iteração do laço **para**
- propriedade invariante: $A[1 \ldots j-1]$ está ordenado

Demonstrando uma invariante de laço

Tipicamente, demonstramos uma invariante com as etapas:

1. mostre que a propriedade vale antes de qualquer iteração
2. mostre que, se a propriedade vale no início da j-ésima iteração, então ela também vale no início da $(j+1)$-ésima iteração
3. conclua que a invariante vale quando o laço termina

Estamos usando o princípio da indução!

- a base corresponde à etapa 1
- o passo indutivo corresponde à etapa 2

Demonstrando uma invariante: caso base

Considere a primeira iteração do laço **para** do INSERTION-SORT

- no início da iteração, temos $j = 2$
- neste caso, o subvetor $A[1 \dots j-1]$ contém apenas um elemento
- A[1] é o primeiro elemento do vetor original e está ordenado
- então, a invariante vale antes da primeira iteração

Demonstrando uma invariante: passo indutivo

Suponha que a invariante vale no início de alguma iteração j

- nessa iteração, temos $chave = A[j]$
- informalmente, o corpo do laço **enquanto** desloca os valores $A[j-1], A[j-2], A[j-3]$, e assim por diante, uma posição para a direita, até encontrar a posição correta para *chave*, que é a posição $i+1$.
- Os valores em $A[1 \dots i]$ são todos menores ou iguais que *chave* e os valores em $A[i+2 \dots j]$ são todos maiores que a *chave* e ambos subarrays estão ordenados
- Após o laço **enquanto**, $A[i+1]$ recebe o valor de *chave*
- Assim, o subarray $A[1 \dots j]$ consiste dos elementos originalmente em $A[1 \dots j]$, mas em ordem crescente.
- Incrementando j para a próxima iteração do laço **para** preserva a invariante de laço

Demonstrando uma invariante: término

Finalmente, examinamos o que acontece quando o laço termina

- o laço **para** termina quando $j > n$
- como cada iteração incrementa j em 1, então $j = n + 1$ na última iteração
- Substituindo $n + 1$ por j no enunciado da invariante de laço, temos que: o subarray $A[1 \dots n]$ consiste dos elementos originalmente em $A[1 \dots n]$, em ordem crescente.
- Assim, o array inteiro está ordenado e o algoritmo é correto.

Projeto de Algoritmos Iterativos

Motivação

- Algoritmo Iterativo
  - Resolve o problema em vários passos (usa repetição)
  - Cada passo deixa mais próximo da solução
- Provar corretude depois de projetar pode ser difícil
- Mais fácil projetar e provar corretude simultaneamente
- Estruturar a tarefa em pequenos passos
  - Se os passos forem cumpridos, teremos um projeto e uma prova de corretude.

Sequências de ações × Sequências de assertivas

- Algoritmos podem ser vistos como:
  - Sequências de ações
  - Sequências de fotografias do estado do computador
- Visão dupla melhora a compreensão
  - Fácil de se perder em if's e while's
- Expressamos estados com assertivas
  - O que deve ser verdadeiro em cada ponto
  - Pré-condições e pós-condições
  - Estados intermediários
  - Geral o bastante para facilitar o entendimento
- Ação garante assertiva, base na anterior
  - $<$assertiva$_i >$ & código$_i \Longrightarrow <$assertiva$_{i+1} >$
  - Corretude: provar a pós-condição

Sequências de ações × Sequências de assertivas

- As assertivas geralmente são comentários
  - Pode intercalar com português (cuidado com ambiguidades)
  - Podemos implementá-las para usar como ferramenta de depuração
  - Todas as suposições feitas devem estar explícitas nas assertivas
- Estruturação baseada nos dados

