- CLRS: Cormen, Leiserson, Rivest, Stein

  "Uma base sólida de conhecimento e técnica de algoritmos
  é uma das características que separa
  o programador experiente do aprendiz.
  Com a moderna tecnologia de computação,
  você pode realizar algumas tarefas
  sem saber muito sobre algoritmos,
  mas com um boa base em algoritmos você pode fazer muito,
  muito mais."

Exercício

Re-escreva o código abaixo em linguagem C

ORDINS (A, n)

1 para $j \leftarrow 2$ até n faça
2 $\quad chave \leftarrow A[j]$
3 $\quad i \leftarrow j - 1$
4 $\quad$ enquanto $i \geq 1$ e $A[i] > chave$ faça
5 $\quad\quad A[i+1] \leftarrow A[i]$
6 $\quad\quad i \leftarrow i - 1$
7 $\quad A[i+1] \leftarrow chave$

Solução:

```
void OrdIns (int A[], int n) {
   int i, j, ch;
   for (j = 2; j <= n; j++) {
      ch = A[j];
      i = j-1;
      while (i >= 1 && A[i] > ch) {
         A[i+1] = A[i];
         i--;
      }
      A[i+1] = ch;
   }
}
```


O que é AA?

Problema: Rearranjar um vetor em ordem crescente

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$

22	33	33	33	44	55	11	99	22	55	77

11	22	22	33	33	33	44	55	55	77	99

Algoritmo: Rearranja $A[1\,..\,n]$ em ordem crescente

ORDENA-POR-INSERÇÃO (A, n)

```
1  para j ← 2 até n faça
2      chave ← A[j]
3      i ← j − 1
4      enquanto i ≥ 1 e A[i] > chave faça
5          A[i + 1] ← A[i]
6          i ← i − 1
7      A[i + 1] ← chave
```

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

Análise da correção: O algoritmo faz o que prometeu?

- Invariante: no início de cada iteração, $A[1\,..\,j-1]$ é crescente
- Se vale na última iteração, o algoritmo está correto!

ORDENA-POR-INSERÇÃO (A, n)

```
1  para j ← 2 até (*) n faça
2      chave ← A[j]
3      i ← j − 1
4      enquanto i ≥ 1 e A[i] > chave faça
5          A[i + 1] ← A[i]
6          i ← i − 1
7      A[i + 1] ← chave
```

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

- vale na primeira iteração
- se vale em uma iteração, vale na seguinte

Análise do desempenho: Quanto tempo consome?

- Regra de três não funciona!
- Suponha 1 segundo por linha

linha	total de segundos		
1	$= n$		
2	$= n-1$		
3	$= n-1$		
4	$\leq 2+3+\cdots+n$	$=$	$(n-1)(n+2)/2$
5	$\leq 1+2+\cdots+(n-1)$	$=$	$n(n-1)/2$
6	$\leq 1+2+\cdots+(n-1)$	$=$	$n(n-1)/2$
7	$= n-1$		
total	$\leq \frac{3}{2}n^2+\frac{7}{2}n-4$	segundos	

- Algoritmo consome $\leq \frac{3}{2}n^2 + \frac{7}{2}n - 4$ segundos
- $\frac{3}{2}$ é "pra valer"?

  Não, pois depende do computador
- n^2 é "pra valer"?

  Sim, pois só depende do algoritmo
- Queremos um resultado que só dependa do algoritmo

Outra tática:

- número de comparações "$A[i] > chave$"

  $\leq 2 + 3 + \cdots + n = \frac{1}{2}(n-1)(n+2) \leq \frac{1}{2}n^2 + \frac{1}{2}n - 1$
- $\frac{1}{2}$ é "pra valer"?

  Não, pois depende do computador
- n^2 é "pra valer"?

  Sim, pois só depende do algoritmo
- Queremos resultado que só dependa do algoritmo

- "n^2" apareceu novamente
- "n^2" é informação valiosa
- mostra evolução do tempo com n:

n	n^2
$2n$	$4n^2$
$10n$	$100n^2$

O que é AA?

