Introdução à Análise de Algoritmos

Marcelo Keese Albertini
Faculdade de Computação
Universidade Federal de Uberlândia

Aula de hoje

Nesta aula veremos:

- Sobre a disciplina
- Exemplo: ordenação

Sobre a disciplina

- Livros
  - Introduction to Algorithms (CLRS)
    - Pseudo-código e teoria de complexidade: limitantes assintóticos
  - Algorithms (Sedgewick e Wayne)
    - Mais didático e códigos
  - An Introduction to the Analysis of Algorithms (Sedgewick e Frajolet)
    - Análise: resolução de recorrências e casos médios

Sobre a disciplina: conteúdo

- Capítulos do CLRS: 1-8, 10-13, 15-18, 21-25, 34, 35
  - Análise: empírica, matemática, probabilística, amortizada e assintótica
  - Técnicas de projeto: divisão e conquista, programação dinâmica e algoritmos gulosos
  - Ordenação: merge sort, quick sort, count sort
  - Estruturas de dados: árvores balanceadas, heaps e tabelas hash
  - Grafos: estruturas básicas e algoritmos diversos
  - Problemas NP-completos: algoritmos de aproximação

Sobre a disciplina: motivações e objetivos

- Motivações
  - Corretude
  - Estimação de custos
  - Comparações de algoritmos
- Objetivo
  - Desenvolver habilidades de compreensão, projeto e análise de algoritmos

Sobre a disciplina: recursos e datas

- http://www.facom.ufu.br/~albertini/ada
- Avaliações: 3 provas (25+25+30) e 2 trabalhos (10+10)
- Datas: a combinar

Definição de um problema

Exemplo: problema de ordenação crescente

- Entrada: uma sequência de n números $a_1, a_2, a_3, \ldots, a_n$
- Saída: uma reordenação $b_1, b_2, \ldots, b_n$, tal que $b_i < b_j$ se $i < j$

A solução: um algoritmo

- correto: se para cada instância de entrada, o algoritmo para com a saída correta
- se o é algoritmo é correto então ele resolve o problema

Instância e tamanho da entrada de um problema

- Uma instância é uma materialização do problema: $3, 1, 2, 4, 1, 5$
- Tamanho da entrada N varia de acordo com a instância

Análise de algoritmos

Custos

- Quanto espaço de memória o algoritmo vai consumir?
- Quanto tempo levar o algoritmo?
- Quantos acessos a disco o algoritmo fará?
- Qual é o consumo de energia?
- Quantas requisições serão feitas a um banco de dados?

Como responder essas perguntas de acordo com a variável do tamanho do problema N ?

Análise do Problema de ordenação

- Como ordenar cartas de baralho?
- Como ordenar depósitos em um banco?
- Como estimar a duração?
- Como extrapolar o resultado?
- Qual é o modelo de aumento de custo com o aumento de entradas?
- Qual tipo de instância maximiza custos para um dado N?

Revisão para implementar ordenação

Conceitos

- Vetor: **int vetor[] = {5, 1, 7, 3, 0};**

Revisão para implementar ordenação

Conceitos

- Vetor: **int vetor[] = {5, 1, 7, 3, 0};**
- Variável índice: posição para acesso de elemento

Revisão para implementar ordenação

Conceitos

- Vetor: **int vetor[] = {5, 1, 7, 3, 0};**
- Variável índice: posição para acesso de elemento
- Variável auxiliar: armazenamento temporário

Revisão para implementar ordenação

Conceitos

- Vetor: **int vetor[] = {5, 1, 7, 3, 0};**
- Variável índice: posição para acesso de elemento
- Variável auxiliar: armazenamento temporário
  - útil para troca de posição de elementos do vetor

