Melhores momentos

AULA 19

Resumo

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

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;

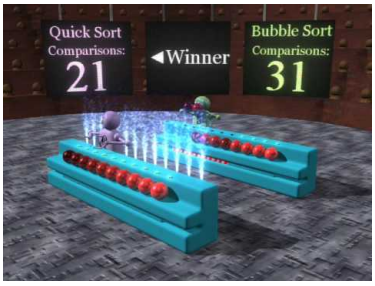
Conquistar: os subproblemas são resolvidos recursivamente e subproblemas "pequenos" são resolvidos diretamente;

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

Exemplo: ordenação por intercalação (`mergeSort`).

AULA 20

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

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

Entra:

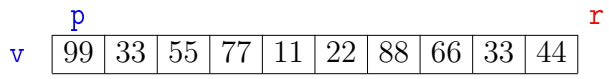
	p									r
v	99	33	55	77	11	22	88	66	33	44

Problema da separação

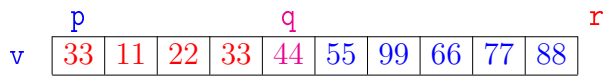
Problema: Rearranjar um dado vetor $v[p\,..\,r-1]$ e devolver um índice q, $p \le q < r$, tais que

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

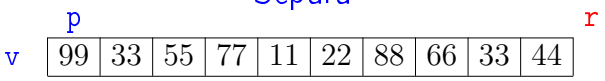
Entra:



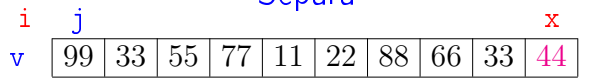
Sai:



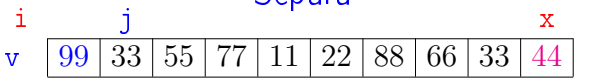
Separa



Separa



Separa



Separa

Separa

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

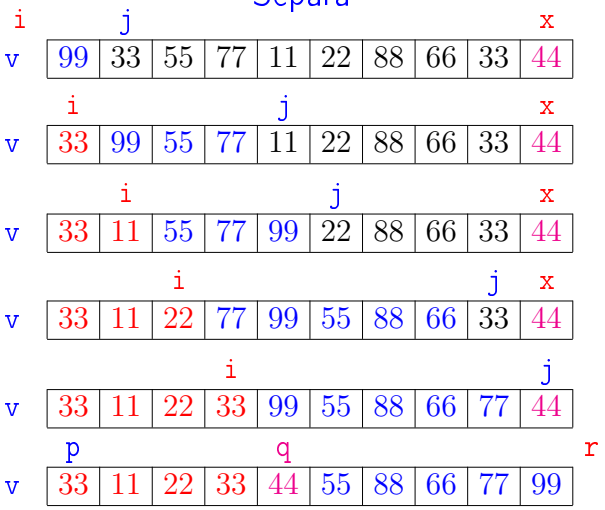
	i				j					x
v	33	99	55	77	11	22	88	66	33	44

		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



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.

