

AULA 20

função	consumo de tempo	observação
bubble	$O(n^2)$	todos os casos
insercao	$O(n^2)$ $O(n)$	piores caso melhor caso
insercaoBinaria	$O(n^2)$ $O(n \lg n)$	piores caso melhor caso
selecao	$O(n^2)$	todos os casos
mergeSort	$O(n \lg n)$	todos os casos

Navigation icons

Navigation icons

Divisão e conquista

Algoritmos por **divisão-e-conquista** têm três passos em cada nível da recursão:

Dividir: o problema é dividido em subproblemas de tamanho menor;

Conquistar: os subproblemas são resolvidos **recursivamente** e subproblemas “pequenos” são resolvidos diretamente;

Combinar: as soluções dos subproblemas são combinadas para obter uma solução do problema original.

Exemplo: ordenação por intercalação (**mergeSort**).

Navigation icons

Navigation icons

AULA 21

Ordenação: algoritmo Quicksort

Problema da separação

Problema: Rearranjar um dado vetor $v[p..r-1]$ e devolver um índice q , $p \leq q < r$, tais que

$$v[p..q-1] \leq v[q] < v[q+1..r-1]$$

Entra:

	p									r
v	99	33	55	77	11	22	88	66	33	44

Navigation icons

Navigation icons

Problema da separação

Problema: Rearranjar um dado vetor $v[p..r-1]$ e devolver um índice q , $p \leq q < r$, tais que

$$v[p..q-1] \leq v[q] < v[q+1..r-1]$$

Entra:

	p										r
v	99	33	55	77	11	22	88	66	33	44	

Sai:

	p			q							r
v	33	11	22	33	44	55	99	66	77	88	

	p										r
v	99	33	55	77	11	22	88	66	33	44	

	i	j									x
v	99	33	55	77	11	22	88	66	33	44	

	i	j									x
v	99	33	55	77	11	22	88	66	33	44	

	i	j									x
v	99	33	55	77	11	22	88	66	33	44	
	i	j									x
v	33	99	55	77	11	22	88	66	33	44	

	i	j									x
v	99	33	55	77	11	22	88	66	33	44	
	i	j									x
v	33	99	55	77	11	22	88	66	33	44	

Separa

i		j								x
v	99	33	55	77	11	22	88	66	33	44
	i			j						x
v	33	99	55	77	11	22	88	66	33	44

Separa

i		j								x
v	99	33	55	77	11	22	88	66	33	44
	i			j						x
v	33	99	55	77	11	22	88	66	33	44
	i			j						x
v	33	11	55	77	99	22	88	66	33	44

Separa

i		j								x
v	99	33	55	77	11	22	88	66	33	44
	i			j						x
v	33	99	55	77	11	22	88	66	33	44
	i			j						x
v	33	11	55	77	99	22	88	66	33	44
	i			j						x
v	33	11	22	77	99	55	88	66	33	44

Separa

i		j								x
v	99	33	55	77	11	22	88	66	33	44
	i			j						x
v	33	99	55	77	11	22	88	66	33	44
	i			j						x
v	33	11	55	77	99	22	88	66	33	44
	i			j						x
v	33	11	22	77	99	55	88	66	33	44

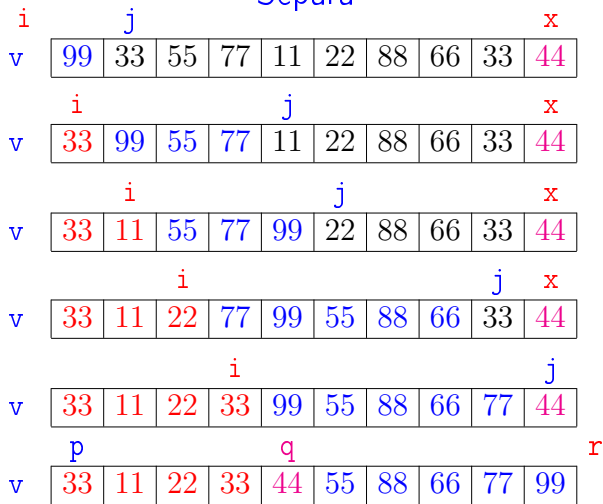
Separa

i		j								x
v	99	33	55	77	11	22	88	66	33	44
	i			j						x
v	33	99	55	77	11	22	88	66	33	44
	i			j						x
v	33	11	55	77	99	22	88	66	33	44
	i			j						x
v	33	11	22	77	99	55	88	66	33	44

