

# Melhores momentos

AULA PASSADA

# Recorrências

$$T(1) = 1$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n + 5 \quad \text{para } n = 2, 3, 4, 5, \dots$$

$n$	1	2	3	4	5	6	7	8	9	10
$n \lceil \lg n \rceil$	0	2	6	8	15	18	21	24	36	40
$T(n)$	1	19	43	67	97	127	157	187	223	259
$20 n \lfloor \lg n \rfloor$	0	40	60	160	200	240	280	480	540	600

**Conclusão:**

$$T(n) \text{ é } \Theta(n \lg n).$$

# Conclusão da conclusão

O consumo de tempo do **MERGE-SORT** é  $\Theta(n \lg n)$   
no pior caso.

**Exercício.** Mostre que:

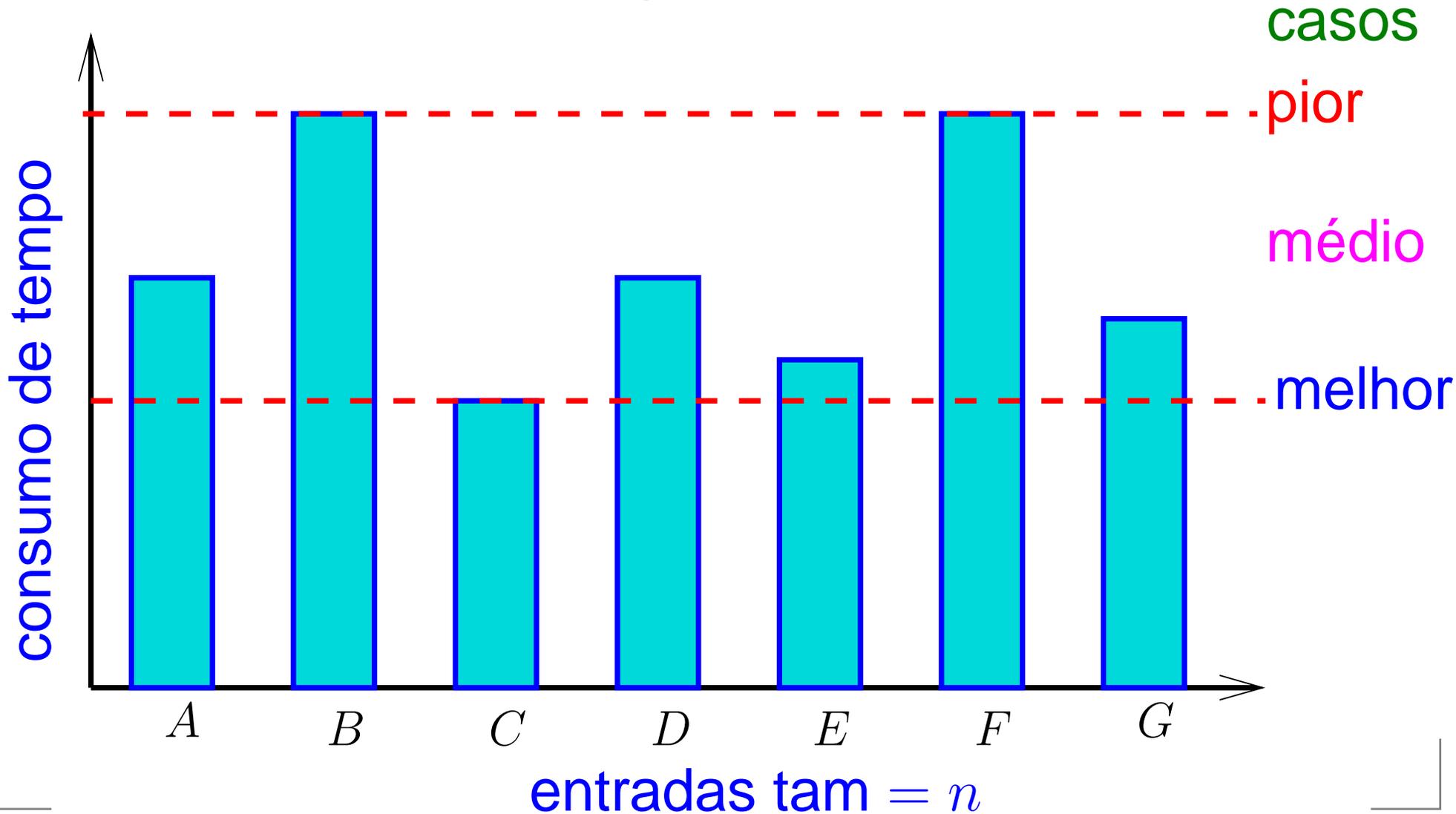
O consumo de tempo do **MERGE-SORT** é  $\Theta(n \lg n)$ .

Hmmmm... Qual a diferença entra as duas afirmações?

# Lembra?

Número mínimo, médio ou máximo?

Melhor caso, caso médio, pior caso?



# AULA 5

# Recorrências com notação $O$

CLRS 4.1–4.2

AU 3.9, 3.11

# Classe $O$ da solução de uma recorrência

Não faço questão de solução **exata**: basta solução **assintótica** (em notação  $O$ , se possível  $\Theta$ )

# Classe $O$ da solução de uma recorrência

Não faço questão de solução **exata**: basta solução **assintótica** (em notação  $O$ , se possível  $\Theta$ )

Exemplo:

$$G(1) = 1$$

$$G(n) = 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16, \dots$$

**Solução exata:**  $G(n) = 7n \lg n + 3n - 2$

**Solução assintótica:**  $G(n) = O(n \lg n) \quad (G(n) = \Theta(n \lg n))$

# Classe $O$ da solução de uma recorrência

Não faço questão de solução **exata**: basta solução **assintótica** (em notação  $O$ , se possível  $\Theta$ )

Exemplo:

$$G(1) = 1$$

$$G(n) = 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16, \dots$$

**Solução exata:**  $G(n) = 7n \lg n + 3n - 2$

**Solução assintótica:**  $G(n) = O(n \lg n)$  ( $G(n) = \Theta(n \lg n)$ )

Em geral, é **mais fácil** obter e provar solução assintótica que solução exata

# Dica prática (sem prova)

A solução da recorrência

$$T(1) = 1$$

$$T(n) = 2T(\lfloor n/2 \rfloor) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \dots$$

está na **mesma classe**  $\Theta$  que a solução de

$$T'(1) = 1$$

$$T'(n) = 2T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \dots$$

# Dica prática (sem prova)

A solução da recorrência

$$T(1) = 1$$

$$T(n) = 2T(\lfloor n/2 \rfloor) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \dots$$

está na **mesma classe**  $\Theta$  que a solução de

$$T'(1) = 1$$

$$T'(n) = 2T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \dots$$

e na **mesma classe**  $\Theta$  que a solução de

$$T''(4) = 10$$

$$T''(n) = 2T''(n/2) + n \quad \text{para } n = 2^3, 2^4, 2^5, \dots$$

# Recorrências com $O$ do lado direito

A “recorrência”

$$T(n) = 2T(n/2) + O(n)$$

**representa** todas as recorrências da forma  $T(n) = 2T(n/2) + f(n)$  em que  $f(n)$  é  $O(n)$ .

# Recorrências com $O$ do lado direito

A “recorrência”

$$T(n) = 2T(n/2) + O(n)$$

**representa** todas as recorrências da forma  $T(n) = 2T(n/2) + f(n)$  em que  $f(n)$  é  $O(n)$ .

**Melhor:** representa todas as recorrências do tipo

$$T'(n) \leq a \quad \text{para } n = k, k + 1, \dots, 2k - 1$$

$$T'(n) \leq 2T'(\lfloor n/2 \rfloor) + bn \quad \text{para } n \geq 2k$$

quaisquer que sejam  $a, b > 0$  e  $k > 0$   
(poderíamos tomar  $n_0 = 1$ ; veja ex ??.)