Ordenação interna

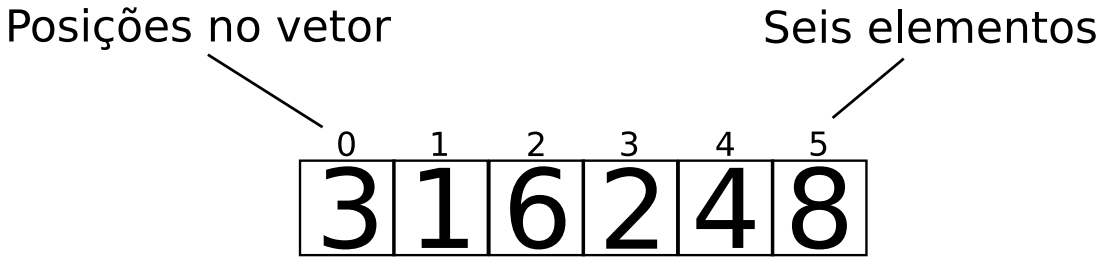
ordenar em memória

- Pré-condições: vetor em memória principal, inicializado com elementos

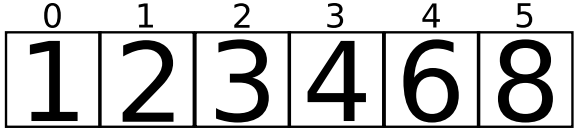
Ordenação interna

ordenar em memória

- Pré-condições: vetor em memória principal, inicializado com elementos
- Pós-condições: vetor com elementos em ordem crescente (ou decrescente)



Ordenado:



Vetores e Ordenação: exemplos

Exemplos:

- Números inteiros ou ponto flutuante

Vetores e Ordenação: exemplos

Exemplos:

- Números inteiros ou ponto flutuante
- Vetor de strings

Vetores e Ordenação: exemplos

Exemplos:

- Números inteiros ou ponto flutuante
- Vetor de strings
- Tipos compostos: necessário definir função para comparar

Vetores e Ordenação: exemplos

Exemplos:

- Números inteiros ou ponto flutuante
- Vetor de strings
- Tipos compostos: necessário definir função para comparar
  - **int compare(ITEM item1, ITEM item2);**

Exemplo

```
1 int compare(Aluno a) {
2     if (this.media > a.media)
3         return 1;
4     else if (this.media == a.media)
5         return -1;
6     else
7         return 0;
8 }
```

Algoritmo de ordenação

Definição de ordenação

Sequência de comparações e trocas de posição entre elementos para obter vetor ordenado.

Algoritmo de ordenação

Definição de ordenação

Sequência de comparações e trocas de posição entre elementos para obter vetor ordenado.

Complexidade

Quantas trocas, comparações (complexidade de tempo) e variáveis auxiliares (de espaço) são necessárias?

Bubble sort - ordenação em "bolhas"

Como programar um algoritmo de ordenação simples?

Ideia

Comparar pares consecutivos de elementos e trocá-los de posição caso o primeiro seja maior que o segundo.

```
1 void troca(int vetor[], int i, int j) {
2     int aux = vetor[i];
3     vetor[i] = vetor[j];
4     vetor[j] = aux;
5 }
```

Primeira iteração

Início da iteração

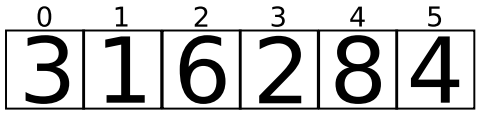
0	1	2	3	4	5
3	1	6	2	8	4

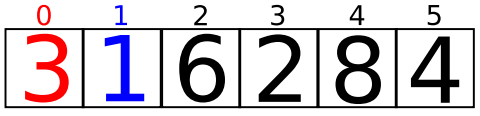
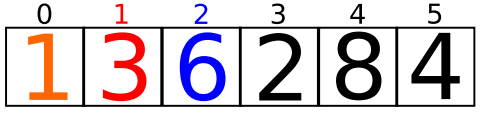
0	1	2	3	4	5
3	1	6	2	8	4

vetor[i]	vetor[i+1]	troca?
3	1	sim

Primeira iteração

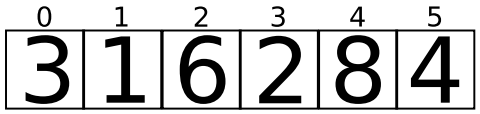
Início da iteração

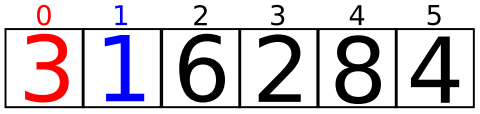
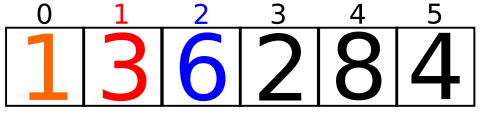
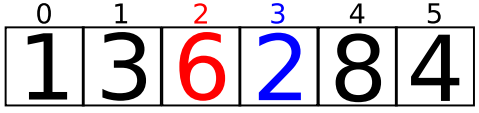


