Projeto e Análise de Algoritmos

Aula 4: Dividir para Conquistar ou Divisão e Conquista (2.1-2.2)

DECOM/UFOP
2020– 5º. Período
Anderson Almeida Ferreira
Contém material elaborado por Andréa Iabrudi Tavares

Projeto de Algoritmos: alguns fatos

- A solução eficiente de um problema nem sempre é a mais simples.
- Diferentes problemas geralmente se reduzem a sub-problemas semelhantes.
- Uma estrutura de dados mais apropriada pode melhorar muito o desempenho de um algoritmo.

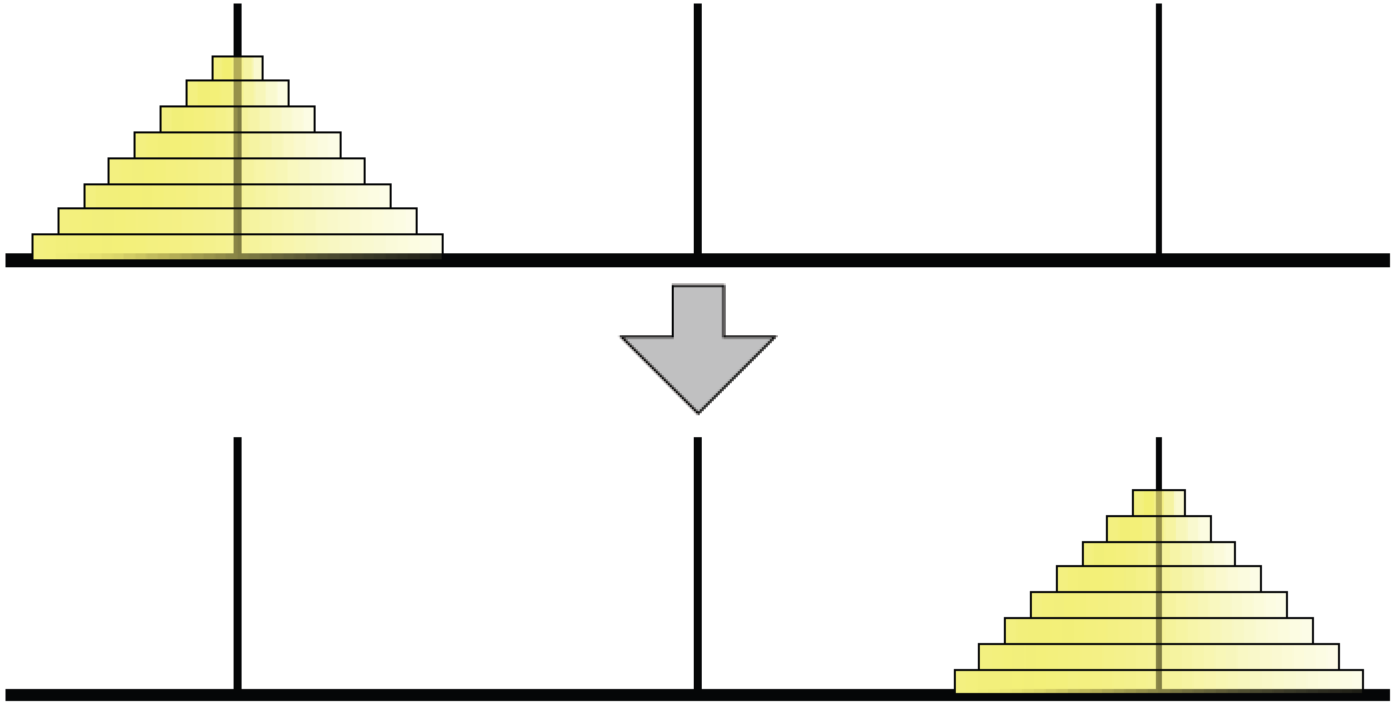
Estratégias para problemas tratáveis

- Estruturas de Dados
  - Use uma estrutura adequada
- Espaço por Tempo
  - Gaste mais espaço para economizar tempo
- Algoritmos Probabilísticos
  - Use aleatoriedade para conseguir eficiência
- Dividir para conquistar (top-down)
  - Divida em subproblemas semelhantes e disjuntos, resolva e combine
- Programação Dinâmica (bottom-up)
  - Comece com subproblemas e componha um maior, reusando solução de subproblemas compartilhados
- Algoritmos Gulosos
  - Sempre pegue o "melhor"

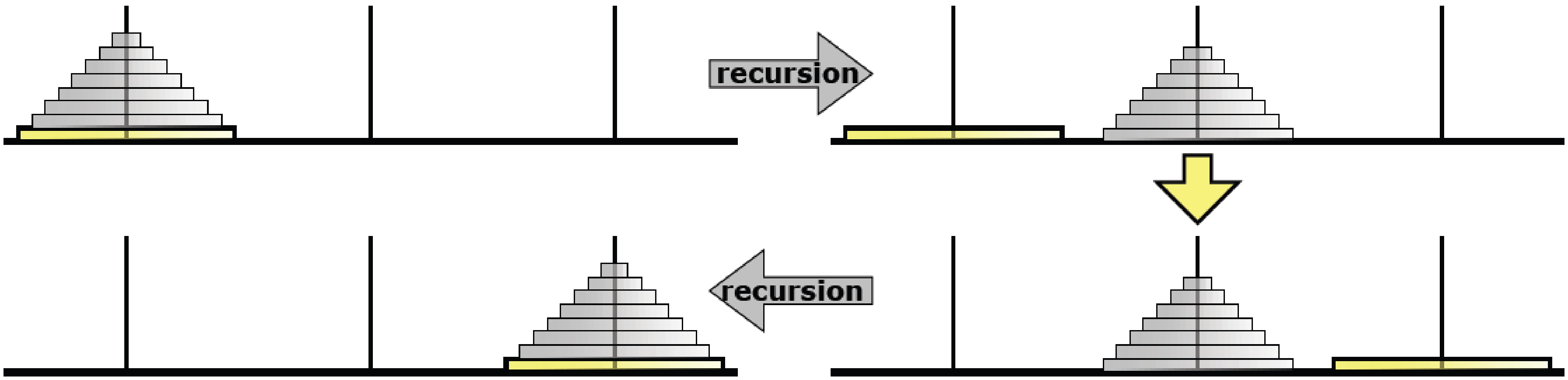
Ordenação:
Um problema, vários algoritmos