- Problema e instâncias
- Instâncias do problema
- Tamanho de uma instância
- Algoritmo
- Análise da correção do algoritmo
- Análise do desempenho (velocidades) do algoritmo
- Desempenho em função do tamanho das instâncias
- Pior caso
- Evolução do consumo de tempo em função de n
- Independente do computador e da implementação

Medição do consumo de tempo

Quem é maior: n^2 ou $2n + 20$?
Qual cresce mais rápido?

n	n^2	$2n + 20$
1	1	22
2	4	24
5	25	30
10	100	40
20	400	60
50	2500	120
100	10000	220

Só interessa n grande

Quem cresce mais rápido: $10n^2$ ou 2^n?

n	$10n^2$	2^n
1	10	1
2	40	4
5	250	32
10	1000	1024
20	4000	1048576
30	9000	1073741824
35	12250	34359738368
40	16000	1099511630000

Só interessa n grande
análise assintótica

Notação O: comparação assintótica de funções

- funções que levam números positivos em números positivos
- comparação assintótica grosseira de funções $f(n)$ e $g(n)$
- algo com sabor de $f(n)$ "$\leq$" $g(n)$
- despreze valores pequenos de n: $n \to \infty$
- despreze constantes multiplicativas: $100\,n^2$ "$\leq$" n^2
- conseqüência:
  despreze termos de ordem inferior: $n^2 + 10n + 10$ "$\leq$" $2\,n^2$

$f(n)$ e $g(n)$ levam números ≥ 0 em números ≥ 0

Definição: $f = O(g)$ significa
existe constante $c > 0$ tal que $f(n) \leq c \cdot g(n)$
para todo n suficientemente grande

constante = não dependende de n

seria mais correto: $f \in O(g)$

Exemplo 1

$n^2 + 10n \;=\; O(n^2)$

Prova:

- $n^2 + 10n \leq 2n^2$ para $n \geq 0$
- pois $n^2 + 10 \cdot n \leq n^2 + n \cdot n = n^2 + n^2$

Exemplo 2

$9n + 9 = O(n^2)$

Prova:

- $9n + 9 \leq 2n^2$ para $n \geq 9$
- pois $9 \cdot n + 9 \leq n \cdot n + n^2$

Outra prova:

- $9n + 9 \leq 18n^2$ para $n \geq 1$
- pois $9n + 9 \leq 9n \cdot n + 9 = 9n^2 + 9 \cdot n^2$

Exemplo 3

$n^2 \;=\; O(9n + 9)$

Não é verdade!
Prova:

- suponha que existem $c > 0$ e N tais que
  $n^2 \leq c \cdot (9n + 9)$ para todo $n \geq N$
- então $n^2 \leq c \cdot (9n + 9n) = 18\,c\,n$ para todo $n \geq N$
- então $n \leq 18\,c$ para todo $n \geq N$
- absurdo

Exemplo 4

$\frac{1}{100}\,n^3 \;=\; O(n^2)$

Não é verdade! Prova:

- suponha que existem $c > 0$ e N tais que
  $\frac{1}{100}\,n^3 \leq c \cdot n^2$ para todo $n \geq N$
- então $n \leq 100 \cdot c$ para todo $n \geq N$
- absurdo

Exemplo 5

$n^2 = O(2^n)$

Prova:

- vou mostrar que $n^2 \leq 2^n$ para $n \geq 4$
- prova por indução
- base: $n = 4$
- passo: suponha $n \geq 4$
- observe que $2n + 1 \leq n^2$ para $n \geq 3$
  pois $n^2 - 2n - 1 = (n + 0.41)(n - 2.41)$
- logo, $(n+1)^2 = n^2 + 2n + 1 \leq n^2 + n^2 \leq 2^n + 2^n = 2^{n+1}$

Exemplo 6

$\log_2 n = O(n)$

Prova:

- $n < n^2 \leq 2^n$ quando $n \geq 4$
- melhor: $n \leq 2^n$ quando $n \geq 1$
- portanto $\log_2 n \leq n$ quando $n \geq 1$

A propósito,

- $\log_2 n = \dfrac{\log_3 n}{\log_3 2} = \dfrac{1}{\log_3 2} \log_3 n$

- $\log n = O(n)$

Reprise: algoritmo da ordenação por inserção

Análise do consumo do algoritmo ORDENA-POR-INSERÇÃO:

- o algoritmo consome $O(n^2)$ unidades de tempo
- a estimativa é justa:

  não é verdade que o algoritmo consome $O(n)$ unidades de tempo

  existem isntâncias de tamanho n que levam o algoritmo a consumir tempo proporcional a n^2

Novo problema: intercalação

Problema: Rearranjar um vetor em ordem crescente sabendo que o lado e o lado direito já são crescentes

Sei fazer em $O(n^2)$ unidades de tempo

Dá pra melhorar?