	vetor[i]	vetor[i+1]	troca?
	3	1	sim
	3	6	não

Primeira iteração

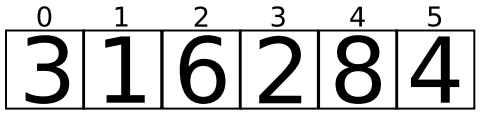
Início da iteração

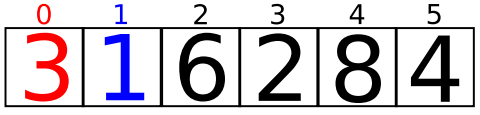
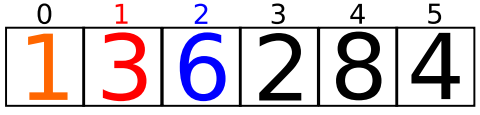
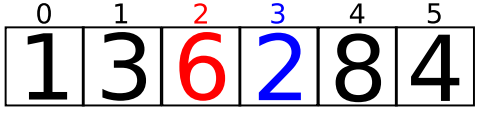
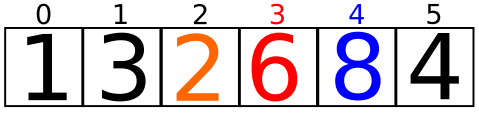


	vetor[i]	vetor[i+1]	troca?
	3	1	sim
	3	6	não
	6	2	sim

Primeira iteração

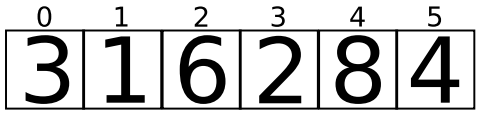
Início da iteração

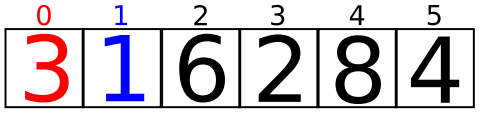
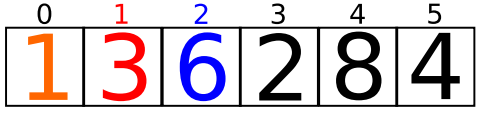
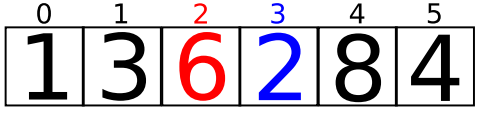
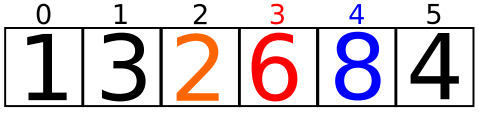
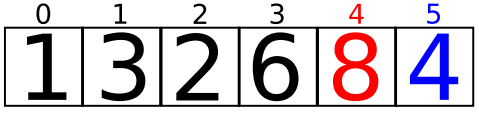


	vetor[i]	vetor[i+1]	troca?
	3	1	sim
	3	6	não
	6	2	sim
	6	8	não

Primeira iteração

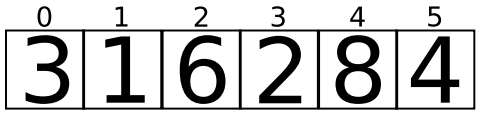
Início da iteração

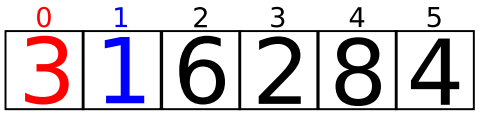
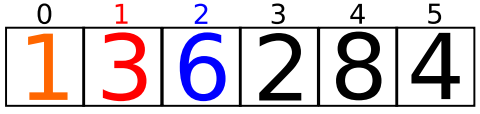
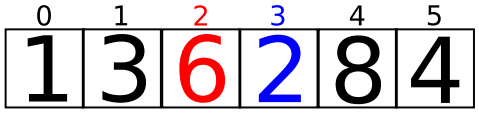
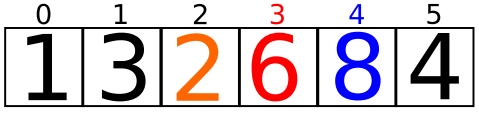
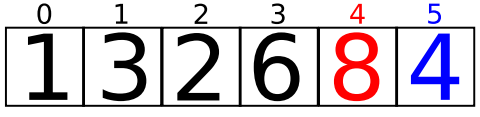


