Melhores momentos

AULA 16

Ordenação

$\mathtt{v}[0\,..\,\mathtt{n}{-}1]$ é crescente se $\mathtt{v}[0] \leq \cdots \leq \mathtt{v}[\mathtt{n}{-}1]$.

Problema: Rearranjar um vetor $\mathtt{v}[0\,..\,\mathtt{n}{-}1]$ de modo que ele fique crescente.

Entra:

1										n−1
33	55	33	44	33	22	11	99	22	55	77

Ordenação

$\mathtt{v}[0\,..\,\mathtt{n}{-}1]$ é crescente se $\mathtt{v}[0] \leq \cdots \leq \mathtt{v}[\mathtt{n}{-}1]$.

Problema: Rearranjar um vetor $\mathtt{v}[0\,..\,\mathtt{n}{-}1]$ de modo que ele fique crescente.

Entra:

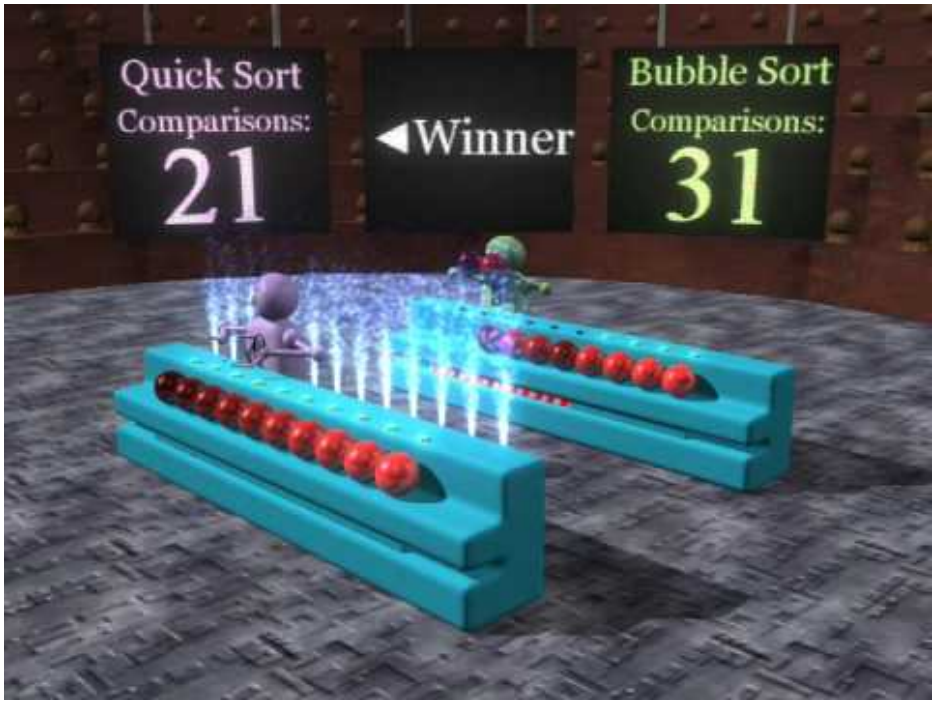
1										n−1
33	55	33	44	33	22	11	99	22	55	77

Sai:

0										n−1
11	22	22	33	33	33	44	55	55	77	99

AULA 17

Ordenação: algoritmo Quicksort



Fonte: https://www.youtube.com/watch?v=vxENKlcs2Tw/

PF 11

http://www.ime.usp.br/ pf/algoritmos/aulas/quick.html

Problema da separação

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

$$\mathtt{v}[\mathtt{p}\,..\,\mathtt{q}-1] \le \mathtt{v}[\mathtt{q}] < \mathtt{v}[\mathtt{q}+1\,..\,\mathtt{r}-1]$$

Entra:

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

Problema da separação

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

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

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

Separa

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

Separa

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

Separa

	i	j								x
v	99	33	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

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

Função separa

Rearranja $v[p\,..\,r-1]$ de modo que $p \le q < r$
e $v[p\,..\,q-1] \le v[q] < v[q+1\,..\,r-1]$.
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   return i;
}
```

Invariantes

Em /*A*/ vale que

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

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

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 $\mathtt{n} := \mathtt{r} - \mathtt{p}$?

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

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 $\mathtt{n} := \mathtt{r} - \mathtt{p}$?

linha		consumo de todas as execuções da linha
1	$=$	1
2	$=$	$\mathtt{n} + 1$
3	$=$	$\mathtt{n}$
4	$\leq$	$\mathtt{n}$
5	$=$	1
total	$\leq$	$3\mathtt{n} + 3 \qquad = \mathrm{O}(\mathtt{n})$

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);
    }
}
```