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

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

Algoritmo: Recebe $A[p\,..\,r]$ tal que $A[p\,..\,q]$ e $A[q{+}1\,..\,r]$ são crescentes. Rearranja o vetor todo em ordem crescente.

```
INTERCALA (A, p, q, r)
1   n1 ← q − p + 1
2   n2 ← r − q
3   crie vetores L[1 .. n1 + 1] e R[1 .. n2 + 1]
4   para i ← 1 até n1 faça L[i] ← A[p + i − 1]
5   para j ← 1 até n2 faça R[j] ← A[q + j]
6   L[n1 + 1] ← R[n2 + 1] ← ∞
⋮   ⋮
```



```
⋮   ⋮
7   i ← j ← 1
8   para k ← p até r faça
9     se L[i] ≤ R[j]
0       então A[k] ← L[i]
1             i ← i + 1
2       senão A[k] ← R[j]
3             j ← j + 1
```

Tamanho de uma instância: $n := r - p + 1$

Análise do desempenho: consume $O(n)$ unidades de tempo

Exercício 1

Escreva uma função recursiva que calcule o máximo de um vetor
Use notação CLR.

Solução:

```
▷ Devolve o valor de um elemento máximo de A[1 .. n].
▷ Supõe n ≥ 1.

MAX (A, n)
1  se n = 1
2     então devolva A[1]
3     senão x ← MAX (A, n − 1)
4           se x ≥ A[n]
5              então devolva x
6              senão devolva A[n]
```

Mergesort

Problema: Rearranjar um vetor em ordem crescente

Algoritmo: Rearranja $A[p..r]$ em ordem crescente

```
MERGE-SORT (A,p,r)
1  se p < r
2     então q ← ⌊(p+r)/2⌋
3           MERGE-SORT (A,p,q)
4           MERGE-SORT (A,q+1,r)
5           INTERCALA (A,p,q,r)
```

Método de divisão e conquista

Segredo da velocidade: INTERCALA é rápido

Parei aqui.

Desempenho: Quanto tempo MERGE-SORT consome?

- tamanho de instância: $n := r - p + 1$
- $T(n)$:= consumo de tempo para instância de tamanho n
- quero cota superior: $T(n) \leq$??
- se $n > 1$ então

$$T(n) \leq T\Big(\Big\lfloor \frac{n}{2} \Big\rfloor\Big) + T\Big(\Big\lceil \frac{n}{2} \Big\rceil\Big) + S(n)$$

  onde $S(n)$ é consumo de tempo de INTERCALA
- digamos que $S(n) \leq 10n$ para todo $n > 1$

- recorrência:

$$T(n) \leq \begin{cases} 3 & \text{se } n \leq 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 10\,n & \text{se } n > 1 \end{cases}$$

n	$T(n)$	
1	3	3
2	$3+3+10\cdot 2$	26
3	$3+26+10\cdot 3$	59
4	$26+26+10\cdot 4$	92
5	$26+59+10\cdot 5$	135

Simplifica: $n = 1, 2, 4, 8, 16, \ldots, 2^k, \ldots$

$$T(n) \leq \begin{array}{ll} 3 & \text{se } n = 1 \\ 2\,T(\frac{n}{2}) + 10\,n & \text{se } n > 1 \end{array}$$

Solução: $T(n) \leq 10\,n \log_2 n + 3n$

Prova, por indução:

- Base: se $n = 1$ então $T(n) = 3$ e $10n \log_2 n + 3n = 3$
- Passo: se $n > 1$ então

$$\begin{aligned} T(n) &\leq 2\,T(\frac{n}{2}) + 10\,n \\ &\leq 2\left(10\frac{n}{2}\log_2\frac{n}{2} + 3\frac{n}{2}\right) + 10\,n \\ &= 10\,n(\log_2 n - 1) + 3n + 10\,n \\ &= 10\,n\log_2 n - 10n + 3n + 10\,n \\ &= 10\,n\log_2 n + 3n \end{aligned}$$

Conclusão: $T(n) = O(n \log n)$

Prova: ...

Como adivinhei "$10\,n\log_2 n + 3n$"?

"Desenrole" a recorrência