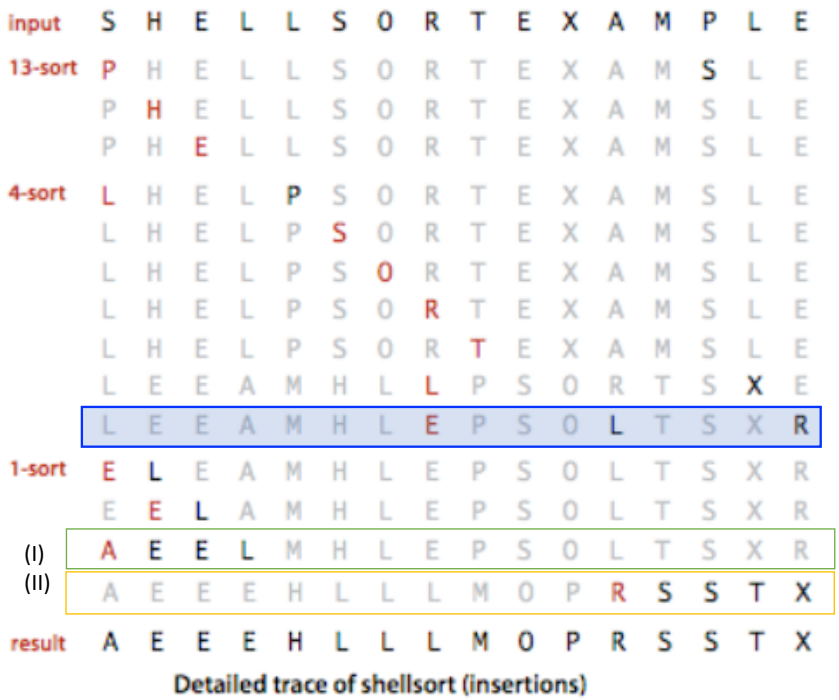
Algoritmos de Ordenação

- Bubble Sort
- Selection Sort
- Insertion Sort
- Shell Sort

187

Algoritmos Elementares: *Shellsort*



Detailed trace of shellsort (insertions)

188

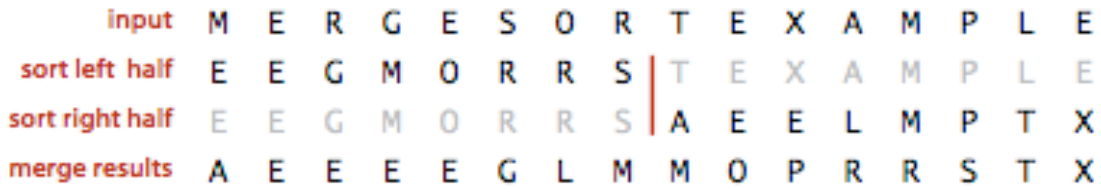
Algoritmos Elementares: *Shellsort*

- PROPRIEDADE:
  - O nº de comparações realizadas com incrementos 1, 4, 13, 40, 121, 364, … é limitado por um pequeno múltiplo de *n* vezes o nº de incrementos usados
- PROPOSIÇÃO:
  - O nº de comparações realizadas com os incrementos é $O(n^{3/2})$
- COMPLEXIDADE:
  - *Pior Caso: depende do gap: O(n log n)*
  - *Caso Médio: depende do gap*
  - *Melhor Caso: O(n log n)*

191

Mergesort

- Baseado na operação simples *"merging"* que combina dois arrays ordenados para fazer um array ordenado maior.
- Para ordenar um array, divide-o em duas metades, ordena-se as duas metades recursivamente e *merge* os resultados



Mergesort overview

192

Mergesort: Top-Down

	lo	hi	a[] 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 0, 0, 1)	0	1	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)	2	3	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 0, 1, 3)	0	3	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)	4	5	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)	6	7	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 5, 7)	4	7	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
merge(a, 0, 3, 7)	0	7	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
merge(a, 8, 8, 9)	8	9	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
merge(a, 10, 10, 11)	10	11	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
merge(a, 8, 9, 11)	8	11	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, 12, 12, 13)	12	13	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, 14, 14, 15)	14	15	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
merge(a, 12, 13, 15)	12	15	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)	8	15	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a, 0, 7, 15)	0	15	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Trace of merge results for top-down mergesort

193

Mergesort: Top-Down

- PROPOSIÇÃO:
  - A versão top-down realiza entre ½ *n log n* e *n log n* comparações e pelo menos *6n log n* acessos para ordenar um array de tamanho N.
- COMPLEXIDADE:
  - *Pior Caso: O(n log n)*
  - *Caso Médio: O(n log n)*
  - *Melhor Caso: O(n log n) ou O(n) para nr. naturais*

194

Mergesort: Bottom-Up

- Embora estejamos pensando em termos de unir dois grandes subarrays, o fato é que a maioria das fusões está mesclando pequenos subarrays.
- Organizar os *merges* para que façamos todas as mesclagens de pequenos arrays em um passo, depois façamos um segundo passo para mesclar esses arrays em pares e assim por diante, continuando até fazermos uma mesclagem que englobe o todo o array.