# Recorrências com $O$ do lado direito

A “recorrência”

$$T(n) = 2T(n/2) + O(n)$$

**também representa** todas as recorrências do tipo

$$T''(n) = a \quad \text{para } n = 2^{k-1}$$

$$T''(n) \leq 2T''(n/2) + b n \quad \text{para } n = 2^k, 2^{k+1}, \dots$$

quaisquer que sejam  $a, b > 0$  e  $k > 0$

# Recorrências com $O$ do lado direito

A “recorrência”

$$T(n) = 2T(n/2) + O(n)$$

**também representa** todas as recorrências do tipo

$$T''(n) = a \quad \text{para } n = 2^{k-1}$$

$$T''(n) \leq 2T''(n/2) + bn \quad \text{para } n = 2^k, 2^{k+1}, \dots$$

quaisquer que sejam  $a, b > 0$  e  $k > 0$

As **soluções exatas** vão depender de  $a, b, k$ ;  
mas todas estarão na **mesma** classe  $O$  (no caso, em  $O(n \lg n)$ )

# Exemplo com $O$

## Recorrência com $O$ do lado direito

Na análise do consumo de tempo do **MERGE-SORT** tínhamos:

$T(n)$  := consumo de tempo máximo quando  $n = r - p + 1$

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

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) \text{ para } n = 2, 3, 4, \dots$$

**Solução assintótica:**  $T(n)$  é  $O(n \lg n)$ .

# Exemplo com $\Theta$

## Recorrências com $\Theta$ do lado direito

Na análise do consumo de tempo do **MERGE-SORT** poderíamos ter feito:

$T(n) :=$  consumo de tempo quando  $n = r - p + 1$

$$T(1) = \Theta(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \text{ para } n = 2, 3, 4, \dots$$

**Solução assintótica:**  $T(n)$  é  $\Theta(n \lg n)$ .

# Dica prática

Suponha dada uma “recorrência” como

$$T(n) = 2T(n/2) + O(n).$$

**Para ter uma idéia** da classe  $O$  a que  $T$  pertence, resolva a recorrência

$$T''(2^0) = 1$$

$$T''(n) \leq 2T''(n/2) + n \quad \text{para } n = 2^1, 2^2, 2^3, \dots$$

# Mais dicas práticas

recorrência	condição	solução
$T(n) = T(n-1) + 4n^3$		$\Theta(n^{3+1})$
$T(n) = 6T(n-1) + 4n^3$		$\Theta(6^n)$
$T(n) = aT(n/5) + 4n^3$	$a < 5^3$	$\Theta(n^3)$
$T(n) = aT(n/5) + 4n^3$	$a = 5^3$	$\Theta(n^3 \log n)$
$T(n) = aT(n/5) + 4n^3$	$a > 5^3$	$\Theta(n^{\log_5 a})$

Veja AU, sec 3.11, p.151

# Mais dicas práticas (continuação)

No lugar de

- “ $n/5$ ”, posso escrever “ $\lfloor n/5 \rfloor$ ” ou “ $\lceil n/5 \rceil$ ”
- “5”, posso escrever qualquer  $b > 1$
- “4”, posso escrever qualquer número real
- “ $4n^3$ ” posso escrever qualquer polinômio de grau 3
- “3”, posso escrever qualquer inteiro  $k \geq 0$
- “6”, posso escrever qualquer número  $a > 1$

# Mais dicas práticas ainda

A mesma coisa, escrita de maneira um pouco diferente (Master Theorem, CLRS, sec. 4.3, p.73):

Suponha  $T(n) = aT(n/5) + f(n)$  para algum  $a \geq 1$ . Então, em geral,

$$\text{se } f(n) = O(n^{\log_5 a - \epsilon}) \quad \text{então } T(n) = \Theta(n^{\log_5 a})$$

$$\text{se } f(n) = \Theta(n^{\log_5 a}) \quad \text{então } T(n) = \Theta(n^{\log_5 a} \lg n)$$

$$\text{se } f(n) = \Omega(n^{\log_5 a + \epsilon}) \quad \text{então } T(n) = \Theta(f(n))$$

para qualquer  $\epsilon > 0$

# Teorema Bambambã

Teorema Mestre (Master Theorem, CLRS, sec. 4.3, p.73):

Suponha

$$T(n) = aT(n/b) + f(n)$$

para algum  $a \geq 1$  e  $b > 1$  e onde  $n/b$  significa  $\lceil n/b \rceil$  ou  $\lfloor n/b \rfloor$ .  
Então, em geral,

$$\text{se } f(n) = O(n^{\log_b a - \epsilon}) \quad \text{então } T(n) = \Theta(n^{\log_b a})$$

$$\text{se } f(n) = \Theta(n^{\log_b a}) \quad \text{então } T(n) = \Theta(n^{\log_b a} \lg n)$$

$$\text{se } f(n) = \Omega(n^{\log_b a + \epsilon}) \quad \text{então } T(n) = \Theta(f(n))$$

para qualquer  $\epsilon > 0$ .

# Exercícios

## Exercício 9.A

A que classe  $\mathcal{O}$  pertencem as soluções de recorrência do tipo  $T(n) = T(n/3) + \mathcal{O}(1)$ ?

## Exercício 9.B

Seja  $T$  a função definida pela recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(\lfloor n/2 \rfloor) + n \quad \text{para } n = 2, 3, 4, 5, \dots \end{aligned}$$

A que ordem  $\Theta$  pertence  $T$ ?

## Exercício 9.C [CLRS 4.2-1]

Seja  $T$  a função definida pela recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T(\lfloor n/2 \rfloor) + n \quad \text{para } n = 2, 3, 4, 5, \dots \end{aligned}$$

A partir da árvore da recorrência, adivinhe a que classe  $\Theta$  pertence  $T(n)$ . Prove a delimitação pelo método da substituição.

# Mais exercícios

## Exercício 9.D

Resolva a recorrência

$$\begin{aligned}T(1) &= 1 \\T(n) &= 2T(\lceil n/2 \rceil) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \dots\end{aligned}$$

## Exercício 9.E

Resolva a “recorrência”  $T(n) = T(n - 2) + O(n)$ .

## Exercício 9.F

Resolva a “recorrência”  $T(n) = 5T(n - 1) + O(n)$ .

# Técnicas em projeto de algoritmos

Programming Pearls: Algorithm Design  
Techniques,  
Jon Bentley, Addison-Wesley, 1986

# Segmento de soma máxima

Um **segmento** de um vetor  $A[1..n]$  é qualquer subvetor da forma  $A[e..d]$ .

**Problema:** Dado um vetor  $A[1..n]$  de números inteiros, determinar um segmento  $A[e..d]$  de **soma máxima**.

Entra:

	1								$n$	
$A$	31	-41	59	26	-53	58	97	-93	-23	84

# Segmento de soma máxima

Um **segmento** de um vetor  $A[1..n]$  é qualquer subvetor da forma  $A[e..d]$ .

**Problema:** Dado um vetor  $A[1..n]$  de números inteiros, determinar um segmento  $A[e..d]$  de **soma máxima**.

Entra:

	1								$n$	
A	31	-41	59	26	-53	58	97	-93	-23	84

Sai:

	1		<b>3</b>				<b>7</b>			$n$
A	31	-41	<b>59</b>	<b>26</b>	<b>-53</b>	<b>58</b>	<b>97</b>	-93	-23	84

$A[e..d] = A[3..7]$  é segmento de soma máxima.

$A[3..7]$  tem soma 187.

# Segmento de soma máxima

**Problema (versão simplificada):** Determinar a **soma máxima** de um segmento de um dado vetor  $A[1..n]$ .

Entra:

	1								$n$	
$A$	31	-41	59	26	-53	58	97	-93	-23	84

Sai:

	1		3			7				$n$
$A$	31	-41	59	26	-53	58	97	-93	-23	84

A soma máxima é 187.

# Algoritmo café-com-leite

Algoritmo determina um segmento de soma máxima de  $A[1..n]$ .

**SEG-MAX-3** ( $A, n$ )

```
1  somamax ← 0
2   $e \leftarrow 0$     $d \leftarrow -1$    ▷  $A[e..d]$  é vazio
3  para  $i \leftarrow 1$  até  $n$  faça
4      para  $f \leftarrow i$  até  $n$  faça
5           $soma \leftarrow 0$ 
6          para  $k \leftarrow i$  até  $f$  faça
7               $soma \leftarrow soma + A[k]$ 
8          se  $soma > somamax$  então
9               $somamax \leftarrow soma$     $e \leftarrow i$     $d \leftarrow f$ 
10 devolva  $e, d$  e somamax
```

# Correção

Relação **invariante** chave:

(i0) na linha 3 vale que:  $A[e..d]$  é um segmento de soma máxima com  $e < i$ .

	<i>e</i>		<i>i</i>			<i>d</i>			<i>n</i>	
<i>A</i>	31	-41	59	26	-53	58	97	-93	-23	84

# Correção

Relação **invariante** chave:

(i0) na linha 3 vale que:  $A[e..d]$  é um segmento de soma máxima com  $e < i$ .

	$e$		$i$			$d$			$n$	
A	31	-41	59	26	-53	58	97	-93	-23	84

Mais relações **invariantes**:

(i1) na linha 3 vale que:

$$somamax = A[e] + A[e + 1] + A[e + 2] + \dots + A[d];$$

(i2) na linha 6 vale que:

$$soma = A[i] + A[i + 1] + A[i + 2] + \dots + A[k - 1].$$

# 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

---

$$1-2 \quad = \quad 2 \quad \quad \quad = \Theta(1)$$

$$3 \quad = \quad n + 1 \quad \quad \quad = \Theta(n)$$

$$4 \quad = \quad (n + 1) + n + (n - 1) + \dots + 1 \quad = \Theta(n^2)$$

$$5 \quad = \quad n + (n - 1) + \dots + 1 \quad = \Theta(n^2)$$

$$6 \quad = \quad (2 + \dots + (n + 1)) + (2 + \dots + n) + \dots + 2 \quad = \Theta(n^3)$$

$$7 \quad = \quad (1 + \dots + n) + (1 + \dots + (n - 1)) + \dots + 1 \quad = \Theta(n^3)$$

$$8 \quad = \quad n + (n - 1) + (n - 2) + \dots + 1 \quad = \Theta(n^2)$$

$$9 \quad \leq \quad n + (n - 1) + (n - 2) + \dots + 1 \quad = O(n^2)$$

$$10 \quad = \quad 1 \quad \quad \quad = \Theta(1)$$

---

$$\text{total} \quad = \quad \Theta(2n^3 + 3n^2 + n + 2) + O(n^2) \quad = \Theta(n^3)$$

# Algoritmo arroz-com-feijão

Algoritmo determina um segmento de soma máxima de  $A[1..n]$ .

