Melhores momentos

AULA 11

Segmento de soma máxima

Um **segmento** de um vetor $v[0\,.\,.\,n{-}1]$ é qualquer subvetor da forma $v[e\,.\,.\,d]$.

Problema: Dado um vetor $v[0\,.\,.\,n{-}1]$ de números inteiros, determinar um segmento $v[e\,.\,.\,d]$ de soma máxima.

Entra:

	0									n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

Segmento de soma máxima

Sai:

	0		2				6			n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

$v[e\,.\,.\,d] = v[2\,.\,.\,6]$ é segmento de soma máxima.

$v[2\,.\,.\,6]$ tem soma **187**.

Algoritmo café-com-leite

```
void segMax3(int v[],int n,int *e,int *d,
                        int *smax){
  int i, j, k, s;
1 *smax = 0; *e = *d = -1;
2 for (i = 0; /*1*/ i < n; i++)
3   for (j = i; j < n; j++) {
4       s = 0;
5       for (k = i; /*2*/ k <= j; k++)
6           s += v[k];
7       if (s > *smax){
8           *smax = s; *e = i; *d = j;
        }
    }
}
```

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e\,.\,.\,*d]$ é um segmento de soma máxima com $*e < i$. ♡

	*e		i				*d			n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

Consumo de tempo segMax3

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

linha		todas as execuções da linha	
1	$=$	1	$= 1$
2	$=$	$n+1$	$\leq n$
3	$=$	$(n+1)+n+(n-1)+\cdots+1$	$\leq n^2$
4	$=$	$n+(n-1)+\cdots+1$	$\leq n^2$
5	$=$	$(2+\cdots+(n+1))+(2+\cdots+n)+\cdots+2$	$\leq n^3$
6	$=$	$(1+\cdots+n)+(1+\cdots+(n-1))+\cdots+1$	$\leq n^3$
7	$=$	$n+(n-1)+(n-2)+\cdots+1$	$\leq n^2$
8	$\leq$	$n+(n-1)+(n-2)+\cdots+1$	$\leq n^2$
total	$\leq$	$2n^3+4n^2+n+1$	$\sim n^3$

Conclusão

O consumo de tempo do algoritmo segMax3 é proporcional a n^3.

Algoritmo arroz-com-feijão

```
void segMax2(int v[],int n,int *e,int *d,
                    int *smax){
  int i, j, s;
1 *smax = 0; *e = *d = -1;
2 for (i = 0; /*1*/ i < n; i++) {
3      s = 0;
4      for (j = i; j < n; j++){
5          s += v[j];
6          if (/*2*/ s > *smax){
7              *smax = s; *e = i; *d = j;
           }
      }
  }
```

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e\,..*d]$ é um segmento de soma máxima com $*e < i$. ♡

	*e						*d	i		n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

Consumo de tempo segMax2

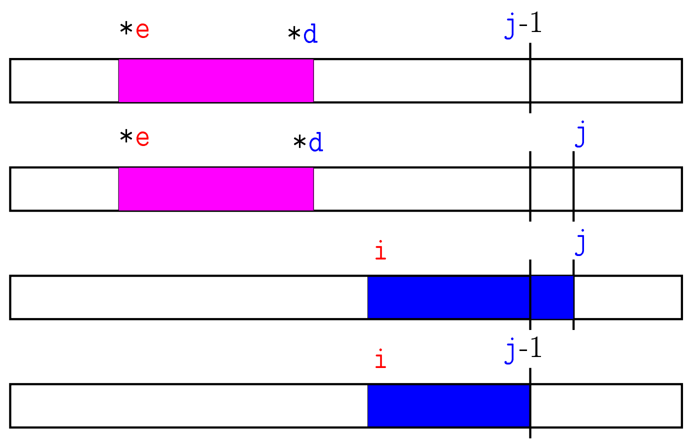
Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha		todas as execuções da linha	
1	$=$	1	$= 1$
2	$=$	$n+1$	$\approx n$
3	$=$	n	$= n$
4	$=$	$(n+1)+n+\cdots+2$	$\leq n^2$
5	$=$	$n+(n-1)+\cdots+1$	$\leq n^2$
6	$=$	$n+(n-1)+\cdots+1$	$\leq n^2$
7	$\leq$	$n+(n-1)+\cdots+1$	$\leq n^2$
total	$\leq$	$4n^2+2n+1$	$\sim n^2$

Conclusão

O consumo de tempo do algoritmo segMax2 é proporcional a n^2.

