ALGORITMOS AVANÇADOS

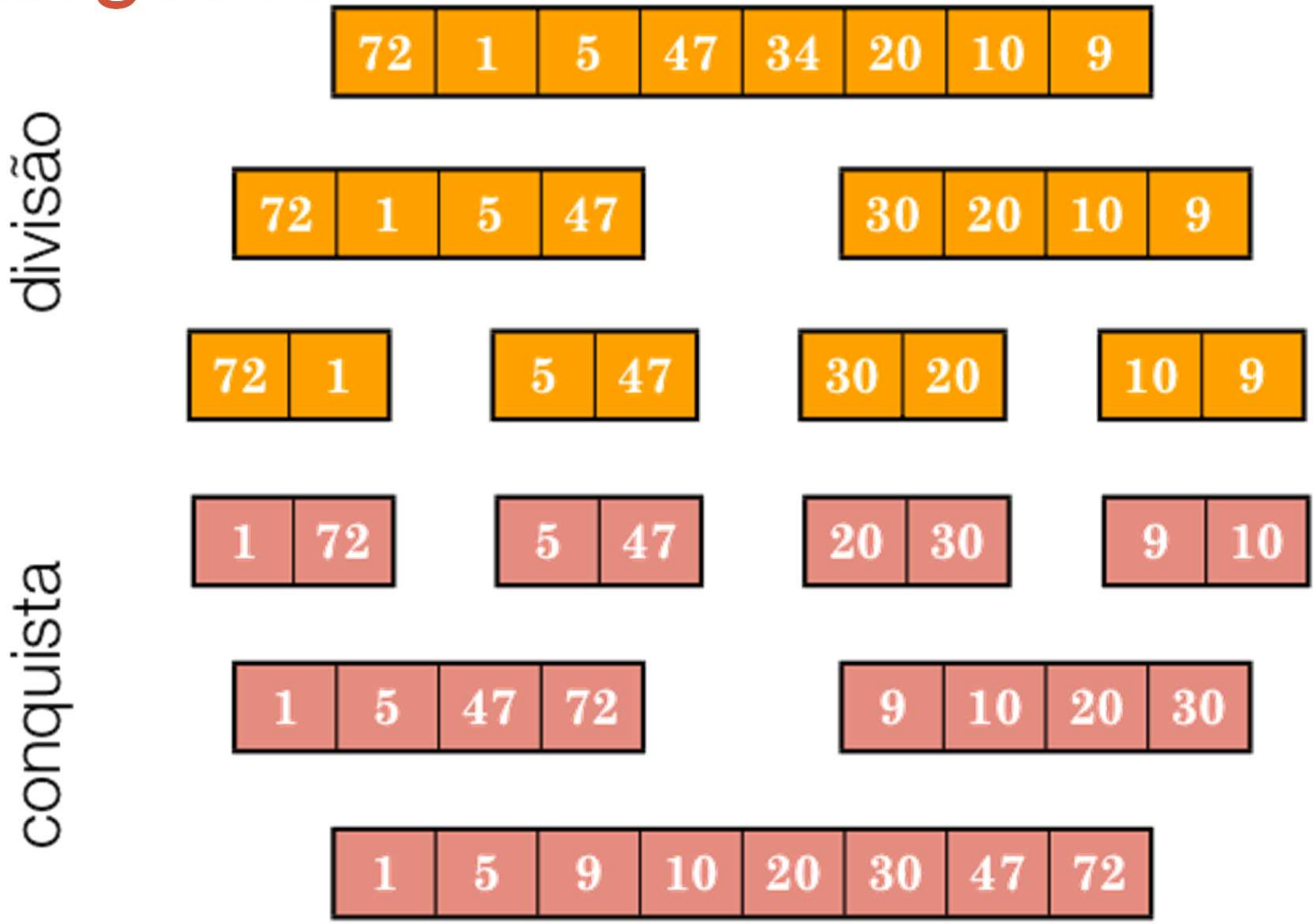
Prof. Michael da Costa Móra

Divisão e Conquista

Motivação

- É preciso revolver um problema com uma entrada grande
- Para facilitar a resolução do problema, a entrada é quebrada em pedaços menores (**DIVISÃO**)
- Cada pedaço da entrada é então tratado separadamente (**CONQUISTA**)
- Ao final, os resultados parciais são combinados para gerar o resultado final procurado

Exemplo típico: Algoritmo Mergesort



A técnica de Divisão e Conquista

A técnica de Divisão e Conquista

A técnica de divisão e conquista consiste de 3 passos:

- **Divisão**: Dividir o problema original em subproblemas menores
- **Conquista**: Resolver cada subproblema recursivamente
- **Combinação**: Combinar as soluções encontradas, compondo uma solução para o problema original

A técnica de Divisão e Conquista

A técnica de Divisão e Conquista

- Algoritmos baseados em divisão e conquista são, em geral, **recursivos**.
- A maioria dos algoritmos de divisão e conquista divide o problema em a subproblemas da mesma natureza, de tamanho n/b
- Vantagens:
  - Requerem um número menor de acessos à memória
  - São altamente paralelizáveis. Se existem vários processadores disponíveis, a estratégia propicia eficiência

Quando utilizar?

Quando utilizar?

- Existem três condições que indicam que a estratégia de divisão e conquista pode ser utilizada com sucesso:
  - Deve ser possível decompor uma instância em sub-instâncias
  - A combinação dos resultados dever ser eficiente (trivial se possível)
  - As sub-instâncias devem ser mais ou menos do mesmo tamanho

Quando utilizar?

Quando utilizar?

- É possível identificar pelo menos duas situações genéricas em que a abordagem por divisão e conquista é adequada:
  - Problemas onde um grupo de operações são correlacionadas ou repetidas. A multiplicação de, matrizes que veremos a seguir, é um exemplo clássico
  - Problemas em que uma decisão deve ser tomada e, uma vez tomada, quebra o problema em peças disjuntas. Em especial, a abordagem por divisão e conquista é interessante quando algumas peças passam a ser irrelevantes

Algoritmo Genérico

Algoritmo Genérico

```
def divisao_e_conquista(x):
    if x é pequeno ou simples:
        return resolve(x)
    else:
        decompor x em n conjuntos menores x_0,x_1,...,x_{n-1}
        for i in [0,1,...,n-1]:
            y_i = divisao_e_conquista(x_i)
        combinar y_0,y_1,...,y_{n-1} em y
        return y
```

Divisão e Conquista: Vantagens

Divisão e Conquista: Vantagens

- Resolução de problemas difíceis
  - Exemplo clássico: Torre de Hanói
- Pode gerar algoritmos eficientes
  - Ótima ferramenta para busca de algoritmos eficientes, com forte tendência a complexidade logarítmica
- Paralelismo
  - Facilmente paralelizável na fase de conquista
- Controle de arredondamentos
  - Em computação aritmética, divisão e conquista traz resultados mais precisos em operações com pontos flutuantes

Divisão e Conquista: Desvantagens

Divisão e Conquista: Desvantagens

- Recursão ou Pilha explícita
- Tamanho da Pilha
  - Número de chamadas recursivas e/ou armazenadas na pilha pode ser um inconveniente
- Dificuldade na seleção dos casos bases
- Repetição de sub-problemas
  - Situação que pode ser resolvida através do uso de **Memoização – Programação Dinâmica**

Maior valor de um vetor

- É possível aplicar Divisão em Conquista para encontrar o maior valor em um vetor?

- Opção 1:

```
int maxVal1(int A[], int n) {
    int max = A[0];
    for (int i = 1; i < n; i++) {
        if( A[i] > max ) max = A[i];
    }
    return max;
}
```

  - Melhor alternativa??

Maior valor de um vetor

Maior valor de um vetor

- Opção 2:

```c
int maxVal2(int A[], int init, int end) {
     if (end - init <= 1)
          return max(A[init], A[end]);
     else {
          int m = (init + end)/2;
          int v1 = maxVal2(A,init,m);
          int v2 = maxVal2(A,m+1,end);
          return max(v1,v2);
     }
}
```

- E agora? Melhorou?

Exponenciação

Exponenciação

- Exponenciação:

```c
int pow1(int a, int n) {
    int p = 1;
    for (int i = 0; i < n; i++)
        p = p * a;
    return p;
}
```

```c
int pow2(int a, int n) {
    if (n == 0)
        return 1;
    if (n % 2 == 0)
        return pow2(a,n/2) * pow2(a,n/2);
    else
        return pow2(a,(n-1)/2) * pow2(a,(n-1)/2) * a;
}
```

Multiplicação de inteiros grandes (bignum)