**SEG-MAX-2** ( $A, n$ )

1  $somamax \leftarrow 0$

2  $e \leftarrow 0$     $d \leftarrow -1$     $\triangleright A[e..d]$  é vazio

3 **para**  $i \leftarrow 1$  **até**  $n$  **faça**

4      $soma \leftarrow 0$

5     **para**  $f \leftarrow i$  **até**  $n$  **faça**

6          $soma \leftarrow soma + A[f]$

7         **se**  $soma > somamax$  **então**

8              $somamax \leftarrow soma$     $e \leftarrow i$     $d \leftarrow f$

9 **devolva**  $e, d$  **e**  $somamax$

# Correção

Relação **invariante** chave:

(i0) na linha 3 vale que:  $A[e..d]$  é um segmento de soma máxima com  $e < i$ .

	$e$					$d$	$i$		$n$	
A	31	-41	59	26	-53	58	97	-93	-23	84

Mais relações invariante:

(i1) na linha 3 vale que:

$$somamax = A[e] + A[e + 1] + A[e + 2] + \dots + A[d];$$

(i2) na linha 5 vale que:

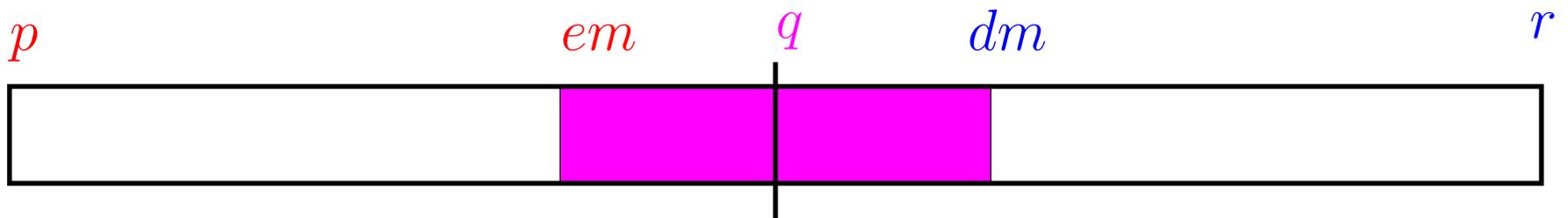
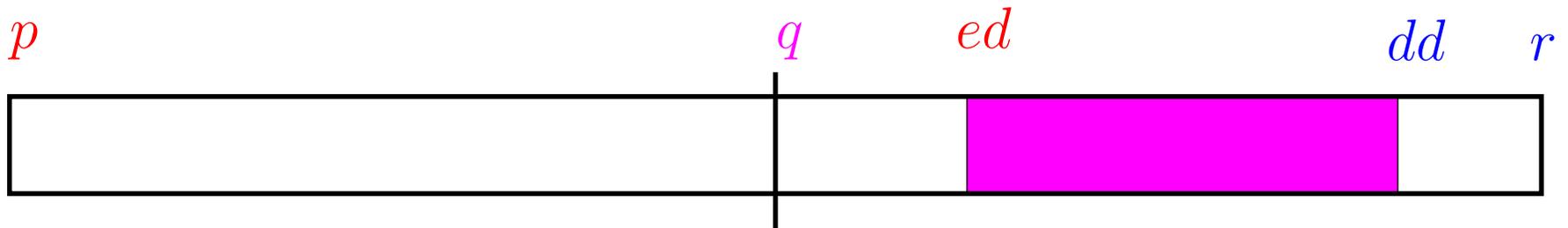
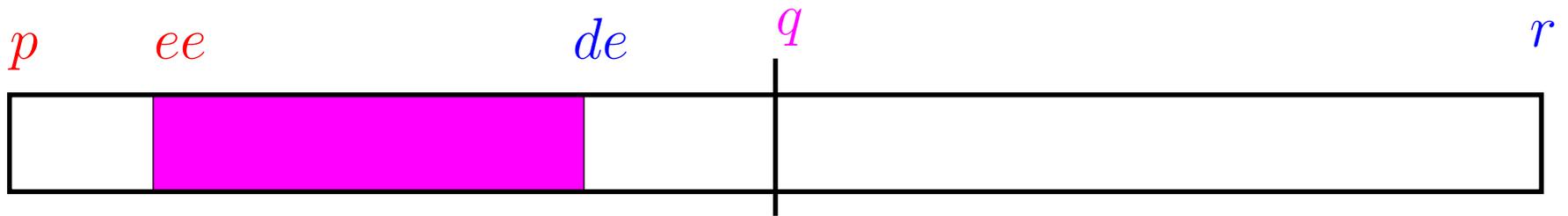
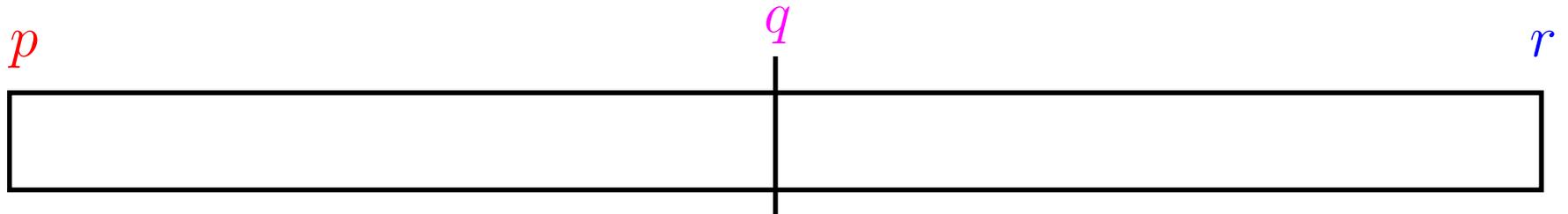
$$soma = A[i] + A[i + 1] + A[i + 2] + \dots + A[f - 1];$$

# 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	
1-2	= 2	= $\Theta(1)$
3	= $n + 1$	= $\Theta(n)$
4	= $n$	= $\Theta(n)$
5	= $(n + 1) + n + \dots + 2$	= $\Theta(n^2)$
6	= $n + (n - 1) + \dots + 1$	= $\Theta(n^2)$
7	= $n + (n - 1) + \dots + 1$	= $\Theta(n^2)$
8	$\leq n + (n - 1) + \dots + 1$	= $O(n^2)$
9	= 1	= $\Theta(1)$
<b>total</b>	= $\Theta(3n^2 + 2n + 2) + O(n^2)$	= $\Theta(n^2)$

# Solução de divisão-e-conquista



# Algoritmo de divisão-e-conquista

Algoritmo determina soma máxima de um seg. de  $A[p..r]$ .

**SEG-MAX-DC** ( $A, p, r$ )

```
1  se  $p = r$  então devolva  $\max(0, A[p])$ 
2   $q \leftarrow \lfloor (p + r) / 2 \rfloor$ 
3   $maxesq \leftarrow \text{SEG-MAX-DC}(A, p, q)$ 
4   $maxdir \leftarrow \text{SEG-MAX-DC}(A, q + 1, r)$ 
5   $max2esq \leftarrow soma \leftarrow A[q]$ 
6  para  $i \leftarrow q - 1$  decrescendo até  $p$  faça
7       $soma \leftarrow soma + A[i]$ 
8       $max2esq \leftarrow \max(max2esq, soma)$ 
9   $max2dir \leftarrow soma \leftarrow A[q + 1]$ 
10 para  $f \leftarrow q + 2$  até  $r$  faça
11      $soma \leftarrow soma + A[f]$ 
12      $max2dir \leftarrow \max(max2dir, soma)$ 
13  $maxcruz \leftarrow max2esq + max2dir$ 
14 devolva  $\max(maxesq, maxcruz, maxdir)$ 
```

# Correção

Verifique que:

- $maxesq$  é a soma máxima de um segmento de  $A[p..q]$ ;
- $maxdir$  é a soma máxima de um segmento de  $A[q+1..r]$ ; e
- $maxcruz$  é a soma máximo de um segmento da forma  $A[i..f]$  com  $i \leq q$  e  $q+1 \leq f$ .

Conclua que o algoritmo devolve a soma máxima de um segmento de  $A[p..r]$ .