Nova ideia (indutiva)



Implementação ingênua

Determina um segmento de soma máxima de $v[0\,.\,.\,n{-}1]$.

```
void segMaxI(int v[],int n,int *e,int *d,
                   int *smax){
  int i, j, k, sk, s;
1 s = *smax = 0; *e = *d = -1;
```

Implementação ingênua

```
2 for(i = j = 0; /*1*/ j < n; j++) {
3    s = sk = v[j]; i = j;
4    for(k = j-1; k >= 0; k--) {
5        sk += v[k];
6        if (sk > s){ s = sk; i = k; }
     }
7    if (/*2*/ s > *smax){
8        *smax = s; *e = i; *d = j;
     }
   }
}
```

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e\,.\,.\,*d]$ é segmento de soma máxima com $*d \leq j-1$. ♡

			*e	*d		j				n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

Mais uma relação **invariante**:

(i1) em /*1*/ vale que:

$$smax = v[*e] + v[*e{+}1] + v[*e{+}2] + \cdots + v[*d].$$

Mais relações invariantes

Em /*2*/ vale que:

(i2) $v[i\,.\,.\,j]$ é segmento de soma máxima com término em j **e contendo** $v[j]$;

(i3) $s = v[i] + v[i{+}1] + v[i{+}2] + \cdots + v[j]$;

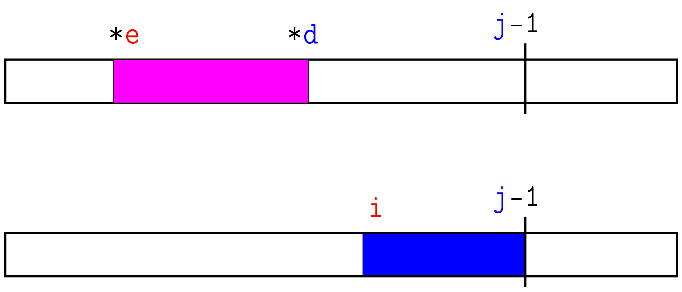
(i4) para $k = i, i+1, \ldots, j{-}1$, vale que

$$v[k] + v[k{+}1] + \cdots + v[j{-}1] \geq 0\,;$$

(i5) para $k = 0, 1, \ldots, i{-}1$, vale que

$$v[k] + v[k{+}1] + \cdots + v[i{-}1] < 0\,;$$

Invariantes (i0) e (i2)



Consumo de tempo segMaxI

linha	todas as execuções da linha			
1	$=$	1	$= 1$
2	$=$	$n+1$	$\approx n$
3	$=$	n	$= n$
4	$=$	$1+2+\ldots+n$	$\leq n^2$
5	$=$	$1+2+\ldots+n-1$	$\leq n^2$
6	$=$	$1+2+\ldots+n-1$	$\leq n^2$
7	$=$	n	$\leq n$
8	$\leq$	n	$= n$
total	$\leq$	$3n^2+4n+1$	$\sim \mathbf{n^2}$