	vetor[i] vetor[i+1]	troca?
	3 1	sim
	3 6	não
	6 2	sim
	6 8	não
	8 4	sim

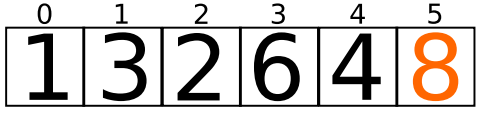
Primeira iteração

Início da iteração



	vetor[i]	vetor[i+1]	troca?
	3	1	sim
	3	6	não
	6	2	sim
	6	8	não
	8	4	sim

Fim da iteração



Maior elemento sempre está na sua posição ordenada na primeira iteração.

Algoritmo Bubblesort: versão simplificada

```
1 void bubblesort (int vetor[]) {
```

Algoritmo Bubblesort: versão simplificada

```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
```

Algoritmo Bubblesort: versão simplificada

```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
5         /*repeticao interna: compara e troca números */
6         for (int i = 0; i < N-1; i++)
```

Algoritmo Bubblesort: versão simplificada

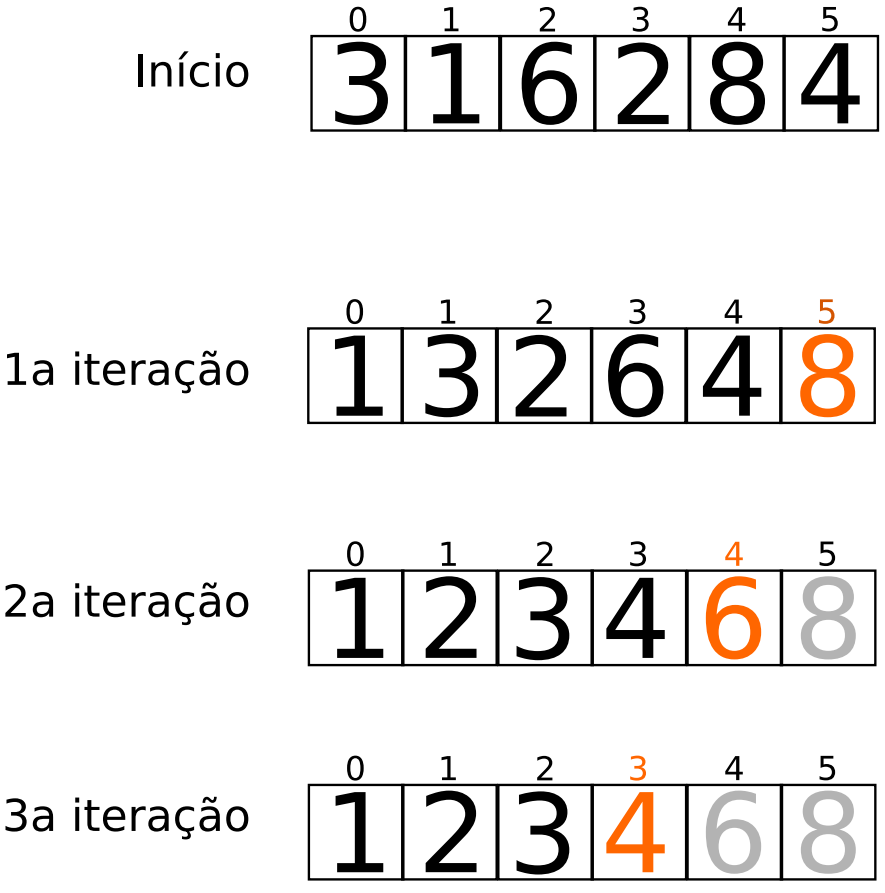
```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
5         /*repeticao interna: compara e troca números */
6         for (int i = 0; i < N-1; i++)
7             if (vetor[i] > vetor[i+1])
```

Algoritmo Bubblesort: versão simplificada

```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
5         /*repeticao interna: compara e troca números */
6         for (int i = 0; i < N-1; i++)
7             if (vetor[i] > vetor[i+1])
8                 troca(vetor, i, i+1); /*devemos trocar */
9 }
```

Termina com vetor ordenado de modo crescente.