- Recursão: Inserção
- Dividir-para-conquistar: MergeSort e QuickSort
- Estrutura de dados: HeapSort
- Tempo por espaço: Counting Sort

Torre de Hanoi



Recursão: Diminuir para conquistar



$\text{HANOI}(n, src, dst, tmp)$:

if $n > 0$

$\text{HANOI}(n-1, src, tmp, dst)$

move disk n from src to dst

$\text{HANOI}(n-1, tmp, dst, src)$

Recursão: Equação de Recorrência

```
HANOI(n, src, dst, tmp):
  if n > 0
      HANOI(n − 1, src, tmp, dst)
      move disk n from src to dst
      HANOI(n − 1, tmp, dst, src)
```

Recursão: Análise

$\underline{\textsc{Hanoi}(n, src, dst, tmp):}$
 if $n > 0$
  $\textsc{Hanoi}(n-1, src, tmp, dst)$
  move disk n from src to dst
  $\textsc{Hanoi}(n-1, tmp, dst, src)$

Tempo de Execução

Dividir para Conquistar (*Divide and Conquer*)

- Problema (instância) pode ser dividido em subproblemas menores (parecidos e independentes) que são resolvidos recursivamente e combinados.

  - Análise por equação de recorrência

Dividir para Conquistar

- Diminuir complexidade (em geral de linear para logarítmico) de algoritmos polinomiais
  - Busca
  - Ordenação
  - Multiplicação
  - Exponenciação

- Apresentar melhor função de complexidade de pior caso
  - Mínimo/máximo

Busca Sequencial

```
function BuscaSequencial(A,x,e,d)
//Entrada: sub-vetor A[e..d]
//Saída: x pertence ao sub-vetor

1. i = e;
2. while (i≤d) & (A[i]≠x)
3.    i = i +1;
4. return (i ≤ d);
```

Lembre-se, nada sendo dito, utilizaremos sempre análise assintótica de pior caso!

Busca Sequencial com Vetor Ordenado

```
function BuscaSequencial2(A,x,e,d)
//Entrada: sub-vetor A[e..d] ordenado
//Saída: x pertence ao sub-vetor

1. i = e;
2. while (i≤d) & (A[i]≤x)
3.    i = i +1;
4. if (i ≤ d): return (A[i]==x);
```

Busca Sequencial Recursiva

```
function BuscaSequencialRec(A,x,e,d)
//Entrada: sub-vetor A[e..d] ordenado
//Saída: x pertence ao sub-vetor

if (e = d): return (A[e] = x);
if (A[e]=x):
      return true;
else:
   if (A[e]>x):
      return false;
   else:
      return BuscaSequencialRec(A,x,e+1,d);
```

D&C: Busca Binária

```
function BuscaBinária(A,x,e,d)
//Entrada: sub-vetor A[e..d] ordenado
//Saída: x pertence ao sub-vetor

if (e = d): return (A[e] = x);
m = (e+d)/2;
if (A[m] = x):
  return true;
else:
  if (A[m] > x):
    return BuscaBinária(A,x,e,m-1);
  else:
    return BuscaBinária(A,x,m+1,d);
```

D&C: Passos

- Divida
  - problema em subproblemas menores
- Conquiste
  - resolva recursivamente cada subproblema OU
  - limite de tamanho: use outro método
- Combine
  - soluções de subproblemas para resolver problema

D&C: Meta-algoritmo

```
function DivididirConquistar(P)
//Entrada: problema P
//Saída: solução S para P

if (ÉPequeno(P)):
   S = ResolvePequeno(P);
else:
   Divida(P, P1, … , Pa);
   for i=1..a:
      Si = DividirConquistar(Pi);
   S = Combine(S1,…,Sa);
return S;
```

Multiplicação de Inteiros Grandes

- x e y de n bits
- n multiplicações de 1-n bit
- n somas de n-n bits

	1	1	0	1
$\times$	1	0	1	1

Carry:	1			1	1	1		
		1	1	0	1	0	1	(53)
		1	0	0	0	1	1	(35)
	1	0	1	1	0	0	0	(88)

Custo: $\Theta(n^2)$

Multiplicação de Inteiros Grandes

- Podemos fazer melhor usando D&C?
  - Como dividir?

$$x = \boxed{\quad x_L \quad} \boxed{\quad x_R \quad} = 2^{n/2}x_L + x_R$$

$$y = \boxed{\quad y_L \quad} \boxed{\quad y_R \quad} = 2^{n/2}y_L + y_R.$$

  - Como combinar?