Passos para projetar um algoritmo iterativo

Estrutura

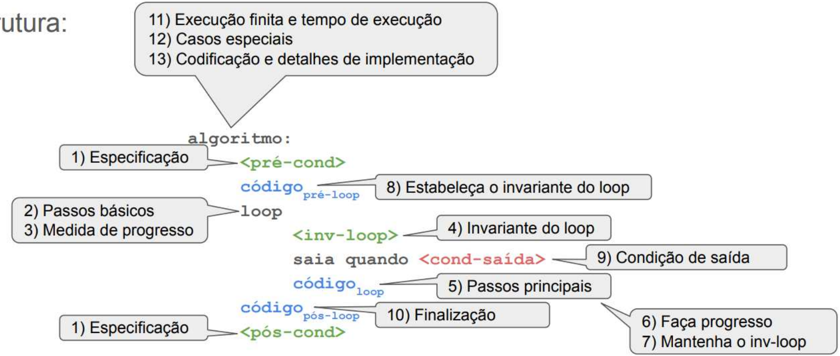
```
algoritmo:
    <pré-cond>
    código pré-loop
    loop
        <inv-loop>
        saia quando <cond-saída>
        código loop
    endloop
    código pós-loop
    <pós-cond>
```

```
algoritmo:
    <pré-cond>
    código pré-loop
    while not <cond-saída>


        código loop
    endwhile
    código pós-loop
    <pós-cond>
```

Passos para projetar um algoritmo iterativo

- Estrutura:



Passos 1-11 fornecem prova formal de corretude!

Passos para projetar um algoritmo iterativo

1. **Especificação**
   - Definição precisa do que deve ser resolvido
   - Pré-condições: tudo que é assumido verdadeiro sobre a entrada
   - Pós-condições: tudo que deve ser satisfeito pela saída
   - **Ex:** (Find-Max) Posição do maior elemento em uma lista
     - Pré-condições: a entrada é uma lista `L[1..n]` com n números
     - Pós-condições: a saída é um índice m tal que `L[m]` tem valor máximo. Em caso de empate, qualquer dos índices pode ser retornado.
   - Corretude depende da especificação:
     `<pre-cond> & código => <pos-cond>`
   - Contrato entre o implementador e o usuário
     - Implementador assume pré-condições e garante pós-condições
     - Usuário assume pós-condições sempre que fornecer entradas válidas (pré-condições)

Passos para projetar um algoritmo iterativo

2. **Passos Básicos**
   - Projeto preliminar indicando em linhas gerais como cada iteração avança para a solução
   - Teste algumas iterações em instâncias simples
   - Ex. (Find-Max): Dois índices: i e m. O índice i percorre a lista (um elemento por iteração), e o índice m guarda a posição do maior (qualquer deles) encontrado até então.

Passos para projetar um algoritmo iterativo

3. **Medida de progresso**
    - Função que, dado o estado atual, fornece quanto progresso foi feito ou quanto ainda falta
    - Essa função deve ter valores inteiros
    - Algoritmo deve terminar: não pode ser infinito, e cada iteração deve gerar progresso
    - Ex.: quantidade de saídas produzidas, quantidade de entradas consideradas, tamanho do espaço de busca, etc
    - Ex. (Find-Max): Quantidade de elementos considerados até então (percorridos por i)

Passos para projetar um algoritmo iterativo

4. **Invariante de laço**
   - Assertiva colocada no início do laço, e deve ser verdadeira em todas as iterações
   - Parte mais difícil (criatividade), mas restante geralmente decorre facilmente
   - Descrição deveria dar uma imagem visual do estado das EDs
   - Deve garantir que a computação se mantém em direção à solução
   - Significativa: quando combinada com a condição de saída e com o código pós-loop, deve garantir a pós-condição
   - Alcançável: deve ser capaz de estabelecê-la e mantê-la
   - O que gostaria que fosse verdadeiro no meio da computação? É razoável?
     - Imagine uma iteração onde o invariante é satisfeito no início do laço
     - É possível fazer progresso na iteração mantendo o invariante?
       - Se é fraco demais, você não tem o que precisa para fazer o progresso
       - Se é forte demais, você avança mas não consegue manter o invariante
   - Ex.: (Find-Max): m tem posição do maior (qualquer deles) dentre os considerado até então

Passos para projetar um algoritmo iterativo

5. Passos principais (código do laço)

- Suponha que está em uma iteração intermediária (não necessariamente a primeira)
- Quais passos devem ser feitos em uma única iteração?
  - Necessário fazer progresso e manter o invariante
- Ex. (Find-Max):
  Avance o índice i. Se $L(i) > L(m)$, copie o valor de i para m.