Outras iterações



Exemplo sobre complexidade de ordenação

Problema

Um banco tem 10 milhões clientes que fazem, em média, 4 transações diárias. O Banco Central pede um relatório diário com as transações ordenadas por valor. Suponha que o computador do banco resolve operações simples na taxa de 2^{30} ($\approx$ 1 bilhão) operações por segundo.

Se o algoritmo disponível for o Bubblesort, o banco:

Exemplo sobre complexidade de ordenação

Problema

Um banco tem 10 milhões clientes que fazem, em média, 4 transações diárias. O Banco Central pede um relatório diário com as transações ordenadas por valor. Suponha que o computador do banco resolve operações simples na taxa de 2^{30} ($\approx$ 1 bilhão) operações por segundo.

Se o algoritmo disponível for o Bubblesort, o banco:

- a) conseguirá tranquilamente fazer o relatório a tempo;

Exemplo sobre complexidade de ordenação

Problema

Um banco tem 10 milhões clientes que fazem, em média, 4 transações diárias. O Banco Central pede um relatório diário com as transações ordenadas por valor. Suponha que o computador do banco resolve operações simples na taxa de 2^{30} ($\approx$ 1 bilhão) operações por segundo.

Se o algoritmo disponível for o Bubblesort, o banco:

- a) conseguirá tranquilamente fazer o relatório a tempo;
- b) deverá comprar um computador melhor;

Exemplo sobre complexidade de ordenação

Problema

Um banco tem 10 milhões clientes que fazem, em média, 4 transações diárias. O Banco Central pede um relatório diário com as transações ordenadas por valor. Suponha que o computador do banco resolve operações simples na taxa de 2^{30} ($\approx$ 1 bilhão) operações por segundo.

Se o algoritmo disponível for o Bubblesort, o banco:

- a) conseguirá tranquilamente fazer o relatório a tempo;
- b) deverá comprar um computador melhor;
- c) deverá contratar um programador melhor.

Quantas comparações o Bubblesort faz?

```
1 void bubblesort (int vetor[]) {
```

Quantas comparações o Bubblesort faz?

```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
```

Quantas comparações o Bubblesort faz?

```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
5         /*repeticao interna: compara e troca números */
6         for (int i = 0; i < N-1; i++)
```

Quantas comparações o Bubblesort faz?

```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
5         /*repeticao interna: compara e troca números */
6         for (int i = 0; i < N-1; i++)
7             if (vetor[i] > vetor[i+1])
```

Quantas comparações o Bubblesort faz?

```
1 void bubblesort (int vetor[]) {
2     int N = vetor.length;
3     /*controle do número de iterações*/
4     for (int iteracao = 0; iteracao < N-1; iteracao++)
5         /*repeticao interna: compara e troca números */
6         for (int i = 0; i < N-1; i++)
7             if (vetor[i] > vetor[i+1])
8                 troca(vetor, i, i+1); /*devemos trocar */
9 }
```

Termina com vetor ordenado de modo crescente.

Exemplo sobre complexidade

Análise:

- Custo de comparações do Bubblesort é $g(n) = (n-1)^2$

Exemplo sobre complexidade

Análise:

- Custo de comparações do Bubblesort é $g(n) = (n-1)^2$
- Elementos a ordenar: número de transações $n = 4 \times 10^7$

Exemplo sobre complexidade

Análise:

- Custo de comparações do Bubblesort é $g(n) = (n-1)^2$
- Elementos a ordenar: número de transações $n = 4 \times 10^7$
- Comparações necessárias: $(4 \times 10^7)^2$

Exemplo sobre complexidade

Análise:

- Custo de comparações do Bubblesort é $g(n) = (n-1)^2$
- Elementos a ordenar: número de transações $n = 4 \times 10^7$
- Comparações necessárias: $(4 \times 10^7)^2$
- Capacidade do computador: 2^{30} operações por segundo
- Tempo necessário em segundos: $(4 \times 10^7)^2 / 2^{30}$

Exemplo sobre complexidade

Análise:

- Custo de comparações do Bubblesort é $g(n) = (n-1)^2$
- Elementos a ordenar: número de transações $n = 4 \times 10^7$
- Comparações necessárias: $(4 \times 10^7)^2$
- Capacidade do computador: 2^{30} operações por segundo
- Tempo necessário em segundos: $(4 \times 10^7)^2 / 2^{30}$

$\approx$ 17 dias, 5 horas e 55 minutos

Counting sort

Ideia: ordenação de números inteiros

Cada elemento é representado por uma posição em um vetor.
Conta-se as repetições de cada número.

