



UFPI – CCN – DC
Ciência da Computação
Estruturas de Dados

Algoritmos de Ordenação

Prof. Raimundo Moura
rsm@ufpi.edu.br

167

Algorithms ROBERT SEDGEWICK | KEVIN WAYNE

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*

➤ *Bubblesort*

2.2 Mergesort

2.3 Quicksort

ROBERT SEDGEWICK | KEVIN WAYNE
<http://algs4.cs.princeton.edu>

168

Problema de Ordenação

Ex. Student records in a university.

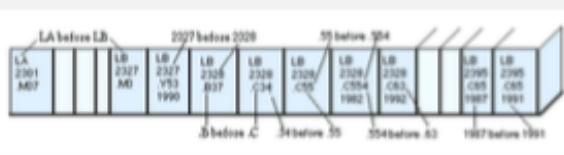
item →	Chen	3	A	991-878-4944	308 Blair
	Rohde	2	A	232-343-5555	343 Forbes
	Gazsi	4	B	766-093-9873	101 Brown
	Furia	1	A	766-093-9873	101 Brown
	Kanaga	3	B	898-122-9643	22 Brown
	Andrews	3	A	664-480-0023	097 Little
key →	Battle	4	C	874-088-1212	121 Whitman

Sort. Rearrange array of N items into ascending order.

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

169

Aplicações de Ordenação



Library of Congress numbers



FedEx packages



contacts



playing cards



Hogwarts houses

170

Exemplo de Cliente de Ordenação

Goal. Sort **any** type of data.

Ex 1. Sort random real numbers in ascending order.

seems artificial (stay tuned for an application)

```
public class Experiment
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        Double[] a = new Double[N];
        for (int i = 0; i < N; i++)
            a[i] = StdRandom.uniform();
        Insertion.sort(a);
        for (int i = 0; i < N; i++)
            StdOut.println(a[i]);
    }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

171

Exemplo de Cliente de Ordenação

Goal. Sort **any** type of data.

Ex 2. Sort strings in alphabetical order.

```
public class StringSorter
{
    public static void main(String[] args)
    {
        String[] a = StdIn.readAllStrings();
        Insertion.sort(a);
        for (int i = 0; i < a.length; i++)
            StdOut.println(a[i]);
    }
}
```

```
% more words3.txt
bed bug dad yet zoo ... all bad yes

% java StringSorter < words3.txt
all bad bed bug dad ... yes yet zoo
[suppressing newlines]
```

% java -jar StringSorted < input.txt

172

Exemplo de Cliente de Ordenação

Goal. Sort **any** type of data.
Ex 3. Sort the files in a given directory by filename.

```
import java.io.File;

public class FileSorter
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            StdOut.println(files[i].getName());
    }
}
```