# 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	
1-2	= 2	= $\Theta(1)$
3	= $T(\lceil \frac{n}{2} \rceil)$	= $T(\lceil \frac{n}{2} \rceil)$
4	= $T(\lfloor \frac{n}{2} \rfloor)$	= $T(\lfloor \frac{n}{2} \rfloor)$
5	= 1	= $\Theta(1)$
6	= $\lceil \frac{n}{2} \rceil + 1$	= $\Theta(n)$
7-8	= $\lceil \frac{n}{2} \rceil$	= $\Theta(n)$
9	= 1	= $\Theta(1)$
10	= $\lfloor \frac{n}{2} \rfloor + 1$	= $\Theta(n)$
11-12	= $\lfloor \frac{n}{2} \rfloor$	= $\Theta(n)$
13-14	= 2	= $\Theta(1)$
<b>total</b>	=	<b><math>T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(4n + 4)</math></b>

# Consumo de tempo

$T(n)$  := consumo de tempo quando  $n = r - p + 1$

Na análise do consumo de tempo do **SEG-MAX-DC** chegamos a (já manjada) **recorrências com  $\Theta$  do lado direito**:

$$T(1) = \Theta(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \text{ para } n = 2, 3, 4, \dots$$

**Solução assintótica:**  $T(n)$  é  $\Theta(n \lg n)$ .

# Algoritmo ingênuo

**SEG-MAX-INGÊNUO** ( $A, n$ )

```
1  somamax ← 0   e ← 0   d ← -1   ▷  $A[e..d]$  é vazio
2  soma ← 0   i ← 1   f ← 0   ▷  $A[i..f]$  é vazio
3  enquanto f < n faça
4      f ← f + 1
5      soma ← 0   i ← 1   f ← 0   ▷  $A[i..f]$  é vazio
6      somaf ← 0
7      para k ← f até 1 faça
8          somaf ← somaf +  $A[k]$ 
9          se somaf > soma
10             então soma ← somaf   i ← k
11     se soma > somamax
12         então somamax ← soma   e ← i   d ← f
13     devolva e, d e somamax
```

# Correção

A correção do algoritmo se apóia nas relações invariantes a seguir.

Relação **invariante** chave:

(i0) na linha 3 vale que:  $A[e..d]$  é **segmento de soma máxima** com  $d \leq f$ .

			<i>e</i>	<i>d</i>	<i>f</i>				<i>n</i>	
<i>A</i>	31	-41	59	26	-53	58	97	-93	-23	84

Mais uma relação **invariante**:

(i1) na linha 3 vale que:

$$somamax = A[e] + A[e + 1] + A[e + 2] + \dots + A[d].$$

# Mais relações invariantes

As relações invariantes a seguir passam meio despercebidas.

Na linha 3 vale que:

(i2)  $A[i..f]$  é segmento de soma máxima com termino em  $f$ ;

(i3)  $soma = A[i] + A[i + 1] + A[i + 2] + \dots + A[f]$ ;

(i4) para  $k = i, i + 1, \dots, f$ , vale que

$$A[i] + A[i + 1] + \dots + A[k] \geq 0;$$

(i5) para  $k = 1, 2, \dots, i - 1$ , vale que

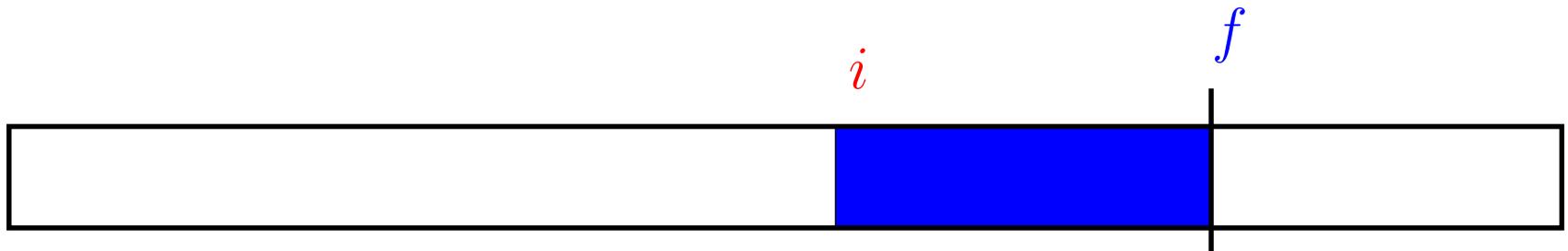
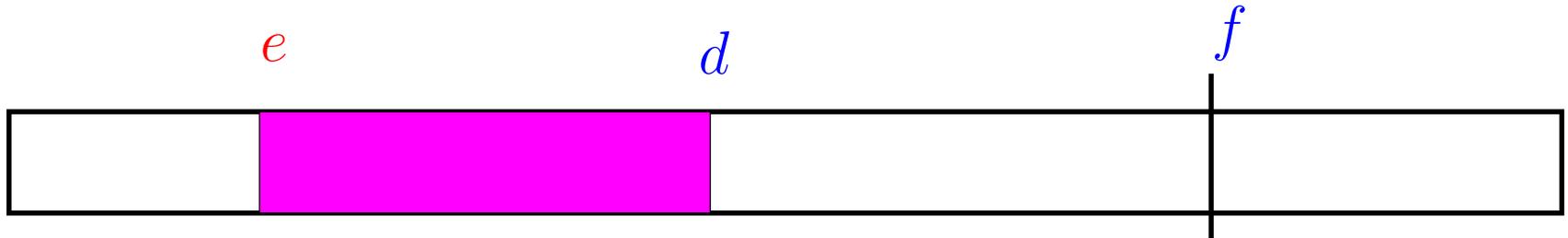
$$A[k] + A[k + 1] + \dots + A[i - 1] \leq 0.$$

O algoritmo linear se aproveita de dessas relações!

# Consumo de tempo

linha	todas as execuções da linha	
1-2	= 2	= $\Theta(1)$
3-4	= 2	= $\Theta(1)$
5	= $n + 1$	= $\Theta(n)$
4-5	= $n$	= $\Theta(n)$
6	= $n$	= $\Theta(n)$
7	= $2 + 3 + \dots + (n + 1)$	= $\Theta(n^2)$
8-9	= $1 + 2 + \dots + n$	= $\Theta(n^2)$
10	$\leq 1 + 2 + \dots + n$	= $O(n^2)$
11	= $n$	= $\Theta(n)$
12	$\leq n$	= $O(n)$
12	= 1	= $\Theta(1)$
<b>total</b>	= $\Theta(2n^2 + 4n + 3) + O(n^2 + n)$	= $\Theta(n^2)$

# Invariantes (i0) e (i2)



# Cara da solução

Solução



# Conclusões

Se  $A[e..d]$  é um segmento de soma máxima então:

- para  $k = e, e + 1, \dots, d$ , vale que

$$A[e] + A[e + 1] + \dots + A[k] \geq 0;$$

- para  $k = e, e + 1, \dots, d$ , vale que

$$A[k] + A[k + 1] + \dots + A[d] \geq 0;$$

- para  $k = 1, 2, \dots, e - 1$ , vale que

$$A[k] + A[k + 1] + \dots + A[e - 1] \leq 0;$$

- para todo  $k = d + 1, d + 2, \dots, n$ , vale que

$$A[d + 1] + A[d + 2] + \dots + A[k] \leq 0.$$

# Mais conclusões

Se  $A[i..f]$  é um segmento de soma máxima terminando em  $f$  então:

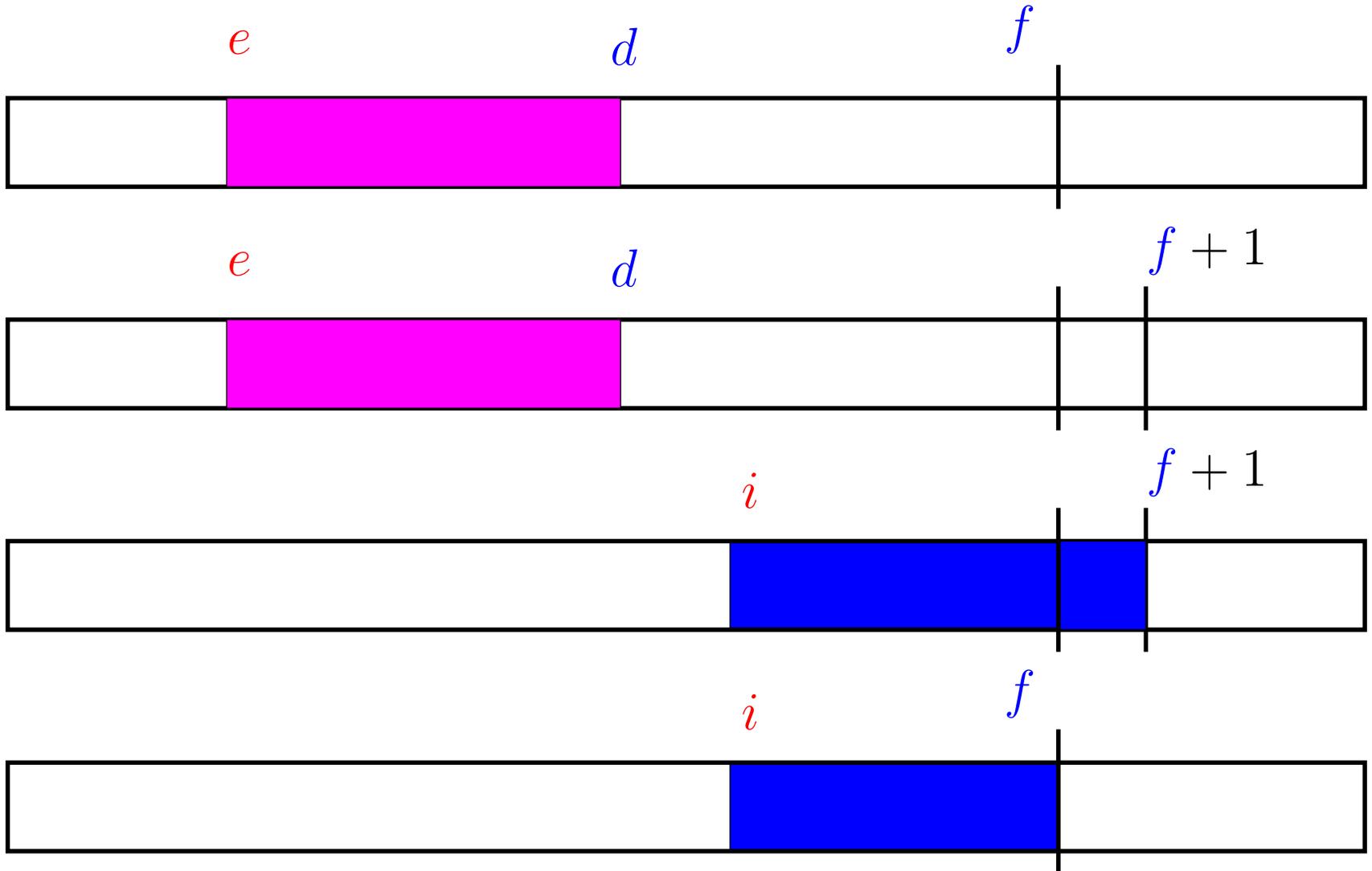
- para  $k = i, i + 1, \dots, f$ , vale que