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R.$$

  - Quanto custa?

Custo: $\Theta(n^2)$

D&C para Multiplicação

- Gauss em no século XVIII e Karatsuba em 1962:

$$bc + ad = (a+b)(c+d) - ac - bd.$$

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R.$$

$$\boxed{x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R}$$

D&C Multiplicação: algoritmo

function multiply(x, y)
Input: Positive integers x and y, in binary
Output: Their product

$n =$ max(size of x, size of y)
if $n = 1$: return xy

Parada recursão

x_L, x_R = leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of x
y_L, y_R = leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of y

Divide

$P_1 = \text{multiply}(x_L, y_L)$
$P_2 = \text{multiply}(x_R, y_R)$
$P_3 = \text{multiply}(x_L + x_R, y_L + y_R)$

Conquiste: Recursão

return $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$

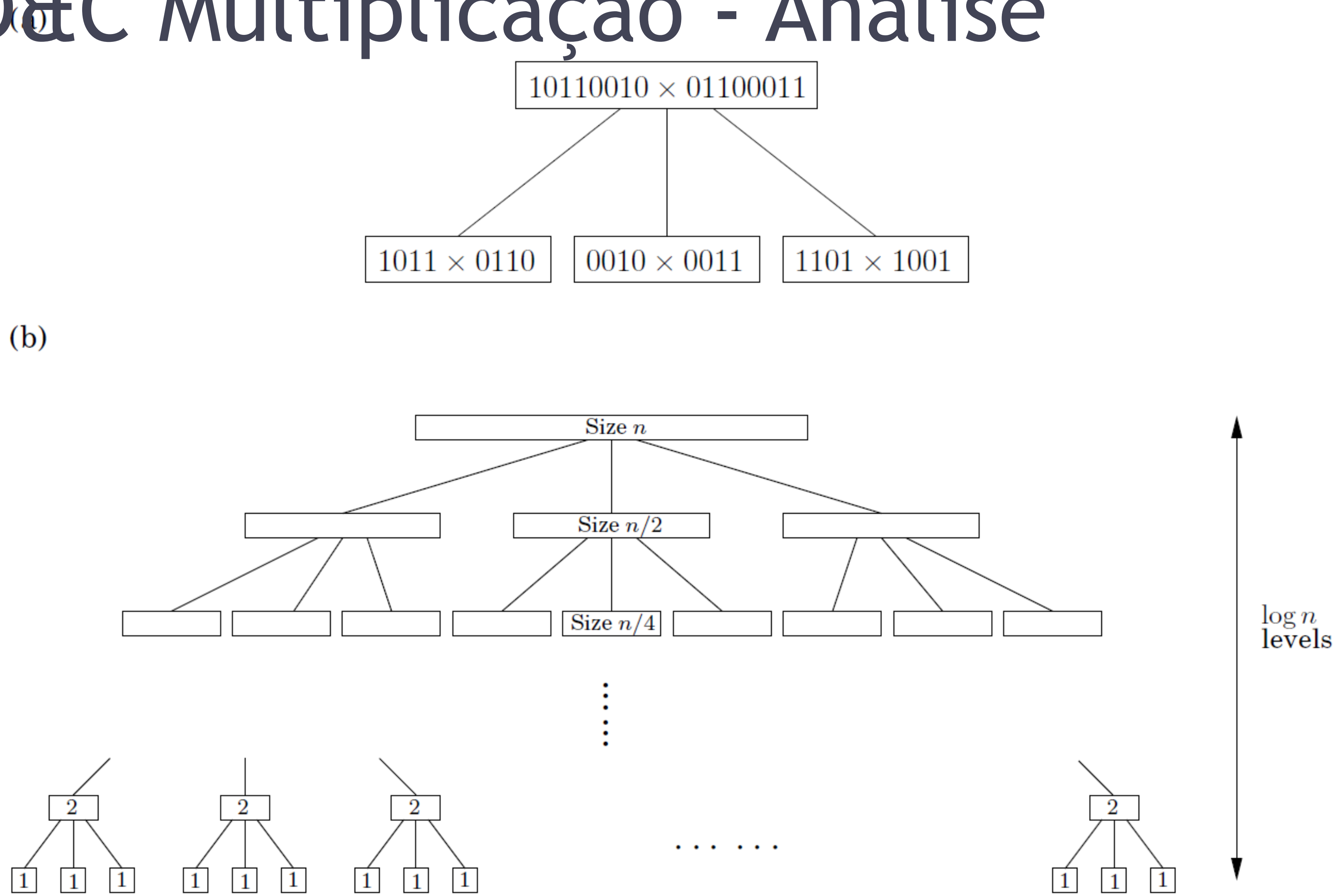
Combine

D&C Multiplicação: Complexidade

```
function multiply(x, y)
Input:  Positive integers x and y, in binary
Output:  Their product

n = max(size of x, size of y)
if n = 1:  return xy

x_L, x_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of x
y_L, y_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of y

P_1 = multiply(x_L, y_L)
P_2 = multiply(x_R, y_R)
P_3 = multiply(x_L + x_R, y_L + y_R)
return P_1 × 2^n + (P_3 − P_1 − P_2) × 2^{n/2} + P_2
```