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

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);
    }
}
```

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

No começo da linha 3,

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

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);
    }
}
```

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

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);
    }
}
```

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

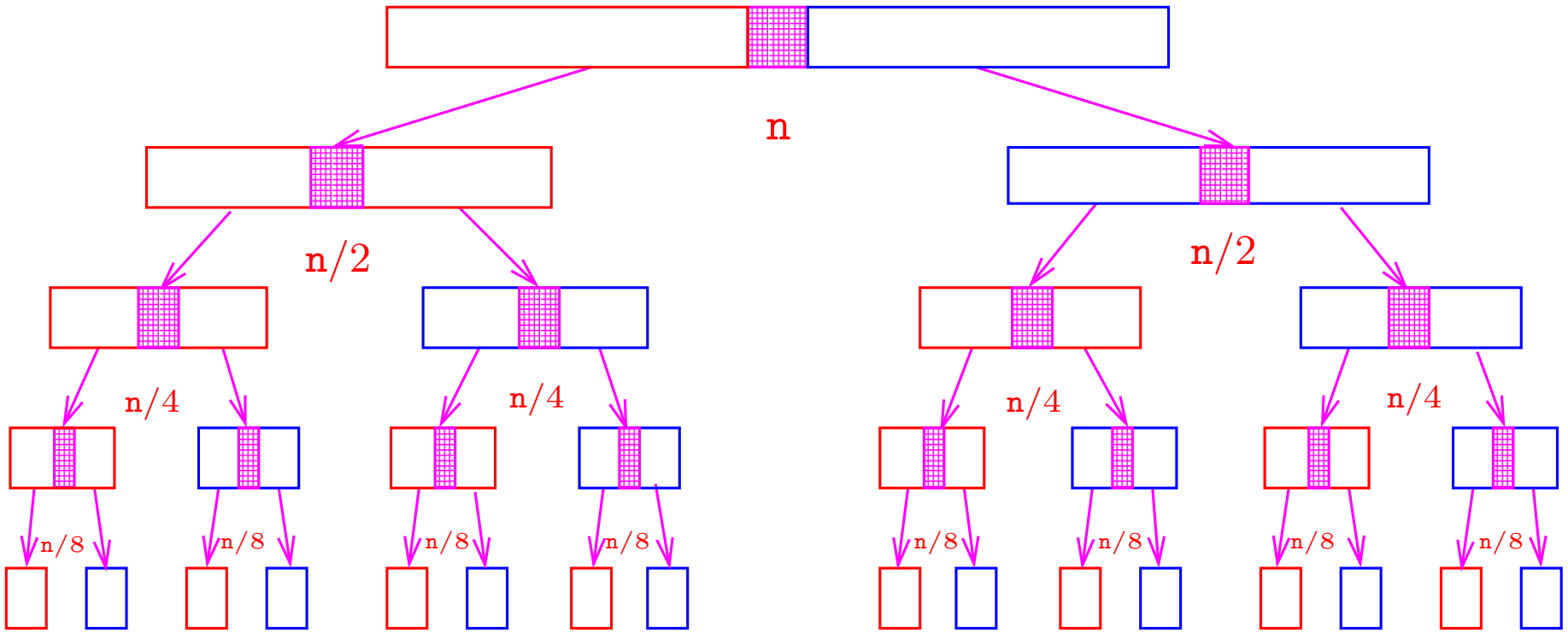
Quicksort

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?

Consumo de tempo: versão MAC0122



Consumo de tempo: versão MAC0122

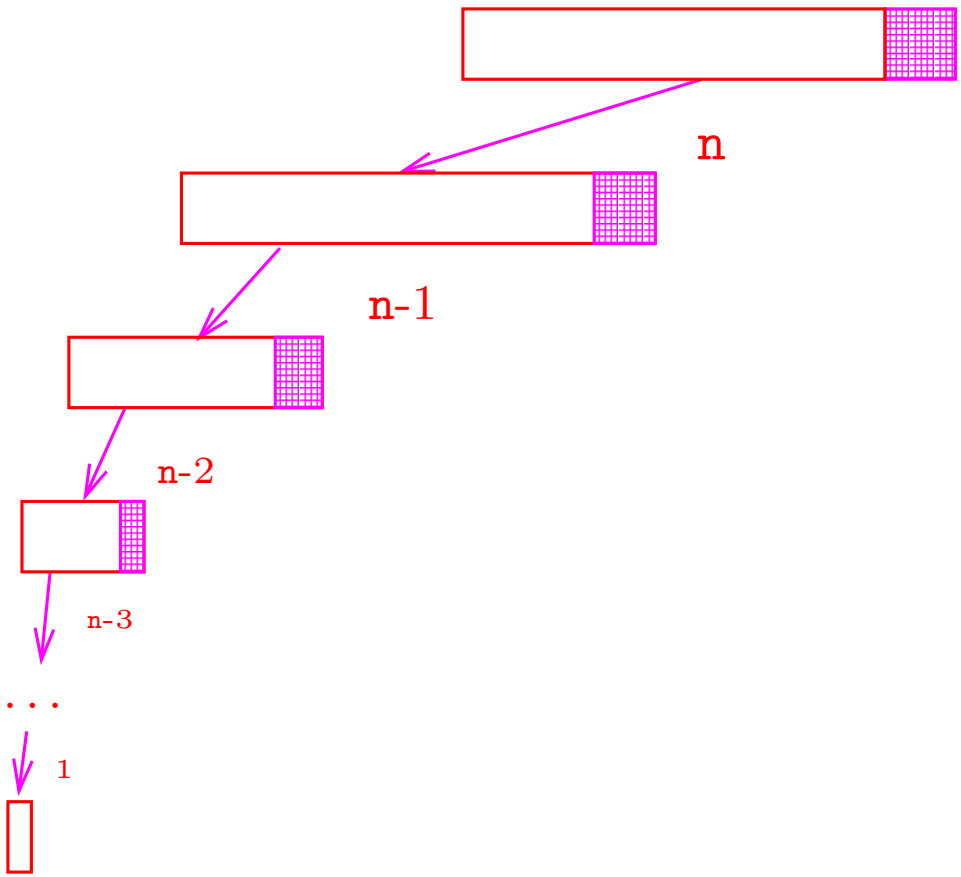
O consumo de tempo em cada nível da recursão é proporcional a $\mathtt{n}$.

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

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

nível	consumo de tempo (proporcional a)
1	$\approx \mathtt{n}$
2	$\approx \mathtt{n}/2 + \mathtt{n}/2$
3	$\approx \mathtt{n}/4 + \mathtt{n}/4 + \mathtt{n}/4 + \mathtt{n}/4 + \mathtt{n}/4$
$\ldots$	$\ldots$
$\lg \mathtt{n}$	$\approx 1 + 1 + 1 + 1 \cdots + 1 + 1 + 1 + 1$
Total	$\approx \mathtt{n} \lg \mathtt{n} = \mathrm{O}(\mathtt{n} \lg \mathtt{n})$

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, $\mathtt{q}$ é aproximadamente $(\mathtt{p} + \mathtt{r})/2$.

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

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

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

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 (decrescente)

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)`.

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)`

Consumo de tempo: outra versão

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

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

Consumo de tempo: outra versão

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

linha		consumo de todas as execuções da linha
1	$=$	$\mathrm{O}(1)$
2	$=$	$\mathrm{O}(\mathtt{n})$
3	$=$	$\mathtt{T}(\mathtt{k})$
4	$=$	$\mathtt{T}(\mathtt{n} - \mathtt{k} - 1)$
total	$=$	$\mathtt{T}(\mathtt{k}) + \mathtt{T}(\mathtt{n} - \mathtt{k} - 1) + \mathrm{O}(\mathtt{n} + 1)$

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

Recorrência: outra versão

$\mathtt{T}(\mathtt{n})$:= consumo de tempo máximo quando $\mathtt{n} := \mathtt{r} - \mathtt{p}$