Passos para projetar um algoritmo iterativo

6. **Faça progresso**
   - Mostre que cada iteração avança em pelo menos uma unidade a medida de progresso
   - Pode ser necessário reforçar a medida de progresso ou corrigir o código do loop
   - Ex. (Find-Max): Cada iteração considera um novo elemento (avanço do índice i)

Passos para projetar um algoritmo iterativo

7. Mantenha a invariante do laço

```
algoritmo:
    <pré-cond>
    código pré-loop
    loop
        <inv-loop>
        saia quando <cond-saída>
        código loop
    endloop
    código pós-loop
    <pós-cond>
```

Passos para projetar um algoritmo iterativo

7. **Mantenha a invariante do laço**
   - Prove que o invariante do laço é mantido a cada iteração
     `<inv-loop'>` & `not` `<cond-saida>` & `código-loop` ⇒ `<inv-loop''>`
   - Técnica de prova:
     - Suponha que o laço começou a executar
     - Suponha que o invariante de laço é satisfeito
     - Suponha que a condição de saída não é satisfeita
     - Execute uma iteração do código do laço. Como isso alterou as EDs?
     - Mostre que as alterações realizadas conservam a invariante de laço
   - Usamos ' e " para diferenciar o estado antes e depois da execução
     - Ex.: matematicamente, a atribuição $x = x + 2$ pode ser expressa com $x'' = x' + 2$

7. Mantenha a invariante do laço

- Ex.: (Find-Max):
  - Suponha início da iteração. Início: $i = i'$ e $m = m'$.
    Final: $i = i''$ e $m = m''$
  - Condição de saída é falsa: existe elemento ainda não considerado.

$$\underbrace{\overbrace{L(1) \qquad \cdots \qquad}^{L(m')\ \text{é o maior}} \ \overbrace{L(i'')}^{\text{iteração}}}_{L(m'')=\max(L(m'),L(i''))}$$

  - Pelo código do laço:
    - $m'' = i''$ se $L[i''] > L[m']$
    - $m'' = m'$ caso contrário
    - Ou seja, $L[m''] = \max\{L[m'], L[i'']\}$

Passos para projetar um algoritmo iterativo

8. Estabeleça o invariante do laço

```
algoritmo:
    <pré-cond>
    código pré-loop
    loop
        <inv-loop>
        saia quando <cond-saída>
        código loop
    endloop
    código pós-loop
    <pós-cond>
```

Passos para projetar um algoritmo iterativo

8. **Estabeleça o invariante do laço**

   - Prove que o invariante do laço vale na 1a iteração
     `<pre-cond>` $\&$ `código-pré-loop` $\Rightarrow$ `<inv-loop>`
   - Técnica de prova
     - Suponha que está no início da execução
     - Suponha que a entrada satisfaz as pré-condições
     - Execute o código pré-loop
     - Mostre que o invariante do laço está satisfeito
   - Ex.: (Find-Max):
     - No código pré-loop fazemos $i = m = 1$.
     - Como apenas o 1o elemento foi considerado, m é o índice para o maior

Passos para projetar um algoritmo iterativo

9. **Condição de saída**
   - Expressa cumprimento da tarefa do laço
   - Será usado na prova da `<pos-cond>`
   - Ex.: (Find-Max):
     O índice i já percorreu por todos os elementos de L?

```
algoritmo:
    <pré-cond>
    código pré-loop
    loop
        <inv-loop>
        saia quando <cond-saída>
        código loop
    código pós-loop
    <pós-cond>
```