$$T(n) = \Theta\left(n^{\log_2 3}\right) = \Theta\left(n^{1.585}\right)$$

D&C Multiplicação - Análise



decom
departamento de computação

Forma Geral de Recursão

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

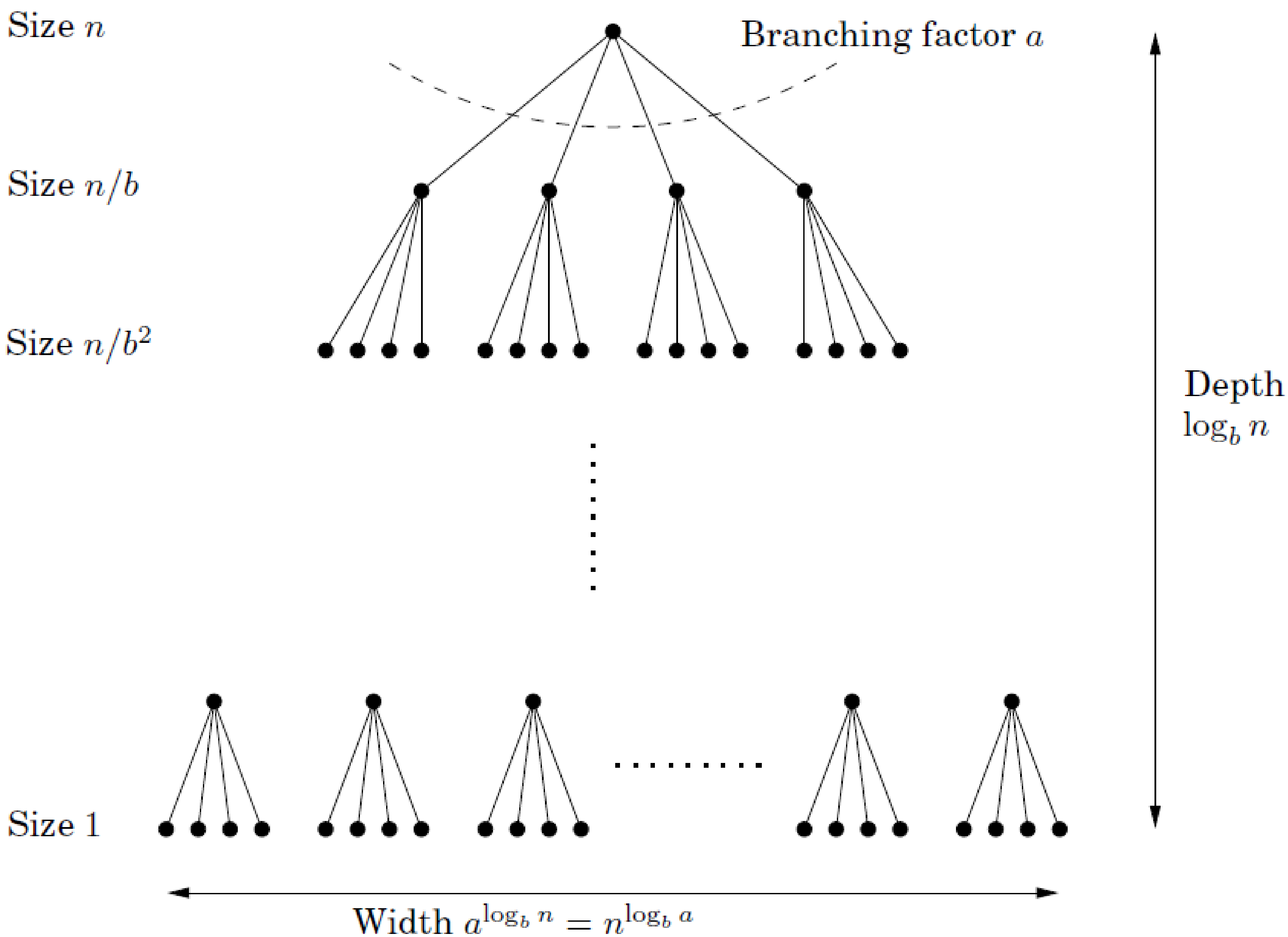
Número de subproblemas

Tamanho dos subproblemas

Custo local da função

decom departamento de computação

Árvore de recursão



Teorema Mestre: intuição

- A cada nível *k* da recursão, qual o total de trabalho?

$$a^k \times O\left(\frac{n}{b^k}\right)^d \;=\; O(n^d) \times \left(\frac{a}{b^d}\right)^k.$$

- Complexidade é o somatório
  - Primeiro ou último termos se forem diferentes
  - Se iguais, multiplica pelo número de termos.

Teorema Mestre – forma simplificada

$T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$,

$$T(n) \;=\; \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a\,. \end{cases}$$

decom
departamento
de computação

Exercício

- Usando o Teorema Mestre simplificado, informe a ordem de complexidade obtida pelas relações de recorrência a seguir. A caso base é $T(1) = O(1)$.

  a) $T(n) = 3T(n/4) + O(n)$

  b) $T(n) = 9\ T(n/3) + O(n^2)$

  c) $T(n) = 9\ T(n/3) + O(n^3)$

D&C Ordenação: MergeSort

- Von Neumann (1945)