Conclusão

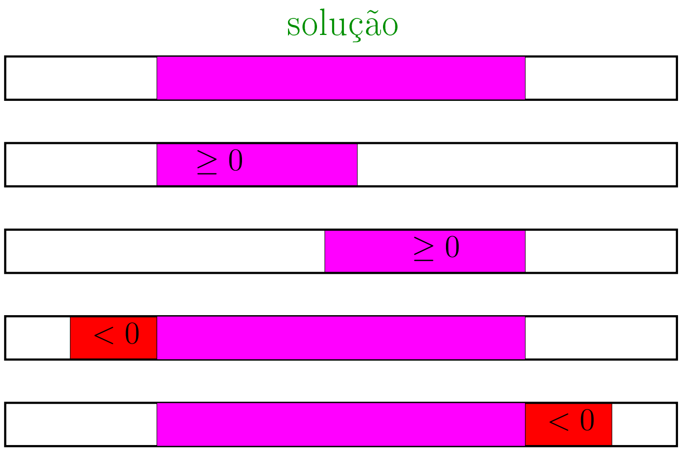
O consumo de tempo do algoritmo `segMaxI` é proporcional a $\mathtt{n}^2$.

AULA 12

Análise de algoritmo (continuação)

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

Cara da solução



Conclusões

Se $\mathtt{v}[*\mathtt{e}\,..\,*\mathtt{d}]$ é um seg de soma máx. então:

- para $\mathtt{k} = *\mathtt{e}, *\mathtt{e}{+}1, \ldots, *\mathtt{d}$, vale que
  $$\mathtt{v}[*\mathtt{e}] + \mathtt{v}[*\mathtt{e}{+}1] + \cdots + \mathtt{v}[\mathtt{k}] \geq 0\,;$$
- para $\mathtt{k} = *\mathtt{e}, *\mathtt{e} + 1, \ldots, *\mathtt{d}$, vale que
  $$\mathtt{v}[\mathtt{k}] + \mathtt{v}[\mathtt{k}{+}1] + \cdots + \mathtt{v}[*\mathtt{d}] \geq 0\,;$$
- para $\mathtt{k} = 1, 2, \ldots, *\mathtt{e} - 1$, vale que
  $$\mathtt{v}[\mathtt{k}] + \mathtt{v}[\mathtt{k}{+}1] + \cdots + \mathtt{v}[*\mathtt{e}{-}1] \leq 0\,;$$
- para todo $\mathtt{k} = *\mathtt{d} + 1, *\mathtt{d} + 2, \ldots, \mathtt{n}$, vale
  $$\mathtt{v}[*\mathtt{d}{+}1] + \mathtt{v}[*\mathtt{d}{+}2] + \cdots + \mathtt{v}[\mathtt{k}] \leq 0\,.$$

Mais conclusões

Se $\mathtt{v}[\mathtt{i}\,..\,\mathtt{j}]$ é um segmento de soma máxima terminando em $\mathtt{j}$ então:

- para $\mathtt{k} = \mathtt{i}, \mathtt{i} + 1, \ldots, \mathtt{j}{-}1$, vale que
  $$\mathtt{v}[\mathtt{i}] + \mathtt{v}[\mathtt{i}{+}1] + \cdots + \mathtt{v}[\mathtt{k}] \geq 0\,;$$
- para $\mathtt{k} = 0, 1, 2, \ldots, \mathtt{i}{-}1$, vale que
  $$\mathtt{v}[\mathtt{k}] + \mathtt{v}[\mathtt{k}{+}1] + \cdots + \mathtt{v}[\mathtt{i}{-}1] \leq 0\,.$$

Algoritmo linear

```
void segMax1(int v[],int n,int *e,int *d,
                      int *smax){
1 int i, j, s, smax;
2 s = smax = 0; *e = *d = -1;
3 for (i = j = 0; /*1*/ j < n; j++) {
4    if (s < 0){
5        /*3*/ i = j; s = v[j];
6    } else s += v[j];
7    if (/*2*/ s > smax){
8        smax = s; *e = i; *d = j;
9    }
10 }
}
```

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e\,..\,*d]$ é segmento de soma máxima com $*d \leq j-1$. ♡