$$A[i] + A[i + 1] + \dots + A[k] \geq 0;$$

- para  $k = 1, 2, \dots, i - 1$ , vale que

$$A[k] + A[k + 1] + \dots + A[i - 1] \leq 0.$$

# Solução indutiva



# Algoritmo linear

Algoritmo determina um segmento de soma máxima de  $A[1..n]$  (por Jay Kadane).

**SEG-MAX-1** ( $A, n$ )

```
1  somamax ← 0
2  e ← 0    d ← -1    ▷  $A[e..d]$  é vazio
3  soma ← 0
4  i ← 1    f ← 0    ▷  $A[i..f]$  é vazio

5  enquanto f < n faça
6      f ← f + 1
7      soma ← soma +  $A[f]$ 
8      se soma < 0
9          então soma ← 0    i ← f + 1
10     senão se soma > somamax
11         então somamax ← soma    e ← i    d ← f

12  devolva e, d e somamax
```

# Correção

Relação **invariante** chave:

(i0) na linha 5 vale que:  $A[e..d]$  é  
segmento de soma máxima com  $d \leq f$ .

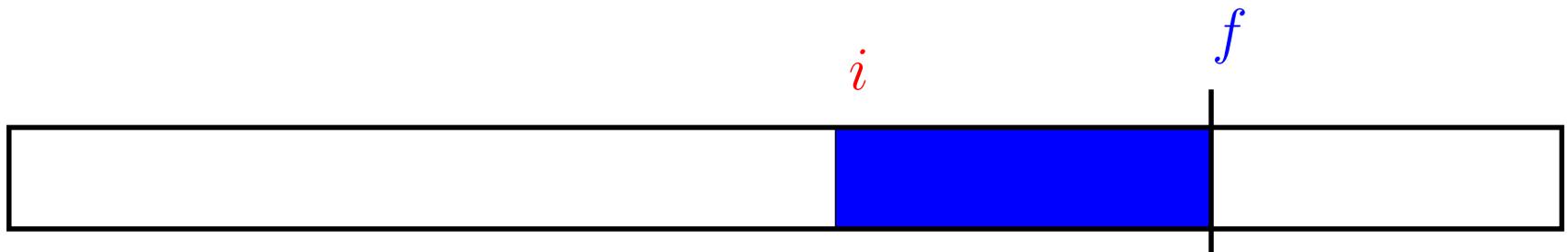
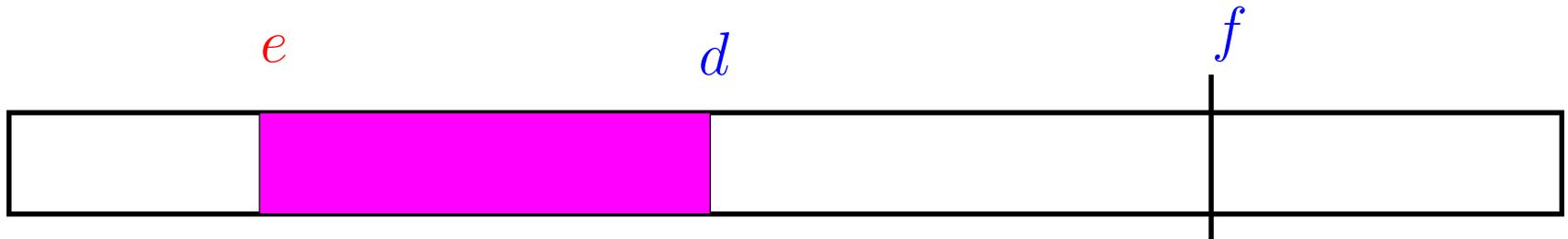
			<i>e</i>	<i>d</i>	<i>f</i>				<i>n</i>	
A	31	-41	59	26	-53	58	97	-93	-23	84

Mais uma relação **invariante**:

(i1) na linha 5 vale que:

$$somamax = A[e] + A[e + 1] + A[e + 2] + \dots + A[d].$$

# Invariantes (i0) e (i2)



# Mais relações invariantes

Na linha 5 vale que:

(i2)  $A[i..f]$  é segmento de soma máxima  
com termino em  $f$ ;

(i3)  $soma = A[i] + A[i + 1] + A[i + 2] + \dots + A[f]$ ;

(i4) para  $k = i, i + 1, \dots, f$ , vale que

$$A[i] + A[i + 1] + \dots + A[k] \geq 0;$$

Na linha 9 vale que:

(i5)  $soma = A[i] + A[i + 1] + \dots + A[f] < 0$

As relações invariantes (i4) e (i5) implicam que na linha 9:

(i6) para  $k = 1, 2, 3, \dots, f$ , vale que

$$A[k] + A[k + 1] + \dots + A[f] < 0;$$

# 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	
1-2	= 2	= $\Theta(1)$
3-4	= 2	= $\Theta(1)$
5	= $n + 1$	= $\Theta(n)$
6-7	= $n$	= $\Theta(n)$
8	= $n$	= $\Theta(n)$
9-10	= $n$	= $\Theta(n)$
11	$\leq n$	= $O(n)$
12	= 1	= $\Theta(1)$
<b>total</b>	= $\Theta(4n + 3) + O(n)$	= $\Theta(n)$

# Conclusões

O consumo de tempo do algoritmo **SEG-MAX-3** é  $\Theta(n^3)$ .

O consumo de tempo do algoritmo **SEG-MAX-2** é  $\Theta(n^2)$ .

O consumo de tempo do algoritmo **SEG-MAX-DC** é  $\Theta(n \lg n)$ .

O consumo de tempo do algoritmo **SEG-MAX-1** é  $\Theta(n)$ .

# Técnicas

- **Evitar recomputações.** Usar espaço para armazenar resultados a fim de evitar recomputá-los (**SEG-MAX-2**, **SEG-MAX-1**, programação dinâmica).
- **Pré-processar os dados.** **HEAPSORT** pré-processa os dados armazenando-os em uma estrutura de dados a fim de realizar computações futuras mais eficientemente.
- **Divisão-e-conquista.** **MERGE-SORT** e **SEG-MAX-DC** utilizam uma forma conhecida dessa técnica.
- **Algoritmos incrementais/varredura.** Solução de um subproblema é estendida a uma solução do problema original (**SEG-MAX-1**).
- **Delimitação inferior.** Projetistas de algoritmos só dormem em paz quando sabem que seus algoritmos são o melhor possível (**SEG-MAX-1**).

# Análise experimental de algoritmos

“O **interesse em experimentação**, é devido ao reconhecimento de que os resultados teóricos, freqüentemente, não trazem informações referentes ao desempenho do algoritmo na prática.”

# Análise experimental de algoritmos

Segundo D.S. Johnson, pode-se dizer que existem quatro motivos básicos que levam a realizar um trabalho de implementação de um algoritmo:

- usar o código em uma aplicação particular, cujo propósito é descrever o impacto do algoritmo em um certo contexto;
- proporcionar evidências da superioridade de um algoritmo;
- melhor compreensão dos pontos fortes e fracos e do desempenho das operações algorítmicas na prática; e
- produzir conjecturas sobre o comportamento do algoritmo no caso-médio sob distribuições específicas de instâncias onde a análise probabilística direta é muito difícil.