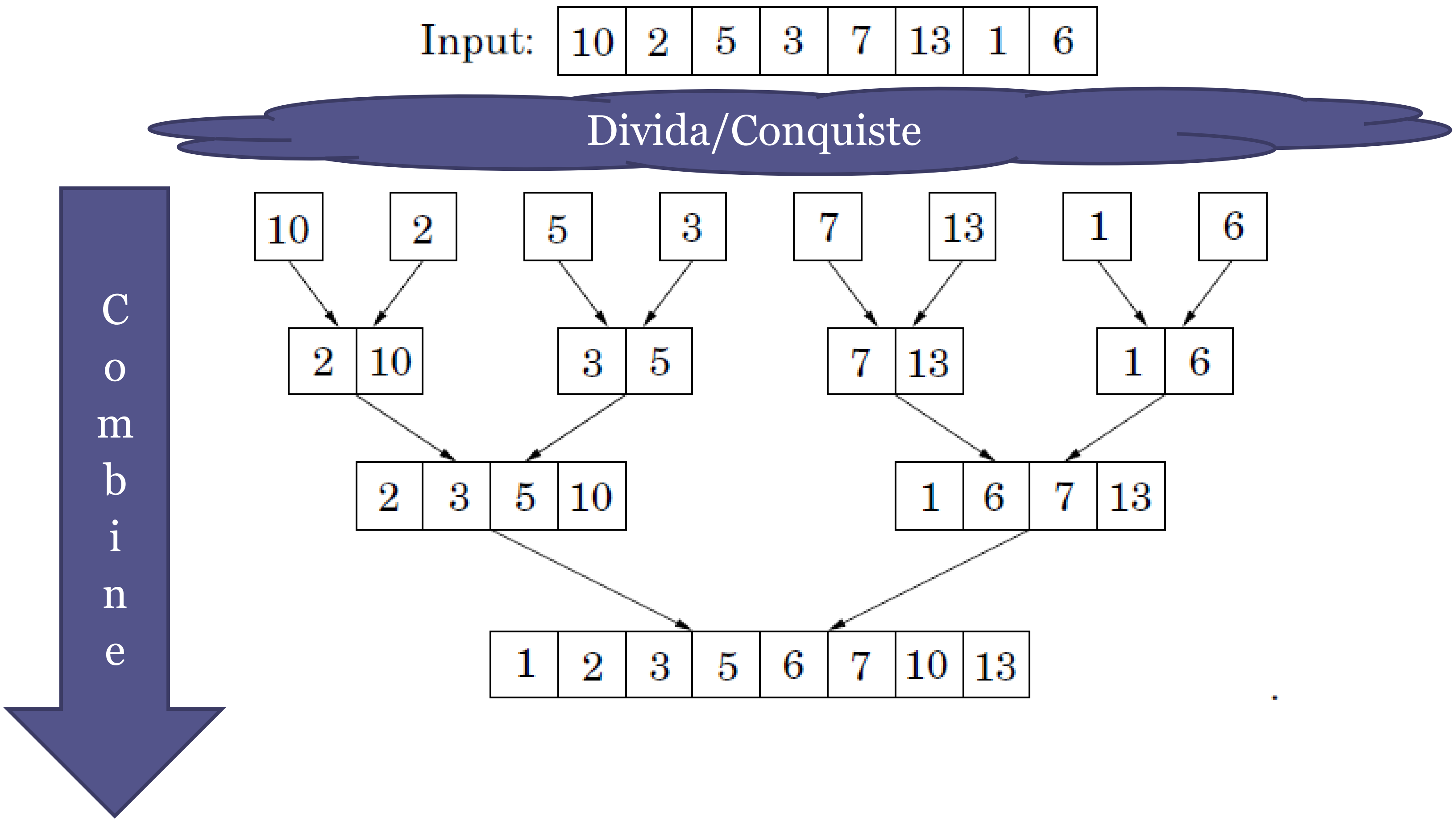
Input:	S	O	R	T	I	N	G	E	X	A	M	P	L
Divide:	S	O	R	T	I	N	G	E	X	A	M	P	L
Recurse:	I	N	O	S	R	T	A	E	G	L	M	P	X
Merge:	A	E	G	I	L	M	N	O	P	S	R	T	X

D&C: MergeSort

```
function mergesort(a[1...n])
Input:  An array of numbers a[1...n]
Output:  A sorted version of this array

if n > 1:
  return merge(mergesort(a[1...⌊n/2⌋]), mergesort(a[⌊n/2⌋ + 1...n]))
else:
  return a
```

```
function merge(x[1...k], y[1...l])
if k = 0:  return y[1...l]
if l = 0:  return x[1...k]
if x[1] ≤ y[1]:
  return x[1] ∘ merge(x[2...k], y[1...l])
else:
  return y[1] ∘ merge(x[1...k], y[2...l])
```

MergeSort: Exemplo



MergeSort: Complexidade

Quanto custa?

```
function mergesort(a[1...n])
Input:  An array of numbers a[1...n]
Output:  A sorted version of this array

if n > 1:
  return merge(mergesort(a[1...⌊n/2⌋]), mergesort(a[⌊n/2⌋+1...n]))
else:
  return a
```

```
function merge(x[1...k], y[1...l])
if k = 0:  return y[1...l]
if l = 0:  return x[1...k]
if x[1] ≤ y[1]:
  return x[1] ∘ merge(x[2...k], y[1...l])
else:
  return y[1] ∘ merge(x[1...k], y[2...l])
```

$$T(n) = \Theta(n \log n)$$

Extra: Combinando iterativamente

```
MERGE(A[1..n], m):
  i ← 1;  j ← m + 1
  for k ← 1 to n
      if j > n
          B[k] ← A[i];  i ← i + 1
      else if i > m
          B[k] ← A[j];  j ← j + 1
      else if A[i] < A[j]
          B[k] ← A[i];  i ← i + 1
      else
          B[k] ← A[j];  j ← j + 1

  for k ← 1 to n
      A[k] ← B[k]
```