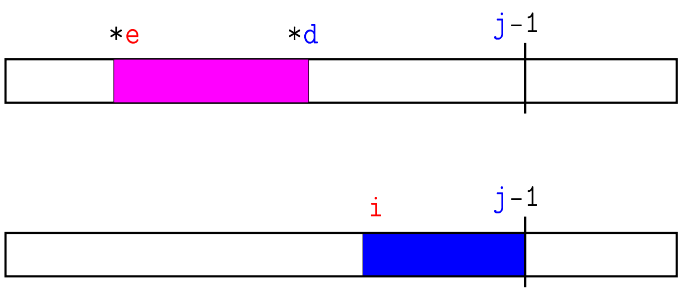
			*e	*d	j					n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

Mais uma relação **invariante**:

(i1) em /*1*/ vale que:

$$smax = v[*e] + v[*e{+}1] + v[*e{+}2] + \cdots + v[*d].$$

Invariantes (i0) e (i2)



Mais relações invariantes

Em /*2*/ vale que:

(i2) $v[i\,..\,j]$ é segmento de soma máxima com término em j **e contendo** $v[j]$;

(i3) $s = v[i] + v[i{+}1] + v[i{+}2] + \cdots + v[j]$;

(i4) para $k = i, i+1, \ldots, j$, vale que

$$v[k] + v[k{+}1] + \cdots + v[j-1] \geq 0\,;$$

(i5) para $k = 0, 1, \ldots, i{-}1$, vale que

$$v[k] + v[k{+}1] + \cdots + v[i{-}1] < 0\,;$$

Mais relações invariantes

Em /*3*/ vale que:

(i6) $s = v[i] + v[i+1] + \cdots + v[j{-}1] < 0$

As relações invariantes (i4) e (i6) implicam que em /*3*/:

(i7) para $k = 0, 1, 2, \ldots, j{-}1$, vale que

$$v[k] + v[k+1] + \cdots + v[j{-}1] < 0\,;$$

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		
2	$=$	1	$= 1$
3	$=$	$n+1$	$\approx n$
4	$=$	n	$= n$
5-6	$=$	n	$= n$
7	$=$	n	$= n$
8	$\leq$	n	$= n$
total	$\leq$	$5n+2$	$\sim n$

Conclusões

O consumo de tempo do algoritmo `segMax3` é proporcional a n^3.

O consumo de tempo do algoritmo `segMax2` é proporcional a n^2.

O consumo de tempo do algoritmo `segMax1` é proporcional a n.

Algumas técnicas

- Evitar recomputações. Usar espaço para armazenar resultados a fim de evitar recomputá-los (`segMax2`, `segMax1`).
- Algoritmos incrementais/varredura. Solução de um subproblema é estendida a uma solução do problema original (`segMax1`).
- Delimitação inferior. Projetistas de algoritmos só dormem em paz quando sabem que seus algoritmos são o melhor possível (`segMax1`).

Notação assintótica

CLRS 3.1

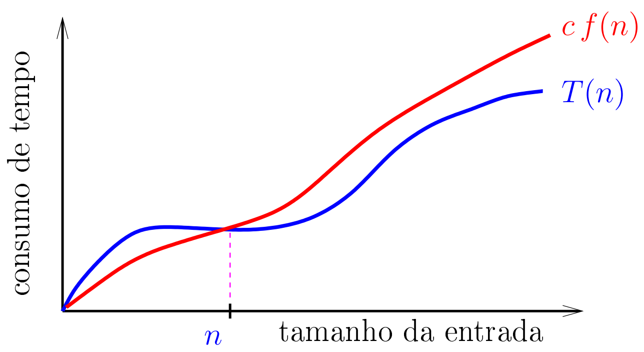
Notação assintótica

Sejam $\mathtt{T}(\mathtt{n})$ e $\mathtt{f}(\mathtt{n})$ funções dos inteiros nos reais. Dizemos que $\mathtt{T}(\mathtt{n})$ **é** $\mathrm{O}(\mathtt{f}(\mathtt{n}))$ se existem constantes positivas c e $\mathtt{n}_0$ tais que

$$\mathtt{T}(\mathtt{n}) \;\leq\; \mathtt{c}\,\mathtt{f}(\mathtt{n})$$

para todo $\mathtt{n} \geq \mathtt{n}_0$.