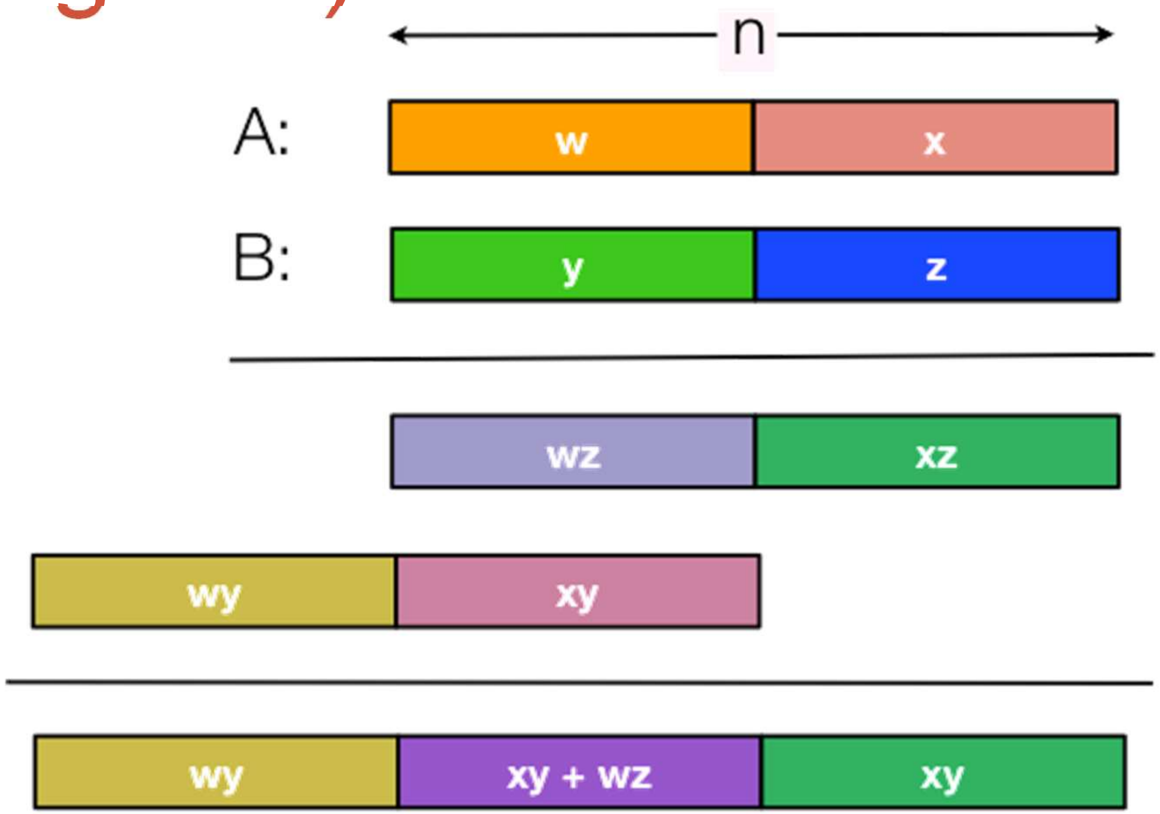
- O problema consiste em multiplicar dois números inteiros grandes (bignum)
- A multiplicação clássica (a que aprendemos a fazer na escola) requer tempo $O(n^2)$. Isso porque fazemos multiplicação dígito a dígito (aula passada).
- Há uma solução alternativa por Divisão e Conquista?

Multiplicação de inteiros grandes (bignum)

- **Sim !!!**

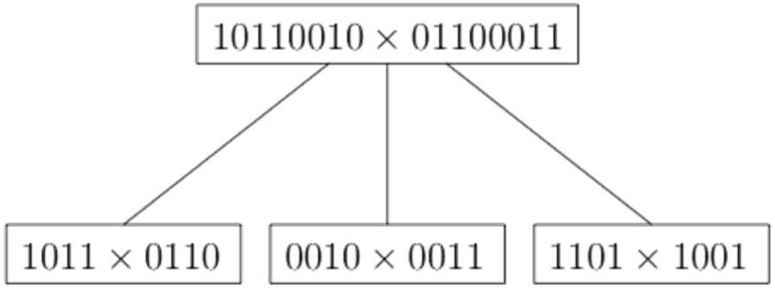
- Solução alternativa por Divisão e Conquista
  - Para evitar maiores complicações, vamos assumir que o número de digitos em cada número é potência de 2
  - A multiplicação de um número A por um número B pode ser efetuada dividindo-se o número original em dois super-dígitos e procedendo a multiplicação