Separa

i		j								x
v	99	33	55	77	11	22	88	66	33	44
	i			j						x
v	33	99	55	77	11	22	88	66	33	44
	i			j						x
v	33	11	55	77	99	22	88	66	33	44
	i			j						x
v	33	11	22	77	99	55	88	66	33	44
	i			j						x
v	33	11	22	33	99	55	88	66	77	44

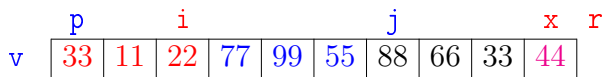
Separa



Invariantes

Em /*A*/ vale que

$$(i0) \ v[p..i] \leq x < v[i+1..j-1]$$



Consumo de tempo

Supondo que a execução de cada linha consome 1 unidade de tempo.

Qual o consumo de tempo da função `separa` em termos de $n := r - p$?

linha	consumo de todas as execuções da linha
1	= 1
2	= $n + 1$
3	= n
4	$\leq n$
5	= 1
total	$\leq 3n + 3$

$$= O(n)$$

Função separa

Rearranja $v[p..r-1]$ de modo que $p \leq q < r$ e $v[p..q-1] \leq v[q] < v[q+1..r-1]$.

A função devolve q .

```
int separa (int p, int r, int v[]) {
1  int i = p-1, j, x = v[r-1], t;

2  for (j = p; /*A*/ j < r; j++)
3      if (v[j] <= x) {
4          t=v[++i]; v[i]=v[j]; v[j]=t;
5      }
6  return i;
}
```

Consumo de tempo

Supondo que a execução de cada linha consome 1 unidade de tempo.

Qual o consumo de tempo da função `separa` em termos de $n := r - p$?

linha	consumo de todas as execuções da linha
1	= ?
2	= ?
3	= ?
4	= ?
5	= ?
total	= ?

Conclusão

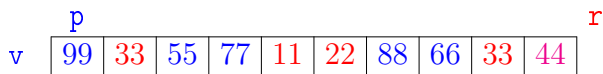
O consumo de tempo da função `separa` é proporcional a n .

O consumo de tempo da função `separa` é $O(n)$.

Quicksort

Rearranja $v[p..r-1]$ em ordem crescente.

```
void quickSort (int p, int r, int v[]) {  
1  if (p < r-1) {  
2      int q = separa(p,r,v);  
3      quickSort(p, q, v);  
4      quickSort(q+1, r, v);  
    }  
}
```

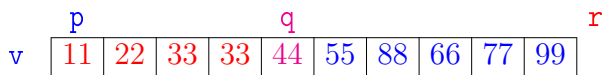


Navigation icons

Quicksort

Rearranja $v[p..r-1]$ em ordem crescente.

```
void quickSort (int p, int r, int v[]) {  
1  if (p < r-1) {  
2      int q = separa(p,r,v);  
3      quickSort(p, q, v);  
4      quickSort(q+1, r, v);  
    }  
}
```



Navigation icons

Quicksort

Rearranja $v[p..r-1]$ em ordem crescente.

```
void quickSort (int p, int r, int v[]) {  
1  if (p < r-1) {  
2      int q = separa(p,r,v);  
3      quickSort(p, q, v);  
4      quickSort(q+1, r, v);  
    }  
}
```

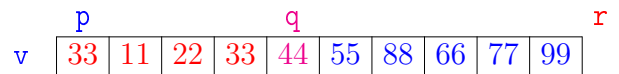
Consumo de tempo?