195

Mergesort: Bottom-Up

	a[i]														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
sz = 2																
merge(a, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
merge(a, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
sz = 4																
merge(a, 0, 1, 3)	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
merge(a, 4, 5, 7)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
merge(a, 8, 9, 11)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
merge(a, 12, 13, 15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
sz = 8																
merge(a, 0, 3, 7)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
sz = 16																
merge(a, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Trace of merge results for bottom-up mergesort

196

Mergesort: Bottom-Up

- PROPOSIÇÃO:
  - A versão bottom-up realiza entre ½ *n log n* e *n log n* comparações e pelo menos *6n log n* acessos para ordenar um array de tamanho N.
  - Nenhum algoritmo de ordenação baseado em comparações pode garantir classificar N itens com menos de $\log(n!) \sim n \log n$ comparações
  - *Mergesort* é um algoritmo asintoticamente ótimo

197

Troca de valores entre variáveis inteiras, sem usar memória auxiliar. Dúvida: e para racionais?

• A=7.5	• A=7.5
• B = 4.3333	• B = 4.3333
• A = A + B	• A = 7.5 + 4.3333 = 11.8333
• B = A – B	• B = 11.8333 – 4.3333 = 7.5
• A = A - B	• A = 11.8333 – 7.5 = 4.3333

198

Mergesort: Bottom-Up

- COMPLEXIDADE:
  - *Pior Caso: O(n log n)*
  - *Caso Médio: O(n log n)*
  - *Melhor Caso: O(n log n) ou O(n) para nr. naturais*

199

Mergesort

- MELHORIAS:
  - Usar *Insertion sort* para pequenos arrays pode melhorar o tempo de execução de 10 a 15%
  - Testar se o array já está ordenado
  - Eliminar a cópia para o array auxiliar

200

Quicksort

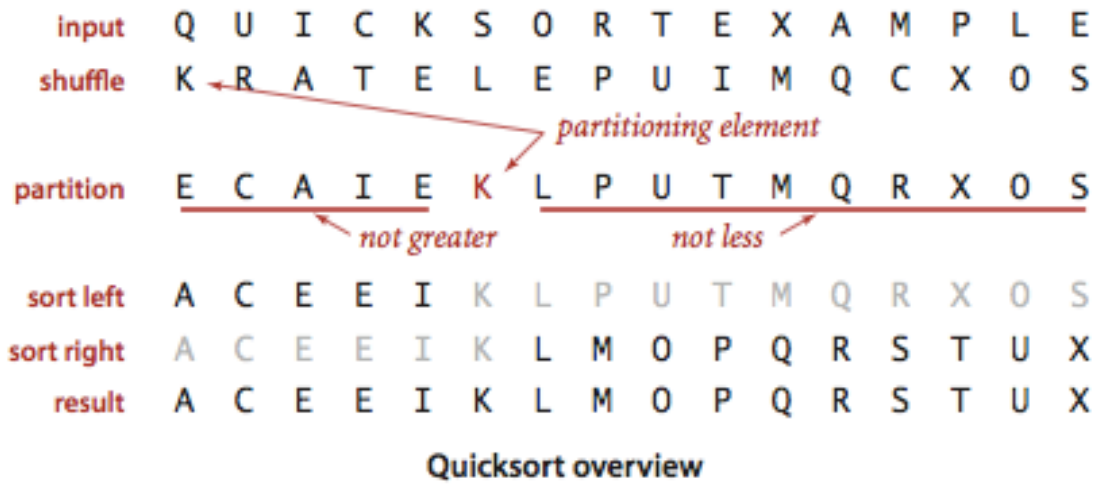
- Popular porque não é difícil implementar
- Trabalha bem para uma variedade de diferentes tipos de dados
- Mais rápido do que qualquer outro método de classificação em aplicações típicas

- Quanto ao espaço, usa apenas uma pequena pilha auxiliar
- Requer tempo proporcional a *n log n* para ordenar N itens

201

Quicksort: funcionamento básico

- Particiona o array em duas partes e, então, as ordena independentemente



Quicksort overview

202

Quicksort: Funcionamento básico

- O cerne do método é o particionamento, que rearranja o array para garantir as condições:
  1) A entrada *a[j]* está na sua posição final no array, para algum j;
  2) Nenhuma entrada de *a[lo]* a *a[j-1]* é maior que *a[j]*;
  3) Nenhuma entrada de *a[j+1]* a *a[hi]* é menor que *a[j]*;

OBS: lo é o limite inferior e hi é o limite superior do array

203

Quicksort: Considerações

- É um *método dividir-e-conquistar*
- Conseguimos uma classificação completa por particionamento, aplicando recursivamente o método aos subarrays.
- É um *algoritmo aleatório*, porque aleatoriamente embaralha o array antes de classificá-lo.