$$\mathtt{T}(0) = \mathrm{O}(1)$$
$$\mathtt{T}(1) = \mathrm{O}(1)$$
$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(\mathtt{k}) + \mathtt{T}(\mathtt{n} - \mathtt{k} - 1) + \mathrm{O}(\mathtt{n}) \quad \text{para } \mathtt{n} = 2, 3, 4, \ldots$$

Recorrência: outra versão

$\mathtt{T}(\mathtt{n})$:= consumo de tempo máximo quando $\mathtt{n} := \mathtt{r} - \mathtt{p}$

$$\mathtt{T}(0) = \mathrm{O}(1)$$
$$\mathtt{T}(1) = \mathrm{O}(1)$$
$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(\mathtt{k}) + \mathtt{T}(\mathtt{n} - \mathtt{k} - 1) + \mathrm{O}(\mathtt{n}) \quad \text{para } \mathtt{n} = 2, 3, 4, \ldots$$

Recorrência grosseira:

$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(0) + \mathtt{T}(\mathtt{n} - 1) + \mathrm{O}(\mathtt{n})$$

$\mathtt{T}(\mathtt{n})$ é $\mathrm{O}(???)$.

Recorrência: outra versão

$\mathtt{T}(\mathtt{n})$:= consumo de tempo máximo quando $\mathtt{n} := \mathtt{r} - \mathtt{p}$

$$\mathtt{T}(0) = O(1)$$
$$\mathtt{T}(1) = O(1)$$
$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(\mathtt{k}) + \mathtt{T}(\mathtt{n} - \mathtt{k} - 1) + O(\mathtt{n}) \quad \text{para } \mathtt{n} = 2, 3, 4, \ldots$$

Recorrência grosseira:

$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(0) + \mathtt{T}(\mathtt{n} - 1) + O(\mathtt{n})$$

$\mathtt{T}(\mathtt{n})$ é $O(\mathtt{n}^2)$.

Demonstração: ...

Recorrência cuidadosa: ...

$\mathtt{T}(\mathtt{n})$:= consumo de tempo máximo quando $\mathtt{n} = \mathtt{r} - \mathtt{p}$

$$\begin{aligned}
\mathtt{T}(0) &= \mathrm{O}(1) \\
\mathtt{T}(1) &= \mathrm{O}(1) \\
\mathtt{T}(\mathtt{n}) &= \max_{0 \le \mathtt{k} \le \mathtt{n}-1} \{\mathtt{T}(\mathtt{k}) + \mathtt{T}(\mathtt{n} - \mathtt{k} - 1)\} + \mathrm{O}(\mathtt{n})
\end{aligned}$$

para $\mathtt{n} = 2, 3, 4, \ldots$

$\mathtt{T}(\mathtt{n})$ é $\mathrm{O}(\mathtt{n}^2)$.

Demonstração: ...

`quickSort`: versão iterativa

Na versão iterativa devemos administrar uma pilha que simula a pilha da recursão.

A pilha armazenará os índices que delimitam segmentos do vetor que estão à espera de ordenação.

Na implementação da pilha

- `stackInit()` cria uma pilha;
- `stackEmpty()` retorna 0 se a pilha não está vazia;
- `stackPush()` põe um índice no topo da pilha;
- `stackPop()` tira e retorna o índice no topo da pilha;
- `stackFree()` destrói a pilha.

quickSort: versão iterativa