Navigation icons

Quicksort

Rearranja $v[p..r-1]$ em ordem crescente.

```
void quickSort (int p, int r, int v[]) {  
1  if (p < r-1) {  
2      int q = separa(p,r,v);  
3      quickSort(p, q, v);  
4      quickSort(q+1, r, v);  
    }  
}
```



No começo da linha 3,

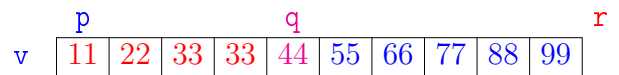
$$v[p..q-1] \leq v[q] < v[q+1..r-1]$$

Navigation icons

Quicksort

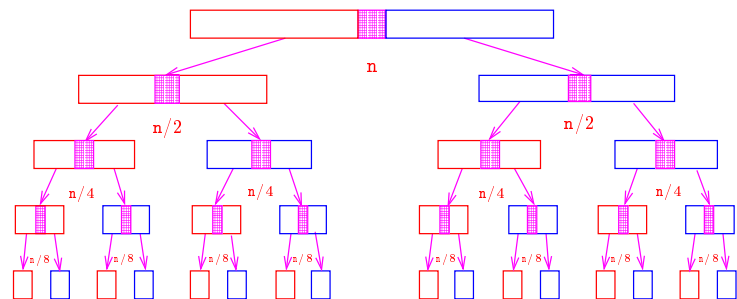
Rearranja $v[p..r-1]$ em ordem crescente.

```
void quickSort (int p, int r, int v[]) {  
1  if (p < r-1) {  
2      int q = separa(p,r,v);  
3      quickSort(p, q, v);  
4      quickSort(q+1, r, v);  
    }  
}
```



Navigation icons

Consumo de tempo: versão MAC0122



Navigation icons

Consumo de tempo: versão MAC0122

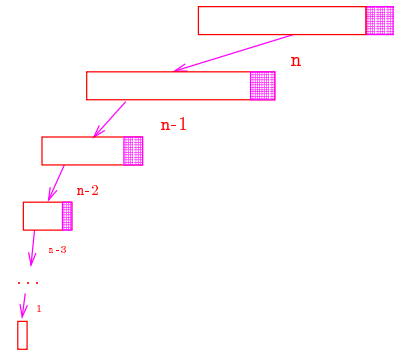
O consumo de tempo em cada nível da recursão é proporcional a n .

No melhor caso, em cada chamada recursiva, q é $\approx (p + r)/2$.

Nessa situação há cerca de $\lg n$ níveis de recursão.

nível	consumo de tempo (proporcional a)
1	$\approx n$
2	$\approx n/2 + n/2$
3	$\approx n/4 + n/4 + n/4 + n/4 + n/4$
...	...
$\lg n$	$\approx 1 + 1 + 1 + 1 \dots + 1 + 1 + 1 + 1$
Total	$\approx n \lg n = O(n \lg n)$

Consumo de tempo: versão MAC0122



Consumo de tempo: versão MAC0122

No pior caso, em cada chamada recursiva, o valor de q devolvido por `separa` é $\approx p$ ou $\approx r$.

Nessa situação há cerca de n níveis de recursão.

nível	consumo de tempo (proporcional a)
1	$\approx n$
2	$\approx n - 1$
3	$\approx n - 2$
4	$\approx n - 3$
...	...
n	≈ 1
Total	$\approx n(n - 1)/2 = O(n^2)$

Quicksort no melhor caso

No melhor caso, em cada chamada recursiva q é aproximadamente $(p + r)/2$.

O consumo de tempo da função `quickSort` no melhor caso é proporcional a $n \log n$.

O consumo de tempo da função `quickSort` no melhor caso é $O(n \log n)$.

Quicksort no pior caso

O consumo de tempo da função `quickSort` no pior caso é proporcional a n^2 .

O consumo de tempo da função `quickSort` no pior caso é $O(n^2)$.

O consumo de tempo da função `quickSort` é $O(n^2)$.

Discussão geral

Pior caso, melhor caso, todos os casos?!?!?

Dado um algoritmo \mathcal{A} o que significam as expressões:

- ▶ \mathcal{A} é $O(n^2)$ no pior caso.
- ▶ \mathcal{A} é $O(n^2)$ no melhor caso.
- ▶ \mathcal{A} é $O(n^2)$.

Análise experimental

Algoritmos implementados:

mergeR `mergeSort` recursivo.
mergeI `mergeSort` iterativo.
quick `quickSort` recursivo.
qsort quicksort da [biblioteca do C](#).