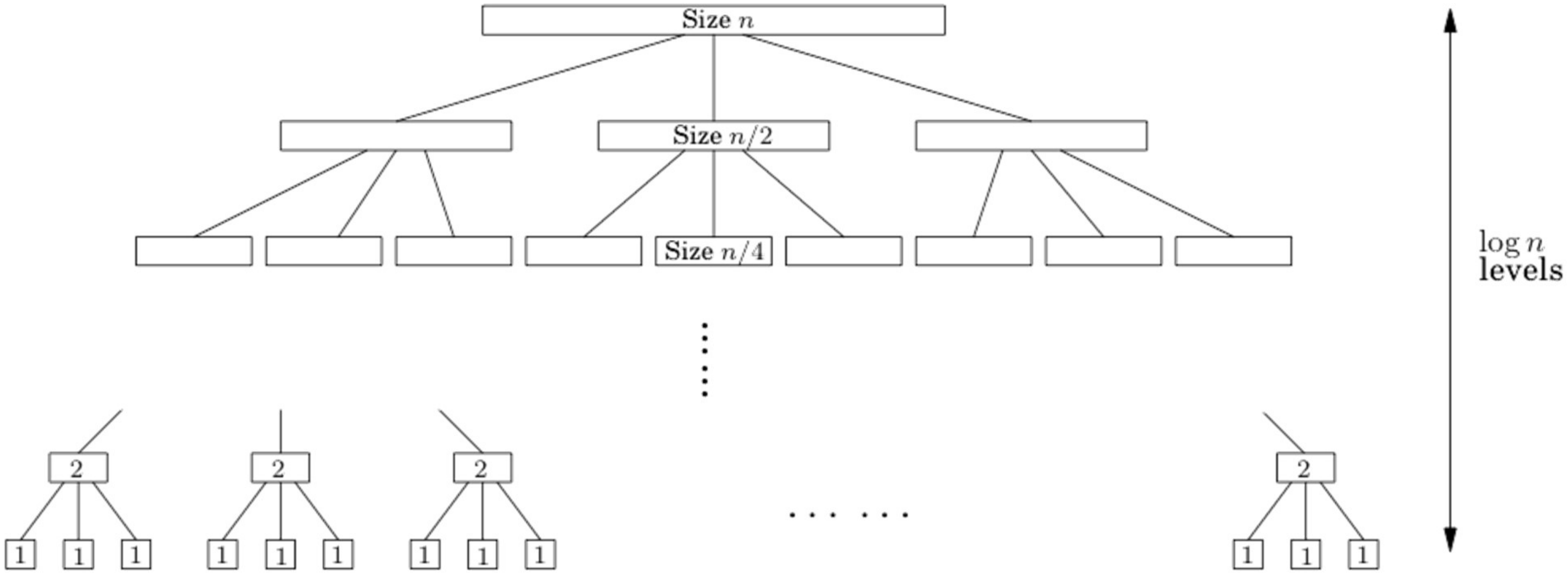
Multiplicação de inteiros grandes (bignum)



$$\text{Mult}(A,B) = 10^{n}\,\text{Mult}(w,y) + 10^{n/2}\,(\text{Mult}(x,z) + \text{Mult}(x,y)) + \text{Mult}(x,z)$$

Multiplicação de inteiros grandes (bignum)





Multiplicação de inteiros grandes (bignum)

- A multiplicação por 10^n deve ser vista como o deslocamento de **n** posições para a direita
- As adições envolvidas tomam tempo O(n) cada
- A multiplicação de dois inteiros longos é o resultado de 4 produtos de inteiros de tamanho duas vezes menor do valor original, e um constante número de adições e deslocamentos, com tempo O(n)

Exemplo: 6514202 x 9898989

Exemplo: 6514202 x 9898989

6514202	$u = p10^4 + q$
9898989	$v = r10^4 + s$
643839	pr
37771778	qs
4853	$p+q$
9978	$r+s$
48423234	$y = (p+q)(r+s)$
10007617	$y-pr-qs$
64484013941778	$uv = pr10^8 + (y-pr-qs)10^4 + qs$

Multiplicação de inteiros grandes (bignum)

Resumindo...

- Multiplicando bignums por Divisão e Conquista:
  - **Divisão**: Dividir cada número em dois números com a metade da quantidade de dígitos
  - **Conquista**: Proceder a multiplicação das quatro partes
  - **Combinação**: Combinar os resultados através dos respectivos deslocamentos e adições