```
% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
```

173

Ordenação Total

Goal. Sort **any** type of data (for which sorting is well defined).

A **total order** is a binary relation \leq that satisfies:

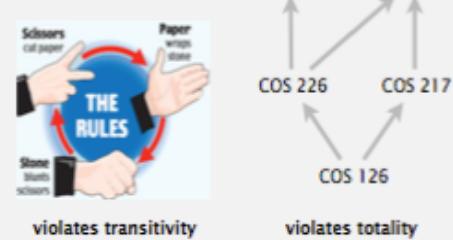
- Antisymmetry: if both $v \leq w$ and $w \leq v$, then $v = w$.
- Transitivity: if both $v \leq w$ and $w \leq x$, then $v \leq x$.
- Totality: either $v \leq w$ or $w \leq v$ or both.

Ex.

- Standard order for natural and real numbers.
- Chronological order for dates or times.
- Alphabetical order for strings.

No transitivity. Rock-paper-scissors.

No totality. PU course prerequisites.



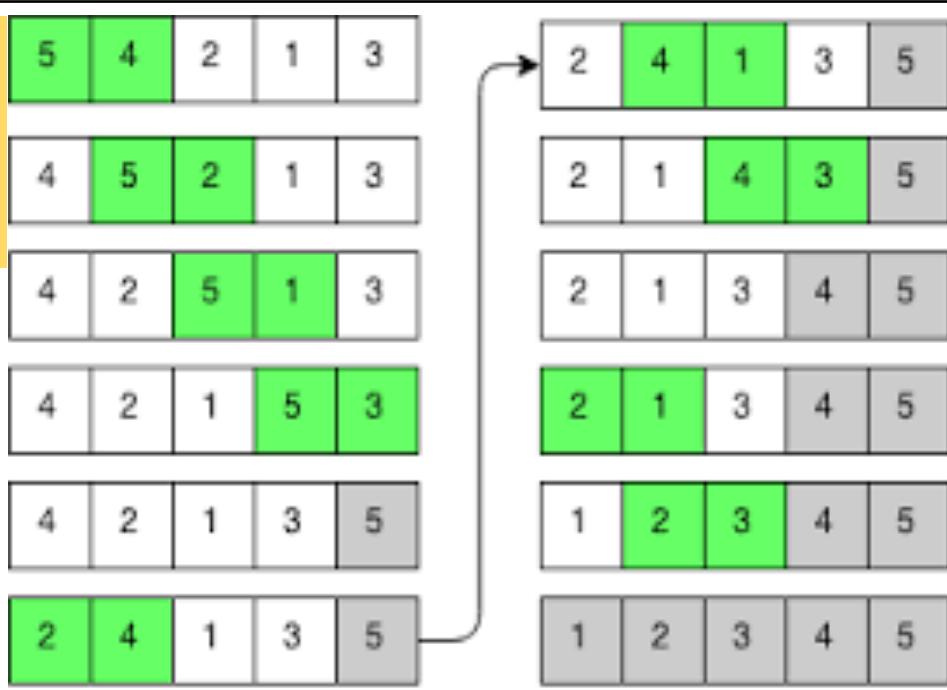
174

Algoritmos Elementares: *Bubblesort*

- Um dos mais simples algoritmos de ordenação (Método da Bolha)
- 1) Percorrer o vetor inteiro comparando elementos adjacentes (dois a dois);
- 2) Trocar as posições dos elementos se eles estiverem fora de ordem;
- 3) Repetir os passos acima com os primeiros $n-1$ itens, depois com os primeiros $n-2$ itens, até que reste apenas um item.

175

Algoritmos Elementares: *Bubblesort*



176

Algoritmos Elementares: *Bubblesort*

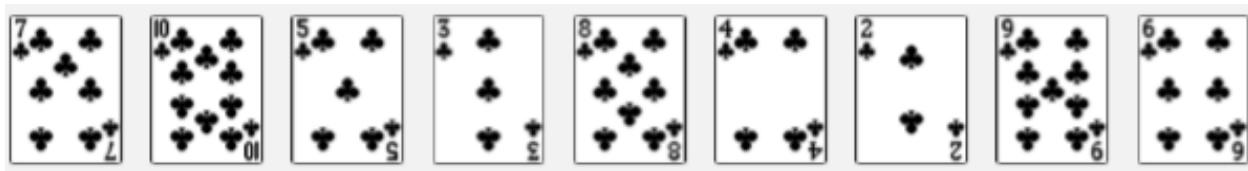
➤ COMPLEXIDADE:

- *Pior Caso: O(n²)*
- *Caso Médio: O(n²)*
- *Melhor Caso: O(n)*

177

Algoritmos Elementares: *Selection sort*

- Um dos mais simples algoritmos de ordenação
- 1) Achar o menor item no array e trocar com a primeira entrada
- 2) Encontrar o próximo menor item e trocar com a segunda entrada
- 3) E assim sucessivamente



178

Algoritmos Elementares: *Selection sort*

<i>i</i>	<i>min</i>	0	1	2	3	4	5	6	7	8	9	10	a[]
		S	O	R	T	E	X	A	M	P	L	E	
0	6	S	O	R	T	E	X	A	M	P	L	E	entries in black are examined to find the minimum
1	4	A	O	R	T	E	X	S	M	P	L	E	entries in red are $a[min]$
2	10	A	E	R	T	O	X	S	M	P	L	E	
3	9	A	E	E	T	O	X	S	M	P	L	R	
4	7	A	E	E	L	O	X	S	M	P	T	R	
5	7	A	E	E	L	M	X	S	O	P	T	R	
6	8	A	E	E	L	M	O	S	X	P	T	R	
7	10	A	E	E	L	M	O	P	X	S	T	R	
8	8	A	E	E	L	M	O	P	R	S	T	X	
9	9	A	E	E	L	M	O	P	R	S	T	X	entries in gray are in final position
10	10	A	E	E	L	M	O	P	R	S	T	X	
		A	E	E	L	M	O	P	R	S	T	X	

Trace of selection sort (array contents just after each exchange)

179

Algoritmos Elementares: *Selection sort*

➤ PROPOSIÇÃO:

- Realiza $\sim n^2/2$ comparações e n trocas para ordenar um array de tamanho n .

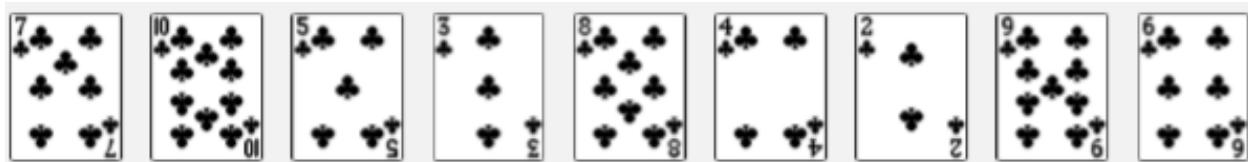
➤ COMPLEXIDADE:

- Pior Caso: $O(n^2)$
- Caso Médio: $O(n^2)$
- Melhor Caso: $O(n^2)$

180

Algoritmos Elementares: *Insertion sort*

- Algoritmo que as pessoas normalmente usam para classificar cartas de baralho. Cada carta é inserida em seu devido lugar entre as cartas já consideradas (mantendo-as ordenadas)
- Antes de inserir o item atual em uma posição, é preciso criar espaço, movendo os itens maiores uma posição à direita.