Passos para projetar um algoritmo iterativo

10. **Finalização**

- Mostre que após encerrar o laço seremos capazes de resolver o problema
  `<inv-loop> & <cond-saída> & código pós-loop ⇒ <pós-cond>`

- Técnica de prova
  - Suponha que acabou de sair do laço
  - Pode supor que inv-loop é satisfeito, pois vale no início de toda iteração, e o teste de saída é a 1a ação da iteração
  - Suponha que cond-saída é satisfeita, pois acabou de sair do laço
  - Execute o código pós-loop
  - Mostre que a pós-condição é satisfeita

- Ex.: (Find-Max):
  - inv-loop: m contém a posição do maior dentre os considerados
  - cond-saída: todos os elementos foram considerados
  - Concluímos que m contém a posição do maior em L
  - Então, para satisfazer pós-cond basta retornar m no código pós-loop

Passos para projetar um algoritmo iterativo

11. Execução finita e tempo de execução

- Mostre que o algoritmo não fica em loop infinito
  - Prove que o laço terá encerrado quando a medida de progresso atingir um determinado valor finito
- O tempo de execução: tempo código pré-loop + tempo código pós-loop + soma dos tempos de código do loop para cada iteração
  - Expressar em notação O ou Θ
- Ex.: (Find-Max):
  - Número de iterações é o tamanho da lista n (finito)
  - código pré-loop, código loop e código pós-loop são $\Theta(1)$
  - Portanto, o algoritmo é $\Theta(n)$

Passos para projetar um algoritmo iterativo

12. **Casos especiais**
    - Comece projetando para um caso geral, e depois acrescente casos particulares
    - Verifique os casos que já são atendidos pelo algoritmo
    - Implemente os casos não cobertos, verificando se os casos anteriores ainda são atendidos
    - Ex.: (Find-Max):
        - Teste entradas com valores repetidos, e com tamanho $n = 0$ e $n = 1$.

Passos para projetar um algoritmo iterativo

13. Codificação e detalhes de implementação

- Forneça o pseudocódigo e detalhes de implementação
- Detalhes de implementação podem ser ocultados por tipos abstratos de dados
- Deixe em aberto: detalhes que não fazem diferença (flexibilidade)
- Ex.: (Find-Max):

```
algoritmo Find-Max(L):
    <pré-cond>: L[1..n] é um array com n números, n > 0.
    <pós-cond>: retorna um índice m t.q. L[m] é máximo
    i = 1; m = 1
    loop
        <inv-loop>: L[m] é máximo em L[1..i].
        saia quando i >= n
        i = i + 1
        se L[i] > L[m] então m = i
    endloop
    return m
```

Passos para projetar um algoritmo iterativo

14. Prova formal

- Os passos 1-11 são suficientes para garantir que o algoritmo funciona para toda entrada válida
- Vimos no passo 8 que inv-loop é satisfeito na primeira iteração
  `<pre-cond>` & `código pré-loop` ⇒ `<inv-loop>`
- Mostramos que inv-loop é mantido em todas as iterações (por indução)
  - Caso base: pelo passo 8, inv-loop vale na 1a iteração
    `<pre-loop>` & `código pré-loop` ⇒ `<inv-loop>`
  - Passo indutivo: pelo passo 7, inv-loop é mantido após a execução da iteração
    `<inv-loop'>` & `not <cond-saída>` & `código loop` ⇒ `<inv-loop''>`
- Por fim, vimos nos passo 10 que pós-cond é satisfeita se inv-loop vale na saída do loop
  `<inv-loop>` & `<cond-saida>` & `código pós-loop` ⇒ `<pos-cond>`