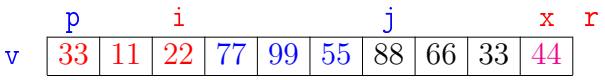
```
int separa (int p, int r, int v[]) {
1   int i = p-1, j, x = v[r-1], t;

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) $v[p\,..\,i] \le x < v[i+1\,..\,j-1]$



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	=	?
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 $n := r - p$?

linha		consumo de todas as execuções da linha
1	=	1
2	=	$n + 1$
3	=	n
4	$\le$	n
5	=	1
total	$\le$	$3n + 3$ $\quad = O(n)$

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,

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

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

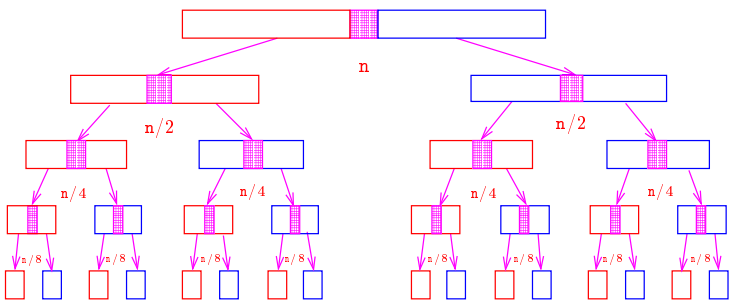
Quicksort

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

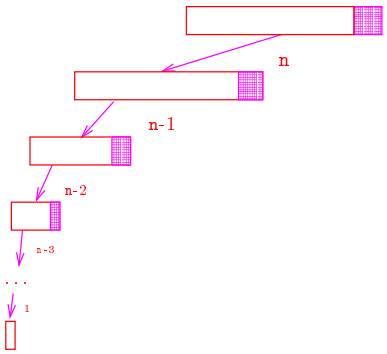
O consumo de tempo em cada nível da recursão é proporcional a n.

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	$\approx n/4 + n/4 + n/4 + n/4$
$\cdots$	$\cdots$
$\lg n$	$\approx 1+1+1+1 \cdots + 1+1+1+1$
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	$\approx n-3$
$\cdots$	$\cdots$
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 (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 $n := r - p$?

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 $n := r - p$?

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

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

Recorrência: outra versão

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

$T(0) = O(1)$
$T(1) = O(1)$
$T(n) = T(k) + T(n-k-1) + O(n)$ para $n = 2, 3, 4, \ldots$

Recorrência: outra versão

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

$T(0) = O(1)$
$T(1) = O(1)$
$T(n) = T(k) + T(n-k-1) + O(n)$ para $n = 2, 3, 4, \ldots$

Recorrência grosseira:

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

$T(n)$ é $O(???)$.

Recorrência: outra versão

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

$T(0) = O(1)$
$T(1) = O(1)$
$T(n) = T(k) + T(n-k-1) + O(n)$ para $n = 2, 3, 4, \ldots$

Recorrência grosseira:

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

$T(n)$ é $O(n^2)$.

Demonstração: ...

Recorrência cuidadosa: ...

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

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

$T(n)$ é $O(n^2)$.

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 e só 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

k-ésimo menor elemento

x é o k-**ésimo menor elemento** de um vetor v[0 . . n−1] se em um rearranjo crescente de v, x é o valor na posição v[k-1].

Problema: encontrar k-ésimo menor elemento de um vetor v[0 . . n-1], supondo $1 \leq k \leq n$.

Exempo: **33** é o 4o. menor elemento de:

	0									n
v	99	33	55	77	11	22	88	66	33	44

k-ésimo menor elemento

x é o **k-ésimo menor elemento** de um vetor v[0..n−1] se em um rearranjo crescente de v, x é o valor na posição v[k-1].

Problema: encontrar k-ésimo menor elemento de um vetor v[0 . . n-1], supondo $1 \le k \le n$.

Exemplo: **33** é o 4o. menor elemento de:

	0									n
v	99	33	55	77	11	22	88	66	33	44

pois: no vetor ordem crescente temos

	0									n
v	11	22	33	**33**	44	55	66	77	88	99

Solução inspirada em selecao()

Algoritmo baseado em ordenação por seleção.
Ao final o k-ésimo menor elemento está em v[k-1].

```
void
kEsimo (int k, int n, int v[]) {
   int i, j, min, x;
1  for (i = 0; i < k; i++) {
2     min = i;
3     for (j = i+1; j < n; j++)
4        if (v[j] < v[min]) min = j;
5     x=v[i]; v[i]=v[min]; v[min]=x;
   }
}
```

Invariantes

Relações **invariantes** chaves dizem que em /*A*/ vale que:

♡ (i0) v[0..i−1] é crescente e
v[0..i−1] ≤ v[i..n−1]

	0				i						n
	10	20	38	44	75	50	55	99	85	50	60

Supondo que a invariantes valem.
Correção do algoritmo é evidente.

No início da última iteração das linhas 1–5 tem-se que i = k.
Da invariante conclui-se que v[0..k−1] é crescente.
e que v[k−1] ≤ v[k..n−1].

Mais invariantes

Na linha 1 vale que: (i1) v[i] ≤ v[i+1..n−1];
Na linha 3 vale que: (i2) v[min] ≤ v[i..j−1]

	0				i	min			j		n
	10	20	38	44	75	50	55	99	85	50	60

invariantes (i1),(i2)
+ condição de parada do for da linha 3
+ troca linha 5 ⇒ validade (i0)

Verifique!

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha	todas as execuções da linha		
1	=	k + 1	= O(k)
2	=	k	= O(k)
3	=	n+(n−1)+···+(n−k+1)	= O(kn)
4	=	(n−1)+(n−2)+···+(n−k+1)	= O(kn)
5	=	k	= O(k)
total	=	O(2kn + 3k)	= O(kn)

Conclusão

O consumo de tempo do algoritmo kEsimo no pior caso e no no melhor caso é proporcional a kn.

O consumo de tempo do algoritmo kEsimo é O(kn).

Solução inspirada em quickSort()

Ao final o k-ésimo menor elemento está em v[k-1].
Primeira chamada: kEsimo(k,0,n,v);.

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

Consumo de tempo?

kEsimo: versão iterativa

```
void kEsimo (int k, int n, int v[]){
1   int p = 0;
2   int r = n;
3   int q = separa(p,r,v);
4   while (q != k-1) {
5       if (q >= k) r = q;
6       if (q <  k) p = q + 1;
7       q = separa(p, r, v);
    }
}
```

Exercícios

Qual o consumo de tempo no melhor caso do algoritmo kEsimo inspirado em quickSort?

Qual o consumo de tempo no pior caso do algoritmo kEsimo inspirado em quickSort?

Tente determinar experimentalmente o consumo de tempo do algoritmo kEsimo inspirado em quickSort.

Sob *hipóteses razoáveis* é possível mostrar que o consumo de tempo esperado do algoritmo kEsimo inspirado no quickSort é proporcional a n.