# Ambiente experimental

A **plataforma utilizada** nos experimentos é um PC rodando Linux Debian ?? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

Os **códigos estão compilados** com o gcc versão ?? e opção de compilação ??.

As **instâncias são obtidas** utilizando-se o gerador `gera.c` disponível em

`...~coelho/mac0338-2007/tarefa-extra/.`

As implementações comparadas neste experimento são **SEG-MAX-3**, **SEG-MAX-2** e **SEG-MAX-1**.

# Ambiente experimental

A estimativa do tempo é calculada utilizando-se:

```
#include <time.h>
[...]  
clock_t start, end;  
double time;  
  
start = clock();  
  
[...implementação...]  
  
end = clock();  
time = ((double)(end - start))/CLOCKS_PER_SEC;
```

O tempo de entrada/saída é computado separadamente.

# Resultados experimentais

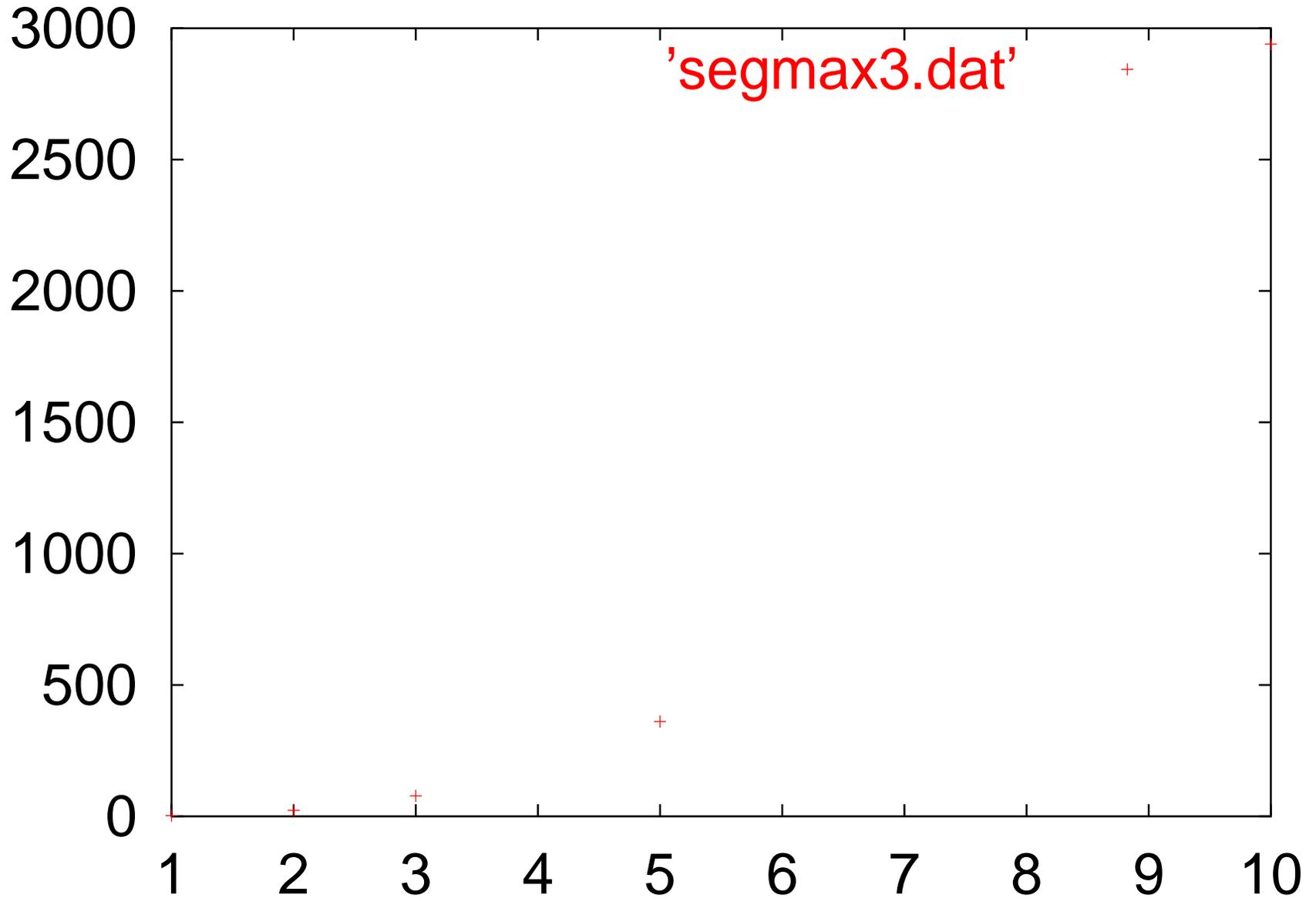
$n$	SEG-MAX-2		SEG-MAX-3	
	clock	user time	clock	user time
1000	0.01	0.01	2.91	2.92
2000	0.03	0.03	23.00	23.04
3000	0.08	0.08	77.58	1m17.57
5000	0.24	0.24	360.78	6m0.45s
10000	1.16	1.62	2940	49m27.32
20000	4.8	4.82	?	?
50000	42.8	42.96	?	?
100000	188.0	3m7.58s	?	?
200000	911.74	15m10.34s	?	?
400000	?	64m22.880s	?	?

# Resultados experimentais

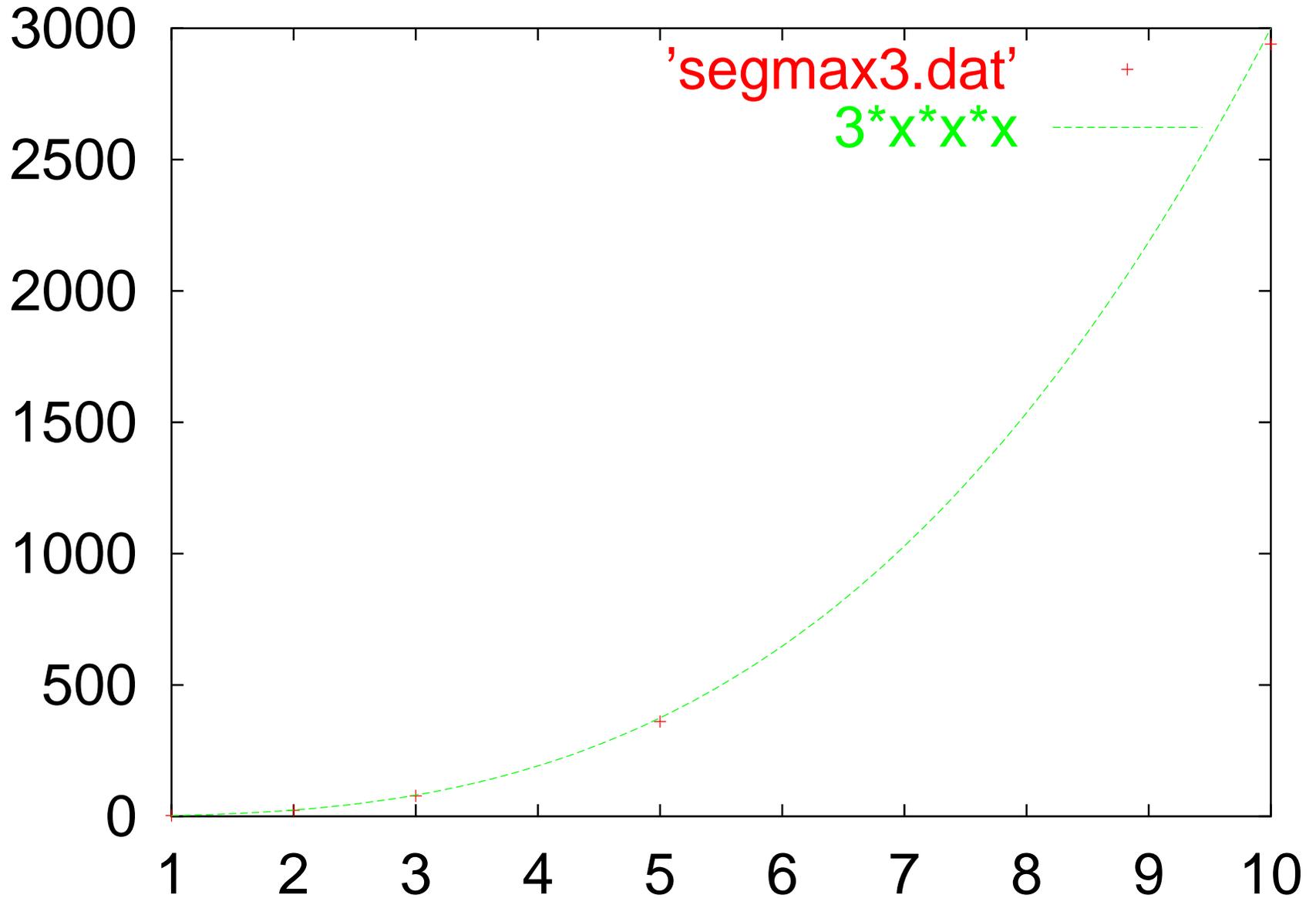
$n$ (M)	SEG-MAX-1 A		SEG-MAX-1 B	
	clock	user time	clock	user time
1	0.06	0.11	0.08	0.07
2	0.11	0.23	0.15	0.06
3	0.16	0.28	0.30	0.12
6	0.32	0.63	0.56	0.29
8	0.43	0.80	0.71	0.43
10	0.55	1.03	1.00	0.52
15	0.82	1.55	1.37	0.64
20	1.08	2.03	2.17	0.95
25	1.37	2.52	2.44	1.38
30	1.70	3.13	3.28	1.68

Todos os tempos estão em segundos.