Tipos de algoritmos iterativos

Tipos de algoritmos iterativos

- Mais da saída
  - Medida de progresso: quantidade da saída construída
  - Invariante do loop: a saída construída até então está correta
- Mais da entrada
  - Medida de progresso: quantidade da entrada considerada
  - Invariante do loop: se a entrada já considerada fosse completa, teríamos solução completa
- Estreitando o espaço de busca
  - Medida de progresso: tamanho do espaço no qual a busca foi restringida
  - Invariante do loop: se o objeto buscado está na entrada, então está no espaço restringido
- Trabalho realizado
  - Medida de progresso: alguma forma criativa de medir o trabalho realizado
  - Ex. (bubble sort): número de pares de elementos ainda fora de ordem

Tipos de algoritmos iterativos

- Mais da saída

Ex. (Mais da saída): Ordenação por seleção

1. Especificação: rearranjar lista com n valores em ordem não decrescente
2. Passos básicos: repetidamente selecione menor dentre os não selecionados, e coloque no final da lista de selecionados
3. Medida de progresso: número k de elementos já selecionados
4. Invariante do loop: os k selecionados são os k menores, e estão em ordem
5. Passos principais: encontre o menor dentre os não selecionados, e mova para a última posição dos selecionados
6. Faça progresso: sim, pois o k aumenta
7. Mantenha o invariante:
    - Pelo invariante anterior, o selecionado não é menor que os selecionados anteriormente
    - Pelo código do loop, o selecionado não é maior que nenhum dentre os não selecionados
    - Então ele pode entrar na posição $k+1$ da lista de selecionados

Ex. (Mais da saída): Ordenação por seleção

8. Estabeleça o invariante: inicialmente $k = 0$ (nenhum foi selecionado)
9. Condição de saída: $k = n$
10. Finalização:
    - Pela condição de saída todos já foram selecionados
    - Pelo invariante os selecionados estão em ordem
    - Então basta retornar a lista de selecionados
11. Execução finita e tempo de execução:
    - Depende da estratégia para localizar o menor elemento

Tipos de algoritmos iterativos

- Mais da entrada

Ex. (mais da entrada): Ordenação por inserção

1. Especificação: rearranjar lista com n valores em ordem não decrescente
2. Passos básicos:
   - Repetidamente leia próxima entrada e coloque em posição que mantenha os lidos em ordem
3. Medida de progresso: número k de elementos já lidos
4. Invariante do loop: os k lidos estão em ordem
5. Passos principais:
   - Leia próxima entrada e coloque em posição que mantenha os lidos em ordem
6. Faça progresso: sim, pois o k aumenta
7. Mantenha o invariante:
   - Código do loop posiciona novo elemento de modo a manter os lidos em ordem (invariante)

Ex. (mais da entrada): Ordenação por inserção

8. Estabeleça o invariante: inicialmente $k = 1$ (array com 1 elemento já está ordenado)
9. Condição de saída: $k = n$ (todos foram lidos)
10. Finalização:
    - Pela condição de saída todos já foram lidos
    - Pelo invariante os lidos estão em ordem
    - Então basta retornar a lista de elementos lidos
11. Execução finita e tempo de execução:
    - Depende da estrutura de dados que mantém os elementos lidos

Tipos de algoritmos iterativos

- Estreitando o espaço de busca

Ex. (estreitando o espaço de busca): Busca binária

1. Especificação:
   - Entrada: Lista ordenada $A[1..n]$ e chave de busca. Elementos podem ser repetidos.
   - Saída: Índice k tal que $A[k] =$ chave, se a chave está na lista. Mensagem, caso contrário.
2. Passos básicos:
   - Divida o espaço de busca ao meio, e continue a busca na parte que contém a chave
3. Invariante do loop:
   - Se chave está na entrada, então ocorre em pelo menos um elemento de $A[i..j]$
     - Caso a chave seja repetida, pode ocorrer também fora de $A[i..j]$
4. Medida de progresso:
   - Número de elementos em $A[i..j]$, ou seja, $j - i + 1$