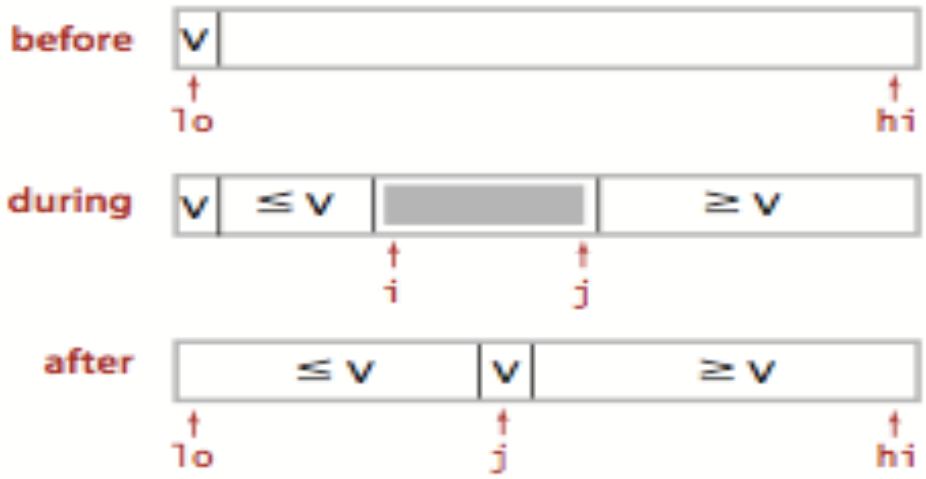
204

Quicksort: Particionamento

1) Escolhemos *a[lo]* como *pivô* (item que está na sua posição final);
2) Percorremos a partir da extremidade esquerda do array até encontrarmos uma entrada maior que (ou igual a) ao pivô;
3) Percorremos a partir da extremidade direita do array até encontrarmos uma entrada menor que (ou igual a) ao pivô

205

Quicksort: Particionamento



Quicksort partitioning overview

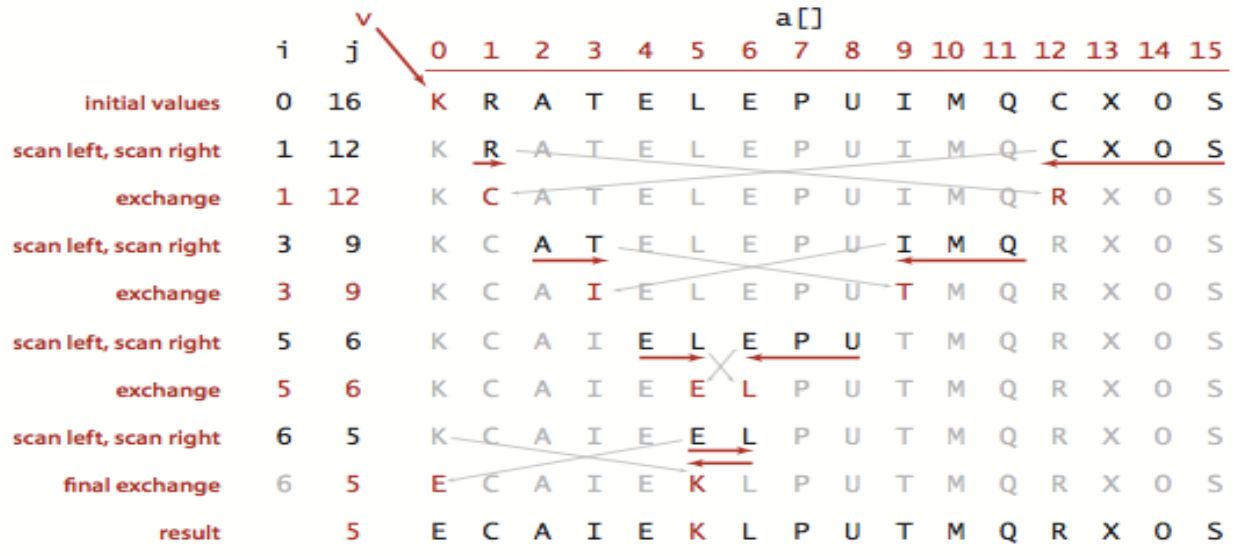
206

Quicksort: Particionamento

4) Os dois itens (*i* e *j*) estão fora de posição e devem ser trocados;
5) Quando os índices se cruzam, temos que trocar o item *a[lo]* com a entrada mais à direita do subarray da esquerda (*a[j]*) e retornar o índice *j*

207

Quicksort: Particionamento



Partitioning trace (array contents before and after each exchange)

208

Quicksort:

209

Quicksort

- PROPOSIÇÃO:
  - Realiza $2n \log n$ comparações (e $1/6$ que muitas trocas) em média para ordenar um array de tamanho N com chaves distintas.
  - No pior caso faz $\sim n^2/2$ comparações, mas o embaralhamento proteje contra esse caso.
- COMPLEXIDADE:
  - *Pior Caso: $O(n^2)$*
  - *Caso Médio: O(n log n)*
  - *Melhor Caso: O(n log n)*

210