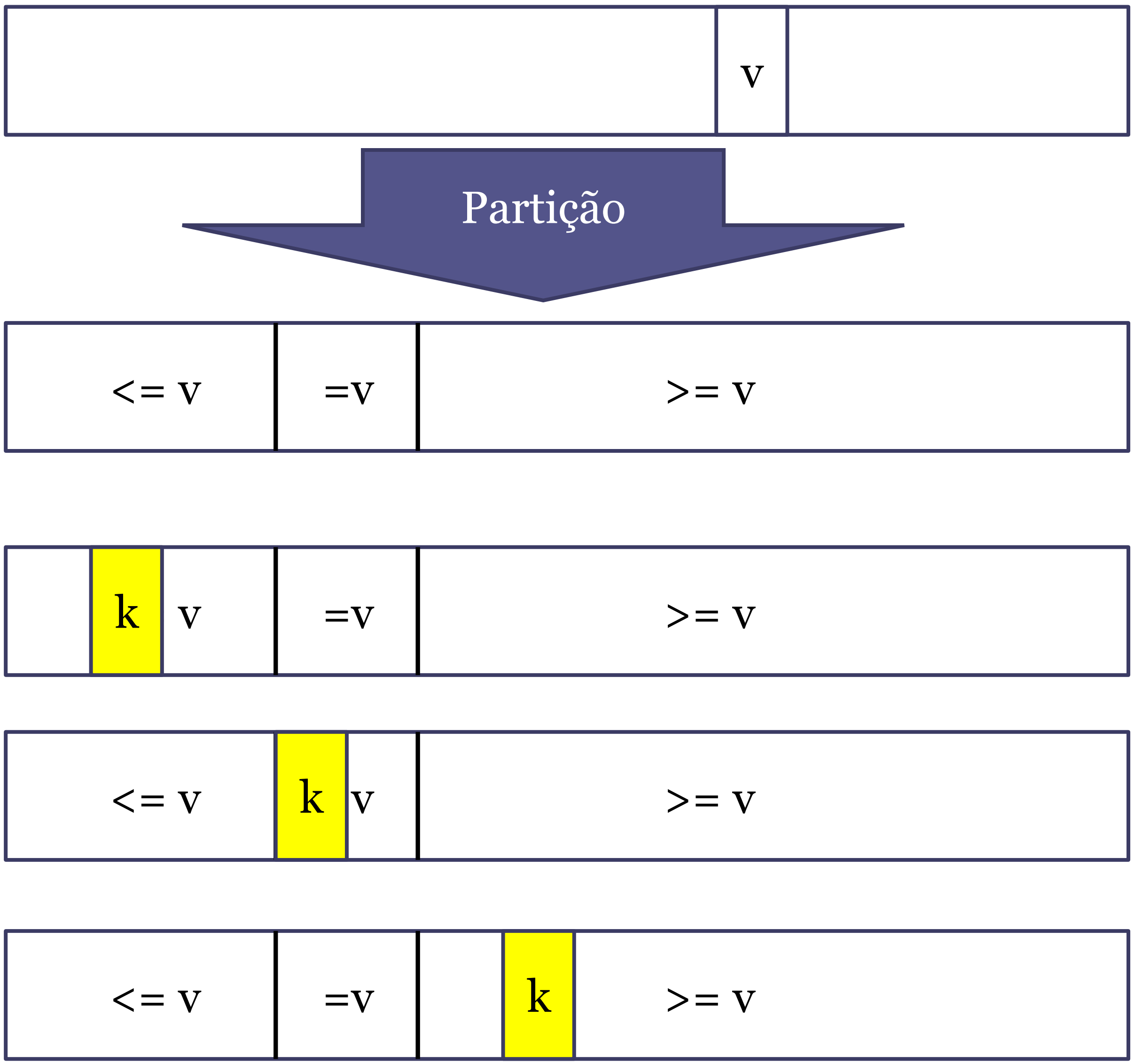
Seleção (Mediana)

- Entrada: Lista de n números S e inteiro k
- Saída: k-ésimo menor elemento de S

- Qual a complexidade?
  - Mínimo: $\Theta(n)$
  - Ordenação: $\Theta(n \log n)$

- Importância da aleatoriedade (outro paradigma de projeto).

D&C Seleção: dividindo...



decom departamento de computação

D&C Seleção: conquistando...

$$\text{selection}(S,k) = \begin{cases} \text{selection}(S_L, k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

$S:$

2	36	5	21	8	13	11	20	5	4	1

$S_L:$

2	4	1

$S_v:$

5	5

$S_R:$

36	21	8	13	11	20

Seleção: Algoritmo

$$\text{selection}(S,k) = \begin{cases} \text{selection}(S_L,k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

```
PARTITION(A[1..n], p):
  if (p ≠ n)
      swap A[p] ↔ A[n]
  i ← 0;  j ← n
  while (i < j)
      repeat i ← i + 1 until (i = j or A[i] ≥ A[n])
      repeat j ← j − 1 until (i = j or A[j] ≤ A[n])
      if (i < j)
          swap A[i] ↔ A[j]
  if (i ≠ n)
      swap A[i] ↔ A[n]
  return i
```

D&C Seleção: Complexidade

Quanto custa?

$$\text{selection}(S, k) = \begin{cases} \text{selection}(S_L, k) & \text{if } k \leq |S_L| \\ v & \text{if } |S_L| < k \leq |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

Tamanhos das listas dependem do pivô v.
Como escolher????