```
void quickSort (int n, int v[]) {
   int p = 0; /* inicio segmentos */
   int r = n; /* fim segmentos */
   int q;

   /* inicialize a pilha */
   stackInit();
   stackPush(r);
   stackPush(p);
```

quickSort: versão iterativa

```
  while (!stackEmpty()) {
    p = stackPop();
    r = stackPop();
    if (p >= r-1) continue;
    q = separa(p,r,v);
    /* segmento inicial */
    stackPush(q);
    stackPush(p);
    /* segmento final */
    stackPush(r);
    stackPush(q+1);
  }
}
```

quickSort: outra versão iterativa

```
void quickSort (int n, int v[]) {
   int p = 0; /* inicio segmentos */
   int r = n; /* fim segmentos */
   int q;

   /* inicialize a pilha */
   stackInit();
   stackPush(r);
   stackPush(p);
```

quickSort: outra versão iterativa

```
while (!stackEmpty()) {
   p = stackPop();    r = stackPop();
   if (p >= r-1) continue;
   q = separa(p,r,v);
   if (q - p > r - q) {
      stackPush(q);    stackPush(p);
      stackPush(r);    stackPush(q+1);
   }
   else {
      stackPush(r);    stackPush(q+1);
      stackPush(q);    stackPush(p);
   }
}
}
```

Exercícios

Qual a diferença das duas versões?

Qual o comportamento da pilha
se `v[0..n-1]` é crescente?

Ver exercícios em

http://www.ime.usp.br/~pf/algoritmos/aulas/quick.html