Ex. (estreitando o espaço de busca): Busca binária

5. Passos principais:
   - Encontre elemento do meio (posição $\lfloor (i+j)/2 \rfloor$)
   - Se $chave \leq A[meio]$, faça $j = meio$ (continue a busca em $A[i..meio]$)
   - Se $chave > A[meio]$, faça $i = meio + 1$ (continue a busca em $A[meio+1..j]$)
6. Condição de saída: quando $j - i + 1 \leq 1$ (0 ou 1 no espaço de busca).
7. Faça progresso:
   - $j - i + 1$ diminui (j reduz ou i aumenta se não for condição de saída)
8. Mantenha o invariante:
   - Como A está ordenado, se chave está em $A[i..j]$,
     - estará em $A[i..meio]$ quando $chave \leq A[meio]$, ou
     - estará em $A[meio+1..j]$ quando $chave > A[meio]$
9. Estabelecendo o invariante: faça $i = 1$ e $j = n$ (lista inteira).

Ex. (estreitando o espaço de busca): Busca binária

10. Finalização:
    - Invariante diz que chave estará em $A[i..j]$, se estiver na entrada
    - Condição de saída diz $A[i..j]$ tem zero ou um elemento
        - Se tem zero, concluímos pelo invariante que chave não está na entrada
        - Se tem um, resta apenas testar se é igual a chave
11. Execução finita e tempo de execução
    - Como cada iteração reduz aprox. à metade o intervalo $[i..j]$, número de iterações $\Theta(\log n)$.
    - Cada iteração é $\Theta(1)$, e também código pré e pós loop.
    - Então, o algoritmo é $\Theta(\log n)$.
12. Casos especiais:
    - Se chave não está na lista, iremos alcançar uma sublista vazia
13. Codificação e detalhes de implementação:
    - Podemos incluir o teste $chave = A[meio]$, reduzindo o número de iterações
        - Na prática deixa o algoritmo mais lento

Tipos de algoritmos iterativos

- Trabalho realizado

Ex. (trabalho realizado): Bubble sort

1. Especificação: rearranjar lista com n valores em ordem não decrescente
2. Passos básicos:
   - Inverter pares de elementos consecutivos que estão fora de ordem
3. Medida de progresso:
   - Involução: par de elementos fora de ordem $(1 \le i < j \le n, A[i] > A[j])$
   - Medida: número de involuções. Ex.: $[1,2,5,4,3,6]$ tem 3 involuções.
4. Invariante do loop: antes da k-ésima iteração, temos uma permutação dos elementos da entrada e os $(k-1)$-maiores elementos encontram-se ao final da sequência na sua posição correta.
5. Passos principais:
   - Passar pelos $n-k+1$ primeiros elementos da lista invertendo os pares de elementos consecutivos fora de ordem.

Ex. (trabalho realizado): Bubble sort

6. Faça progresso: se ouver alguma involução, ela será eliminada
7. Mantendo o invariante: uma inversão mantêm os elementos originais e, além disso, garantimos que colocamos o maior elemento entre os $n-k+1$ primeiros elementos depois de todos os demais.
8. Estabelecendo o invariante: lista de entrada já é uma permutação e nenum elemento foi considerado.
9. Condição de saída: nenhuma involução restante (lista está ordenada)
10. Finalização:
    - Invariante: lista é uma permutação dos elementos de entrada
    - Condição de saída: lista está ordenada
    - Basta retornar a lista

Ex. (trabalho realizado): Bubble sort

11. Execução finita e tempo de execução
    - Máximo de involuções (ordem inversa): $n(n-1)/2 \in \Theta(n^2)$
    - Cada k-ésima iteração coloca o k-ésimo maior elemento na sua posição final, resultando um total máximo de $O(n-k)$ involuções.
    - Como temos que checar n elementos, logo, temos um algoritmo $O(n^2)$