Notação assintótica

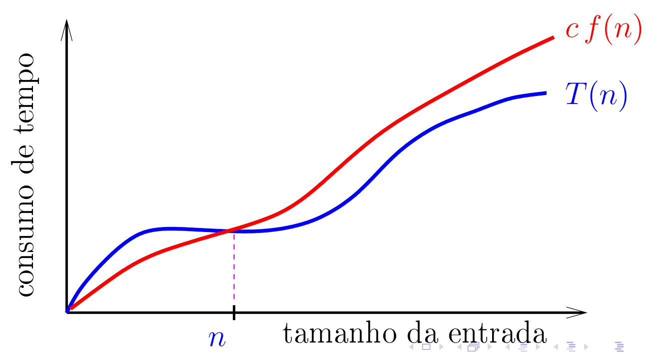


Mais informal

$\mathtt{T}(\mathtt{n})$ é $\mathrm{O}(\mathtt{f}(\mathtt{n}))$ se existe $\mathtt{c} > 0$ tal que

$$\mathtt{T}(\mathtt{n}) \;\leq\; \mathtt{c}\,\mathtt{f}(\mathtt{n})$$

para todo $\mathtt{n}$ suficientemente GRANDE.



Consumo de tempo segMax3

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

linha		todas as execuções da linha	
1	$=$	1	$= O(1)$
2	$=$	$n+1$	$= O(n)$
3	$=$	$(n+1)+n+(n-1)+\cdots+1$	$= O(n^2)$
4	$=$	$n+(n-1)+\cdots+1$	$= O(n^2)$
5	$=$	$(2+\cdots+(n+1))+(2+\cdots+n)+\cdots+2$	$= O(n^3)$
6	$=$	$(1+\cdots+n)+(1+\cdots+(n-1))+\cdots+1$	$= O(n^3)$
7	$=$	$n+(n-1)+(n-2)+\cdots+1$	$= O(n^2)$
8	$\leq$	$n+(n-1)+(n-2)+\cdots+1$	$= O(n^2)$
total	$=$	$O(2n^3+4n^2+n+1)$	$= O(n^3)$

Consumo de tempo segMax2

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

linha		todas as execuções da linha	
1	$=$	1	$= O(1)$
2	$=$	$n+1$	$= O(n)$
3	$=$	n	$= O(n)$
4	$=$	$(n+1)+n+\cdots+2$	$= O(n^2)$
5	$=$	$n+(n-1)+\cdots+1$	$= O(n^2)$
6	$=$	$n+(n-1)+\cdots+1$	$= O(n^2)$
7	$\leq$	$n+(n-1)+\cdots+1$	$= O(n^2)$
total	$=$	$O(4n^2+2n+1)$	$= O(n^2)$

Consumo de tempo segMaxI

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

linha		todas as execuções da linha	
1	$=$	1	$= O(1)$
2	$=$	$n+1$	$= O(n)$
3	$=$	n	$= O(n)$
4	$=$	$2+3+\ldots+(n+1)$	$= O(n^2)$
5	$=$	$1+2+\ldots+n$	$= O(n^2)$
6	$=$	$1+2+\ldots+n$	$= O(n^2)$
7	$=$	n	$= O(n)$
8	$\leq$	n	$= O(n)$
total	$=$	$O(3n^2+4n+1)$	$= O(n^2)$

Conclusões

O consumo de tempo do algoritmo segMax3 é $O(n^3)$.

O consumo de tempo do algoritmo segMax2 é $O(n^2)$.

O consumo de tempo do algoritmo segMax1 é $O(n)$.