181

Algoritmos Elementares: *Insertion sort*

i	j	0	1	2	3	4	5	6	7	8	9	10
$a[]$												
1	0	O	S	R	T	E	X	A	M	P	L	E
2	1	O	R	S	T	E	X	A	M	P	L	E
3	3	O	R	S	T	E	X	A	M	P	L	E
4	0	E	O	R	S	T	X	A	M	P	L	E
5	5	E	O	R	S	T	X	A	M	P	L	E
6	0	A	E	O	R	S	T	X	M	P	L	E
7	2	A	E	M	O	R	S	T	X	P	L	E
8	4	A	E	M	O	P	R	S	T	X	L	E
9	2	A	E	L	M	O	P	R	S	T	X	E
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

Trace of insertion sort (array contents just after each insertion)

182

Algoritmos Elementares: *Insertion sort*

➤ PROPOSIÇÃO:

➤ Para arrays ordenados randomicamente de tamanho n com chaves distintas, realiza $\sim n^2/4$ comparações e $\sim n^2/4$ trocas em média. O pior caso é $\sim n^2/2$ comparações e $\sim n^2/2$ trocas e o melhor caso $n - 1$ comparações e 0 trocas.

➤ COMPLEXIDADE:

- Pior Caso: $O(n^2)$
- Caso Médio: $O(n^2)$
- Melhor Caso: $O(n)$

183

Algoritmos Elementares: *Shellsort*

➤ Extensão simples de *Insertion sort* que ganha velocidade ao permitir trocas de entradas que estão muito distantes, para produzir arrays parcialmente ordenados que podem ser ordenados com eficiência, eventualmente com *Insertion sort*.

- A ideia é reorganizar o array para fornecer a propriedade que, tomando todas as h -ésimas entradas (iniciando em qualquer lugar), produz uma sequência classificada.

184

Algoritmos Elementares: *Shellsort*

$h = 4$

L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R
L				M			P				T				
E				H			S			S					
E				L			O			X					
A				E			L				R				

$h = 13$

P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
P					S										
H													L		
E														E	
L															

(8 additional files of size 1)

An h -sorted file is h interleaved sorted files

185

Algoritmos Elementares: *Shellsort*

Input	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E
13-sort	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
4-sort	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E
	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E
	L	E	E	A	P	H	O	L	T	S	X	R	T	S	L	E
	L	E	E	A	M	H	O	L	P	S	X	R	T	S	L	E
	L	E	E	A	M	H	L	L	P	S	O	R	T	S	X	E
	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R

186

Algoritmos de Ordenação

Bubble Sort

Selection Sort

Insertion Sort

Shell Sort

187

Algoritmos Elementares: *Shellsort*

input	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E
13-sort	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
4-sort	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	E	E	A	M	H	L	L	P	S	O	T	S	X		E
	L	E	E	A	M	H	L	L	E	P	S	O	L	T	S	R
1-sort	E	L	E	A	M	H	L	E	P	S	O	L	T	S	X	R
	E	E	L	A	M	H	L	E	P	S	O	L	T	S	X	R
(I)	A	E	E	L	M	H	L	E	P	S	O	L	T	S	X	R
(II)	A	E	E	E	H	L	L	M	O	P	R	S	S	T	X	
result	A	E	E	E	H	L	L	M	O	P	R	S	S	T	X	

Detailed trace of shellsort (insertions)

188