D&C Seleção: Complexidade

- Melhor caso: partições sempre balanceadas...
  - Oráculo

$S:$

2	36	5	21	8	13	11	20	5	4	1

$$\boxed{T(n) = T(n/2) + O(n)}$$

decom departamento de computação

D&C Seleção: Complexidade

- Pior caso: partição diminui somente de 1...
  - Primeiro elemento, achar o máximo e lista ordenada.

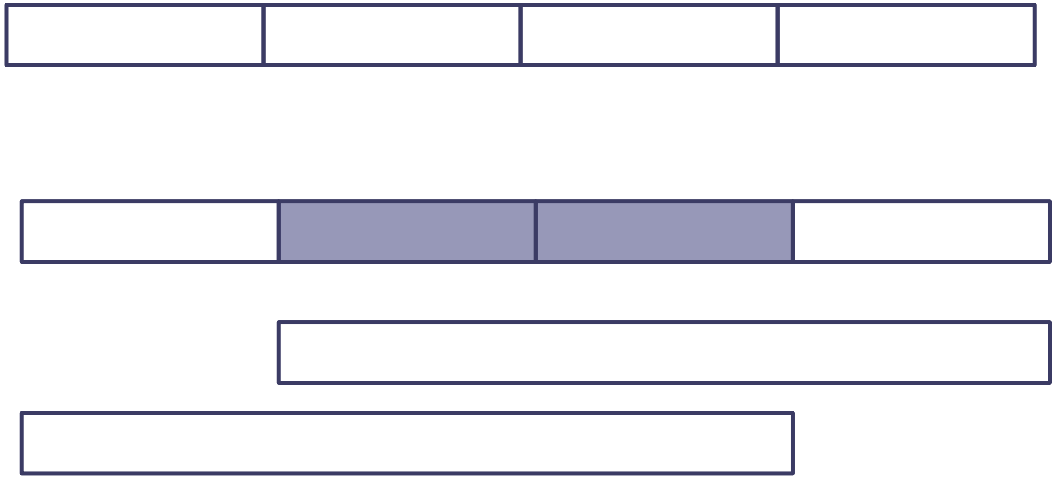
D&C e Probabilístico: Seleção

- Escolher o pivô aleatoriamente...
- Ideal: divide no meio
  - Custo:
  - Probabilidade (se todos diferentes):

- Boa: divide em no mínimo ¾
  - Custo:
  - Probabilidade: ????

D&C e Probabilístico: Seleção

- Boa: divide em no mínimo ¾
    - Probabilidade de v estar no 2o. ou 3o. quartil



decom departamento de computação

D&C e Probabilístico: Seleção

- Boa: divide em no mínimo ¾
  - Probabilidade de v estar no 2o. ou 3o. Quartil
    - P[v dividir em ¾] = 0.5
- Qual o número esperado de escolhas para conseguir uma boa?
  - Número de jogadas até dar cara...
    - E[número de escolhas até dividir em ¾] = 2

D&C Ordenação: QuickSort

- Hoare 1962

Input:	S	O	R	T	I	N		G		E	X	A	M	P	L
Choose a pivot:	S	O	R	T	I	$\boxed{N}$		G		E	X	A	M	P	L
Partition:	M	A	E	G	I	L	|	N	|	R	X	O	S	P	T
Recurse:	A	E	G	I	L	M	|	N	|	O	P	S	R	T	X

D&C: QuickSort

Quanto custa?

```
QUICKSORT(A[1..n]):
  if (n > 1)
      Choose a pivot element A[p]
      k ← PARTITION(A, p)
      QUICKSORT(A[1..k − 1])
      QUICKSORT(A[k + 1..n])
```

Depende da escolha do pivô!!!!!!
Sabemos, contudo, que escolha aleatória provê, em número esperado de duas pivotações, a divisão em 3n/4 .

```
PARTITION(A[1..n], p):
  if (p ≠ n)
      swap A[p] ↔ A[n]
  i ← 0;  j ← n
  while (i < j)
      repeat i ← i + 1 until (i = j or A[i] ≥ A[n])
      repeat j ← j − 1 until (i = j or A[j] ≤ A[n])
      if (i < j)
          swap A[i] ↔ A[j]
  if (i ≠ n)
      swap A[i] ↔ A[n]
  return i
```

MergeSort e QuickSort

Fase	MergeSort	QuickSort
Dividir	O(1)	O(n)
Conquistar	2T(n/2)	T(k) T(n-k)
Combinar	O(n)	O(1)

decom
departamento de computação

D&C: resumo

- Problemas menores (fração), independentes e não-sobrepostos
- Divisão e combinação são partes não-recursivas
  - Algoritmos tendem a tornar uma das duas mais complicada
  - Mergesort, Quicksort
- Importante definir o que é pequeno
  - Parada de recursão (multiplicação de números)

D&C: Exercícios

- Calcular x^n em $\Theta(\log n)$ passos.