Análise experimental

A **plataforma utilizada** nos experimentos foi um computador rodando Ubuntu GNU/Linux 3.5.0-17

Compilador:

```
gcc -Wall -ansi -O2 -pedantic  
-Wno-unused-result.
```

Computador:

```
model name: Intel(R) Core(TM)2 Quad CPU Q6600 @  
2.40GHz  
cpu MHz : 1596.000  
cache size: 4096 KB  
MemTotal : 3354708 kB
```

Estudo empírico (aleatório)

n	mergeR	mergeI	quick	qsort
8192	0.00	0.00	0.00	0.00
16384	0.00	0.00	0.00	0.00
32768	0.01	0.01	0.01	0.01
65536	0.01	0.01	0.01	0.01
131072	0.02	0.02	0.02	0.03
262144	0.05	0.04	0.04	0.07
524288	0.10	0.08	0.08	0.15
1048576	0.21	0.20	0.17	0.31
2097152	0.44	0.43	0.35	0.66
4194304	0.91	0.89	0.73	1.37
8388608	1.91	1.88	1.52	2.86

Tempos em segundos.

Estudo empírico (crescente)

n	mergeR	mergeI	quick	qsort
1024	0.00	0.00	0.00	0.00
2048	0.00	0.00	0.01	0.00
4096	0.00	0.00	0.01	0.00
8192	0.00	0.00	0.04	0.00
16384	0.00	0.00	0.14	0.01
32768	0.00	0.00	0.57	0.00
65536	0.00	0.01	2.27	0.01
131072	0.02	0.01	9.07	0.02

Tempos em segundos.

Para $n=262144$ `quickSort` dá **Segmentation fault (core dumped)**

Estudo empírico (decrecente)

n	mergeR	mergeI	quick	qsort
1024	0.00	0.00	0.00	0.00
2048	0.00	0.00	0.00	0.00
4096	0.01	0.00	0.01	0.00
8192	0.00	0.00	0.03	0.00
16384	0.00	0.00	0.14	0.01
32768	0.00	0.01	0.57	0.00
65536	0.01	0.01	2.27	0.01
131072	0.02	0.01	9.06	0.02

Tempos em segundos.

Para $n=262144$ `quickSort` dá **Segmentation fault (core dumped)**

Consumo de tempo: versão MAC0338

Quanto tempo consome a função `quickSort` em termos de $n := r - p$?

linha	consumo de todas as execuções da linha
1	= ?
2	= ?
3	= ?
4	= ?
<hr/>	
total	= ????

Consumo de tempo: versão MAC0338

Quanto tempo consome a função `quickSort` em termos de $n := r - p$?

linha	consumo de todas as execuções da linha
1	= $O(1)$
2	= $O(n)$
3	= $T(k)$
4	= $T(n - k - 1)$

$$\text{total} = T(k) + T(n - k - 1) + O(n + 1)$$

$$0 \leq k := q - p \leq n - 1$$

Recorrência: versão MAC0338

$T(n)$:= consumo de tempo máximo quando $n := r - p$

$$T(0) = O(1)$$

$$T(1) = O(1)$$

$$T(n) = T(k) + T(n - k - 1) + O(n) \text{ para } n = 2, 3, 4, \dots$$

Recorrência grosseira:

$$T(n) = T(0) + T(n - 1) + O(n)$$

$T(n)$ é $O(???)$.

Recorrência cuidadosa: versão MAC0338

$T(n)$:= consumo de tempo máximo quando $n = r - p$

$$T(0) = O(1)$$

$$T(1) = O(1)$$

$$T(n) = \max_{0 \leq k \leq n-1} \{T(k) + T(n - k - 1)\} + O(n) \\ \text{para } n = 2, 3, 4, \dots$$

$T(n)$ é $O(n^2)$.

Demonstração: ... em MAC0338.

Recorrência: versão MAC0338

$T(n)$:= consumo de tempo máximo quando $n := r - p$

$$T(0) = O(1)$$

$$T(1) = O(1)$$

$$T(n) = T(k) + T(n - k - 1) + O(n) \text{ para } n = 2, 3, 4, \dots$$

Recorrência: versão MAC0338

$T(n)$:= consumo de tempo máximo quando $n := r - p$

$$T(0) = O(1)$$

$$T(1) = O(1)$$

$$T(n) = T(k) + T(n - k - 1) + O(n) \text{ para } n = 2, 3, 4, \dots$$

Recorrência grosseira:

$$T(n) = T(0) + T(n - 1) + O(n)$$

$T(n)$ é $O(n^2)$.

Demonstração: ... em MAC0338.