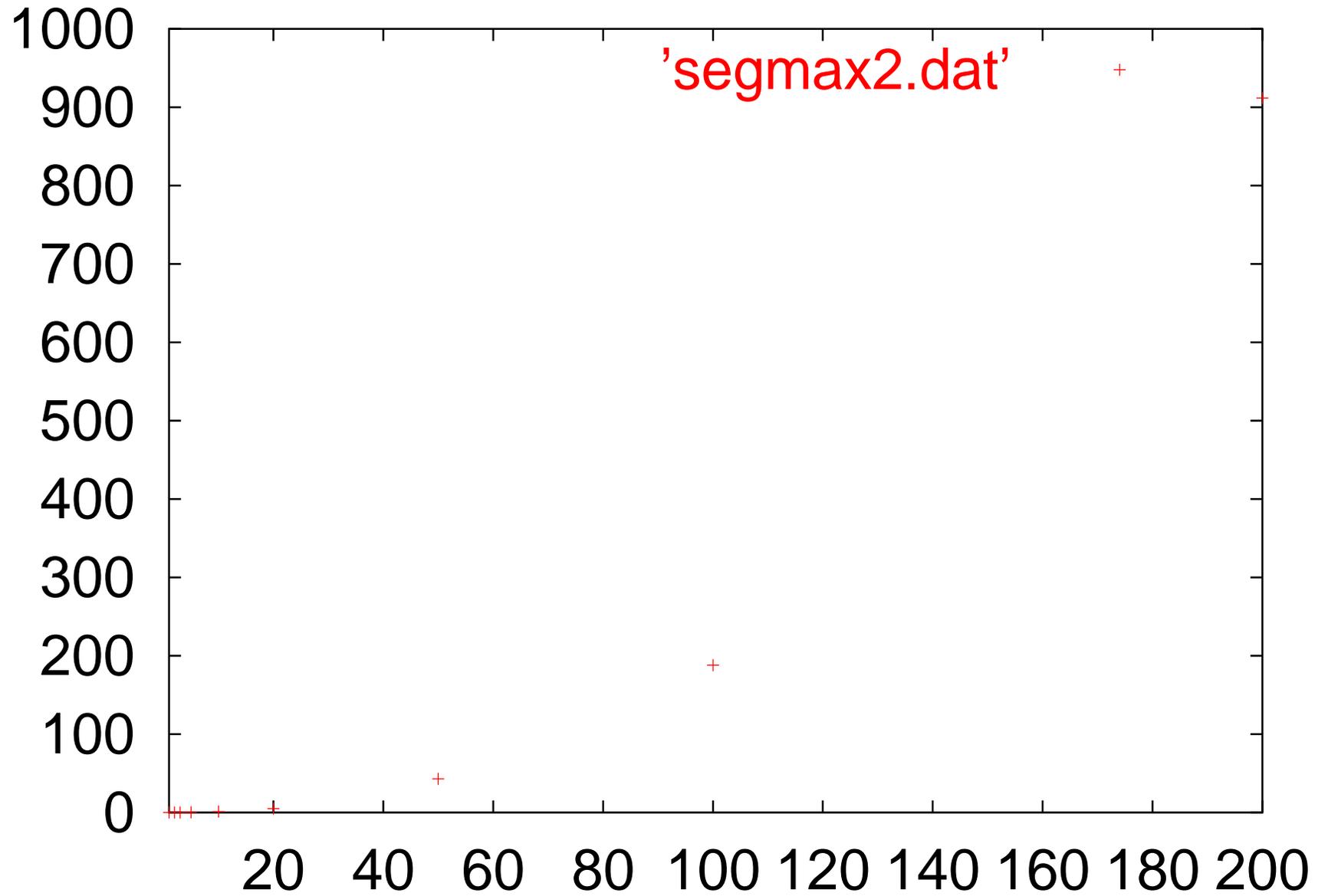
# SEG-MAX-3



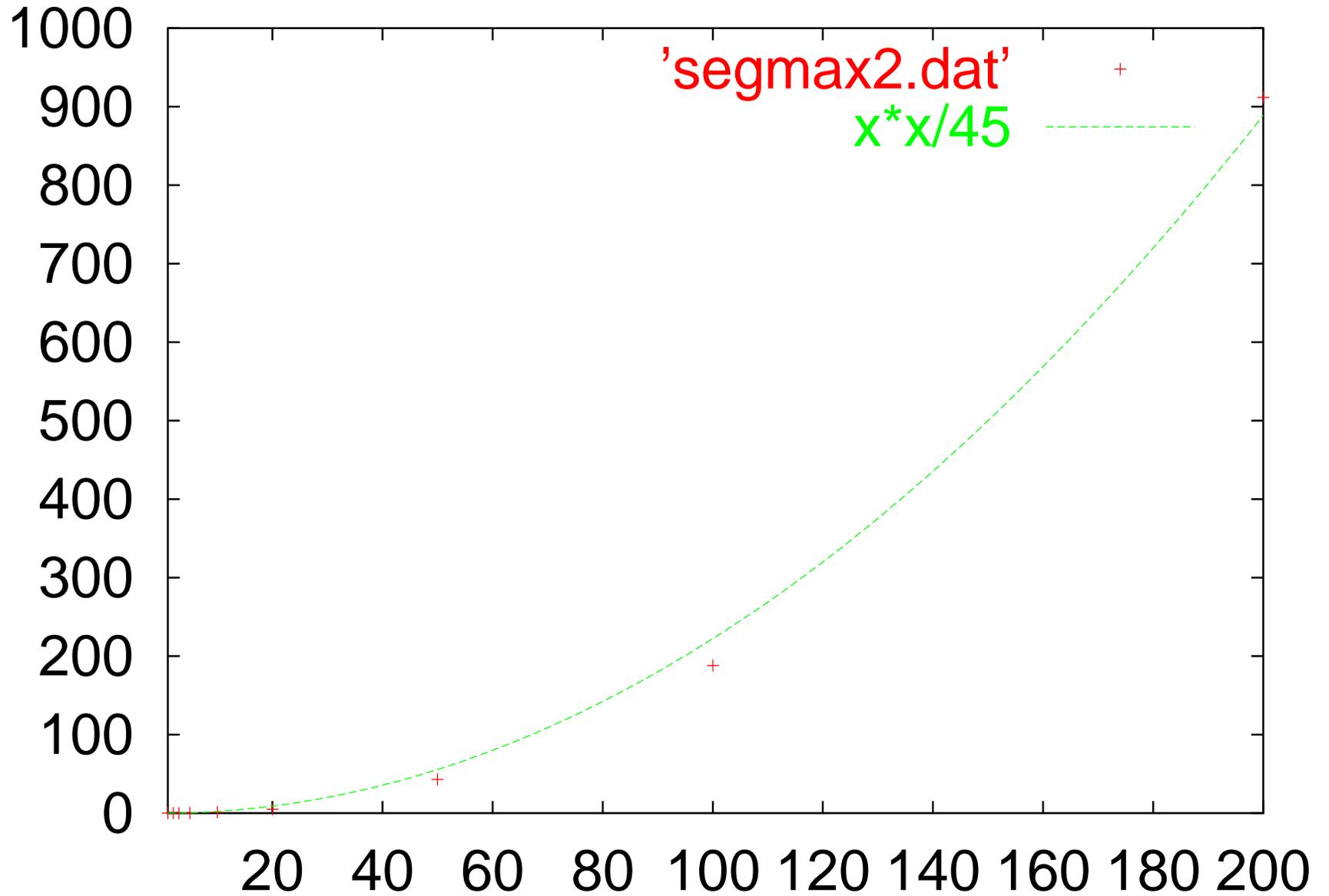
# SEG-MAX-3



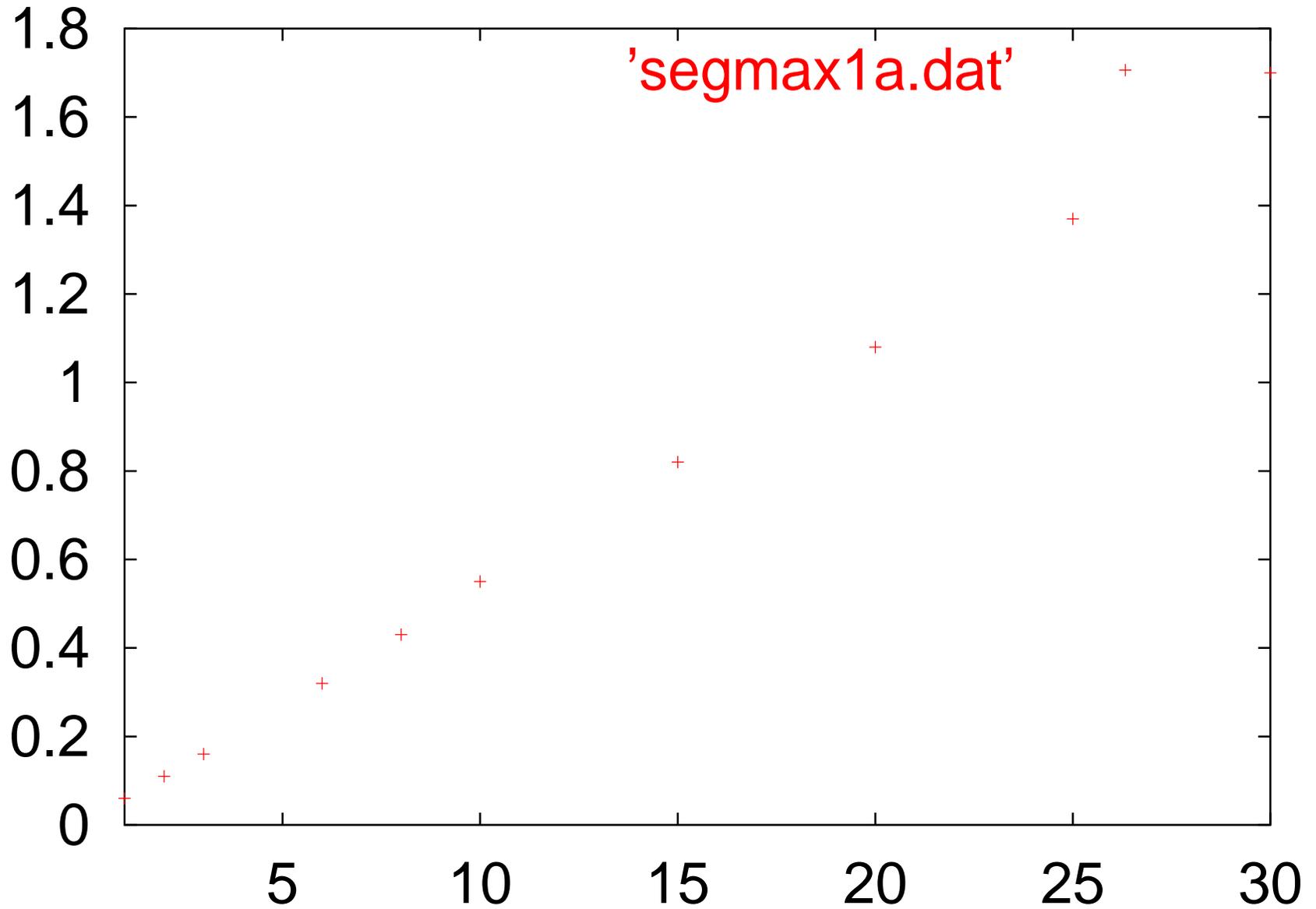
# SEG-MAX-2



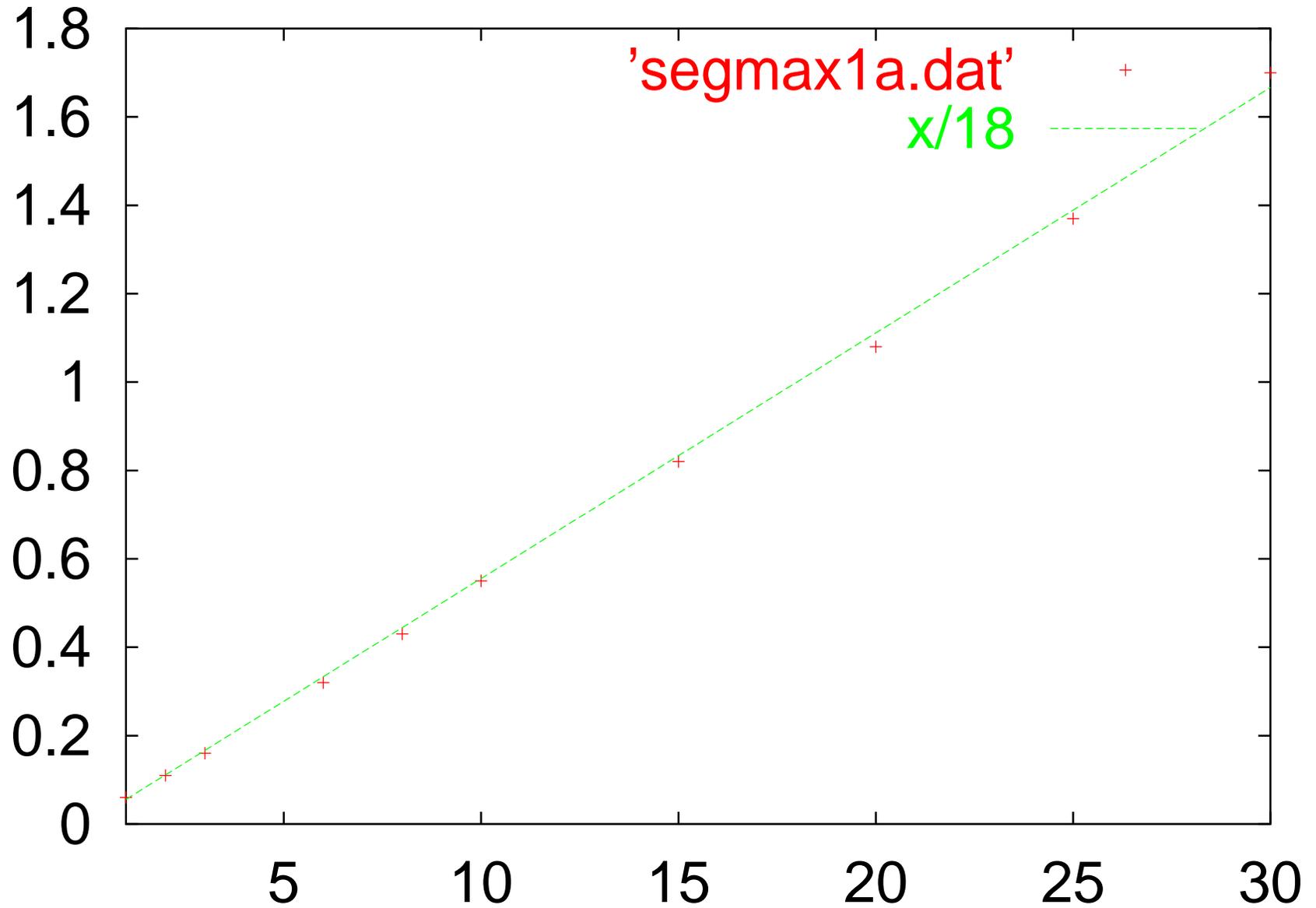
# SEG-MAX-2



# SEG-MAX-1



# SEG-MAX-1



# Exercícios

## Exercício 10.A

Nos algoritmos convencionamos o segmento vazio como sendo o de soma máxima se  $A[1..n]$  tem apenas números negativos. Suponha que em vez disso tivéssemos decidido que nesse caso o maior elemento de  $A[1..n]$  é o segmento de soma máxima. Como isto alteraria os quatro algoritmos?

## Exercício 10.B

Suponha que desejamos determinar um segmento como soma mais próxima de zero, em vez do de soma máxima. Projete um algoritmo para esta tarefa. Qual é o consumo de tempo do seu algoritmo? E se estivéssemos interessados em um algoritmo para encontrar um segmento de soma mais próxima de um dado valor?