Outros exemplos interessantes

- RSA
- Fast Fourier Transform
- Par de pontos mais próximo
- Mínimo e máximo
- Fibonacci

Teorema Mestre – forma geral

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), a \geq 1, b > 1, f(n) > 0$$

$$1. f(n) = O\left(n^{\log_b a - \varepsilon}\right), \varepsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$$

$$2. f(n) = \Theta\left(n^{\log_b a} \log^k n\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$$

$$3. f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right), \varepsilon > 0, af(n/b) \leq cf(n) \Rightarrow T(n) = \Theta\left(f(n)\right),$$

$$onde \qquad c < 1$$

Teorema Mestre: Aplicação Mecânica

- Identifique a, b, f(n)
- Calcule grau = log_b (a)
- Compare n^grau e f(n)
- Encontre o maior

decom
departamento de computação

Exemplos de Teorema Mestre

- $T(n) = 9T(n/3)+n$
- $T(n) = T(2n/3)+1$
- $T(n) = 3T(n/4)+n \log n$

Quando não se aplica?

<u>function</u> <u>fatorial</u>(*n*)

return *n* * fatorial(*n-1)*;

$$T(n)=\begin{cases}1 & n=1\\ 1+T(n-1) & n>1\end{cases}$$

Teorema Mestre: Quando não se aplica

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), a \geq 1, b > 1, f(n) > 0$$

$$1. f(n) = O\left(n^{\log_b a - \varepsilon}\right), \varepsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$$

$$2. f(n) = \Theta\left(n^{\log_b a} \log \right. \ldots \left.\Theta\left(n^{\log_b a} \log\ n\right)\right.$$

$$3. f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right), \varepsilon > 0, af(n/b) \leq cf(n) \Rightarrow T(n) = \Theta(f(n))$$

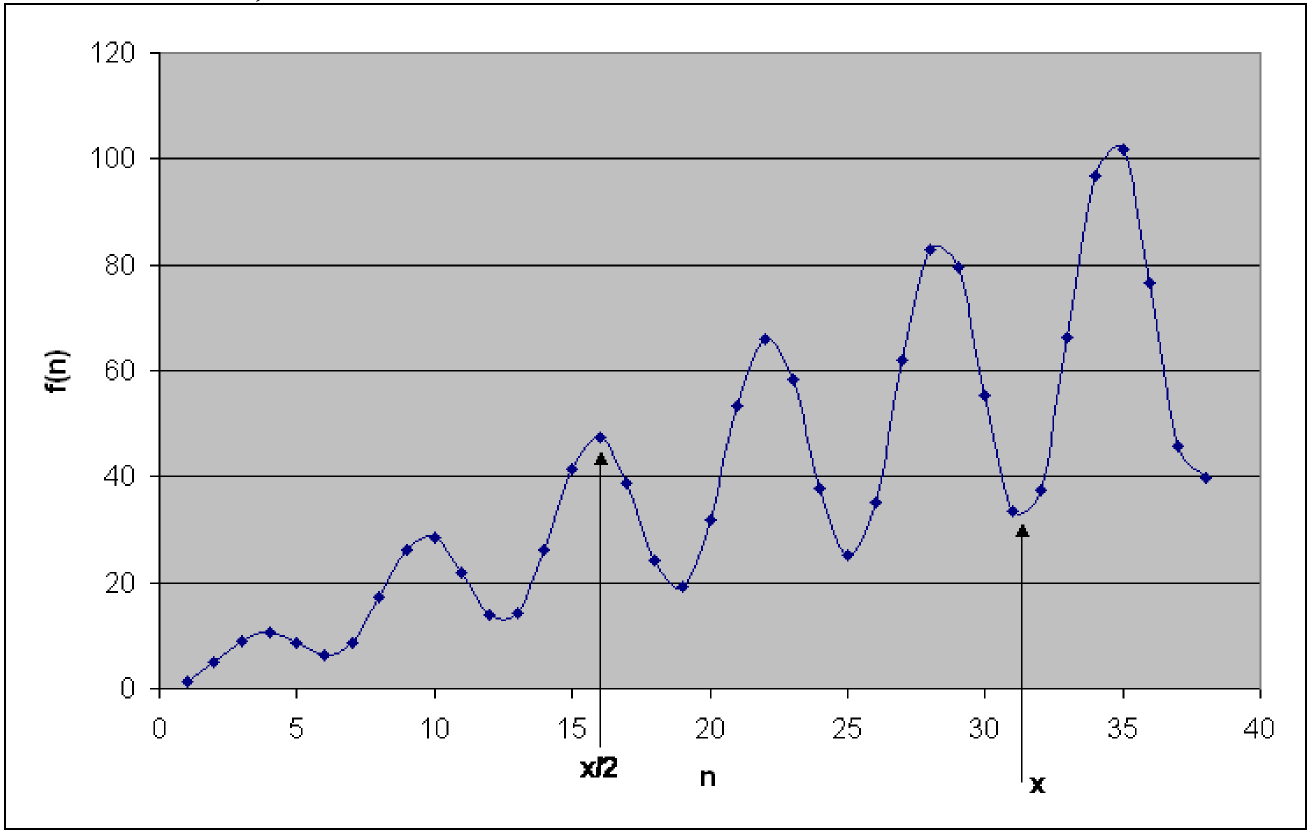
Polinomialmente menor

Regularidade

Condição de regularidade violada

$$T(n) = T\left(\frac{n}{2}\right) + n\left(\sin\left(n - \frac{\pi}{2}\right) + 2\right)$$

$$f(n) = \Omega(n)$$



decom
departamento de computação

Exercícios

- Usando divisão e conquista forneça algoritmos para os itens a seguir e forneça a ordem de complexidade de execução:

  a) Encontrar o maior valor em um vetor

  b) Encontrar o maior e o menor elemento em um vetor

  c) Exponenciação