$(3/2)n^2 + (7/2)n - 4$ versus $(3/2)n^2$

n	$(3/2)n^2+(7/2)n-4$	$(3/2)n^2$
64	6364	6144
128	25020	24576
256	99196	98304
512	395004	393216
1024	1576444	1572864
2048	6298620	6291456
4096	25180156	25165824
8192	100691964	100663296
16384	402710524	402653184
32768	1610727420	1610612736

$(3/2)n^2 + (7/2)n - 4$ versus $(3/2)n^2$

n	$(3/2)n^2+(7/2)n-4$	$(3/2)n^2$
64	6364	6144
128	25020	24576
256	99196	98304
512	395004	393216
1024	1576444	1572864
2048	6298620	6291456
4096	25180156	25165824
8192	100691964	100663296
16384	402710524	402653184
32768	1610727420	1610612736

$(3/2)n^2$ domina os outros termos

Tamanho máximo de problemas

Suponha que cada operação consome 1 microsegundo ($1\mu s$).

consumo de tempo(μs)	Tamanho máximo de problemas (n)		
	1 segundo	1 minuto	1 hora
$400n$	2500	150000	9000000
$20\,n\,\lceil\lg n\rceil$	4096	166666	7826087
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

Michael T. Goodrich e Roberto Tamassia, *Projeto de Algoritmos*, Bookman.

Crescimento de algumas funções

n	$\lg n$	$\sqrt{n}$	$n \lg n$	n^2	n^3	2^n
2	1	1,4	2	4	8	4
4	2	2	8	16	64	16
8	3	2,8	24	64	512	256
16	4	4	64	256	4096	65536
32	5	5,7	160	1024	32768	4294967296
64	6	8	384	4096	262144	1,8 10^{19}
128	7	11	896	16384	2097152	3,4 10^{38}
256	8	16	1048	65536	16777216	1,1 10^{77}
512	9	23	4608	262144	134217728	1,3 10^{154}
1024	10	32	10240	1048576	1,1 10^{9}	1,7 10^{308}

Nomes de "classes" O

classe	nome
$\mathrm{O}(1)$	constante
$\mathrm{O}(\lg n)$	logarítmica
$\mathrm{O}(n)$	linear
$\mathrm{O}(n \lg n)$	$n \log n$
$\mathrm{O}(n^2)$	quadrática
$\mathrm{O}(n^3)$	cúbica
$\mathrm{O}(n^k)$ com $k \geq 1$	polinomial
$\mathrm{O}(2^n)$	exponencial
$\mathrm{O}(a^n)$ com $a > 1$	exponencial

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;

Análise experimental de algoritmos

- 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 foi um computador rodando Ubuntu GNU/Linux 3.2.0-30

As especificações do computador que geraram as saídas a seguir são

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

MemTotal   : 3354708 kB
```

Ambiente experimental

Os códigos foram compilados com o gcc 4.6.3 e com opções de compilação

-Wall -ansi -O2 -pedantic -Wno-unused-result

As implementações comparadas neste experimento são `segMax3`, `segMax2` e `segMax1`.

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;
```

Resultados experimentais

segMax3		
n	tempo (s)	comentário
256	0.00	
512	0.02	
1024	0.12	
2048	0.89	
4096	6.99	
8192	55.55	$\approx$ 1 min
16384	444.25	$>$ 7 min
32768	59m15.550s	$\approx$ 1 hora

Resultados experimentais

segMax2		
n	tempo (s)	comentário
2048	0.00	
4096	0.01	
8192	0.02	
16384	0.13	
32768	0.53	
65536	2.12	
131072	8.52	
262144	34.08	$\approx$ 0.5 min
524288	136.56	$>$ 2 min
1048576	561.41	$>$ 9 min

Resultados experimentais

segMax1		
n	tempo (s)	comentários
1048576	0.00	
2097152	0.01	
4194304	0.01	
8388608	0.01	
16777216	0.02	
33554432	0.05	
67108864	0.09	
134217728	0.19	$>$ 134 milhões
268435456	0.37	$>$ 268 milhões
536870912	0.75	$>$ 0.5 bilhões