## Exercício 10.C

No problema do retângulo de soma máxima é dada uma matriz  $A[1..n, 1..n]$  de números inteiros e deseja-se determinar um retângulo de soma máxima. Projete um algoritmo para este problema. Qual o consumo de tempo do seu algoritmo?

## Exercício 10.D

Modifique o algoritmo **SEG-MAX-DC** para que seu consumo de tempo seja linear.

# Mais exercícios

## Exercício 10.E

Demonstre que qualquer algoritmo para o problema do segmento de soma máxima deve examinar cada um dos  $n$  elementos do vetor. (Algoritmos para alguns problemas podem, corretamente, ignorar alguns dos dados; o algoritmo de Boyer e Moore para busca de padrões é um exemplo.)

## Exercício 10.F

Projete um algoritmo que recebe um vetor  $A[1..n]$  de números inteiros e um número inteiro  $m$  e devolve um índice  $i$  tal que  $1 \leq i \leq n - m$  e a soma  $A[i] + \dots + A[i + m]$  seja próxima de zero. Qual o consumo de tempo do seu algoritmo.

## Exercício 10.G

Resolva a recorrência

$$T(1) = 0$$

$$T(n) = 2T(n/2) + n \text{ para } n = 2, 4, 8, \dots$$

Demonstre por indução que a sua solução está correta.