Counting sort

Ideia: ordenação de números inteiros

Cada elemento é representado por uma posição em um vetor.
Conta-se as repetições de cada número.

```
1 void countingSort(int[] vetor, int max) {
2     long[] count = new long[max+1];
```

Counting sort

Ideia: ordenação de números inteiros

Cada elemento é representado por uma posição em um vetor.
Conta-se as repetições de cada número.

```
1 void countingSort(int[] vetor, int max) {
2     long[] count = new long[max+1];
3     for (int i = 0; i < vetor.length; i++)
4         count[vetor[i]]++;// conta repetições
5
6
```

Counting sort

Ideia: ordenação de números inteiros

Cada elemento é representado por uma posição em um vetor.
Conta-se as repetições de cada número.

```
1 void countingSort(int[] vetor, int max) {
2     long[] count = new long[max+1];
3     for (int i = 0; i < vetor.length; i++)
4         count[vetor[i]]++;// conta repetições
5
6     for (int j = 0, i = 0; i < vetor.length; j++)
7         while (count[j]-- > 0) //tira em ordem
8             vetor[i++] = j;
9 }
```

Complexidade de espaço do Counting sort

Problema do banco: ordenação de 40 milhões de números

Se 1% das transações forem de mais de 1 milhão de reais, então é possível usar 2 algoritmos de ordenação:

Complexidade de espaço do Counting sort

Problema do banco: ordenação de 40 milhões de números

Se 1% das transações forem de mais de 1 milhão de reais, então é possível usar 2 algoritmos de ordenação:

- Para transações de até 1 milhão, usar o Counting sort: 0.672 segundos.
- Para transações de mais de 1 milhão, usar o Bubble sort: 2.5 minutos.

Complexidade de espaço do Counting sort

Problema do banco: ordenação de 40 milhões de números

Se 1% das transações forem de mais de 1 milhão de reais, então é possível usar 2 algoritmos de ordenação:

- Para transações de até 1 milhão, usar o Counting sort: 0.672 segundos.
- Para transações de mais de 1 milhão, usar o Bubble sort: 2.5 minutos.

Balanceamento de complexidades

Bubble sort: complexidade de tempo alta e complexidade de espaço baixa
Counting sort: complexidade de tempo baixa e complexidade de espaço muito alta

Limite assintótico de complexidade

Limite assintótico superior $O(g(n))$

Objetivo: encontrar função limitante superior $g(n)$ para representar o "teto" do custo do algoritmo.

Limite assintótico de complexidade

Limite assintótico superior $O(g(n))$

Objetivo: encontrar função limitante superior $g(n)$ para representar o "teto" do custo do algoritmo.

Bubble sort simplificado

Para n elementos, faz-se $n-1$ iterações e $n-1$ comparações em cada iteração: $g(n) = (n-1)^2$. Então a complexidade de tempo é $O(n^2)$.

Complexidades assintóticas

Complexidade de tempo

Como cada número é avaliado apenas uma vez, o Counting sort tem complexidade de tempo $O(n)$. Devido ao for aninhado, o Bubble sort tem complexidade de tempo $O(n^2)$.

Complexidades assintóticas

Complexidade de tempo

Como cada número é avaliado apenas uma vez, o Counting sort tem complexidade de tempo $O(n)$. Devido ao for aninhado, o Bubble sort tem complexidade de tempo $O(n^2)$.

Complexidade de espaço

Tamanho do vetor de contagens cresce de acordo com a faixa de valores na entrada. Complexidade de espaço é $O(max)$, sendo max o maior número no vetor de entrada. Se max for um dos maiores números armazenados em n bits então, espaço é $O(max) = O(2^n)$, Devido ao uso de um número constante variáveis auxiliares para trocas, o Bubble sort tem complexidade de espaço $O(1)$.