Aula 7 - Divisão e Conquista

Luís Felipe

UFF

19 de Setembro de 2023

Luís Felipe

19/09/23

Teorema Mestre

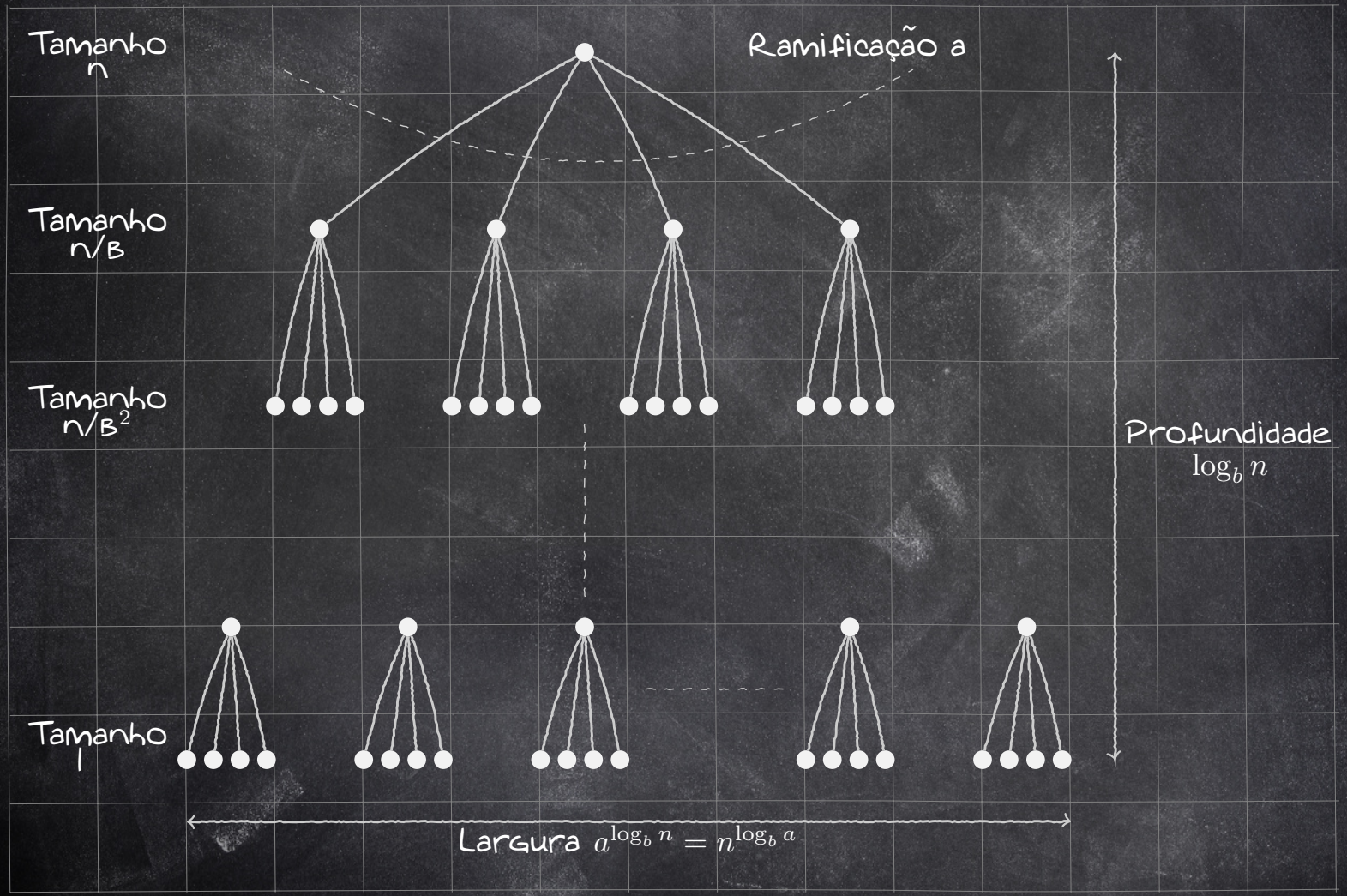
Se $T(n) = a\,T(\lceil \frac{n}{b} \rceil) + O(n^d)$ para constantes $a > 0$, $b > 1$ e $d \geq 0$, então

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

Luís Felipe

19/09/23

Árvore da Recursão



Luís Felipe

19/09/23

Dividindo e conquistando - Busca

- Há algumas aulas, falamos sobre o método da Busca Binária.
- Note que, se T(n) é o número de passos que o algoritmo executa para uma entrada de tamanho n, no pior caso, podemos escrever a seguinte equação:

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

- Vimos que a complexidade do algoritmo de Busca Binária é $O(\log_2 n)$.
- O Teorema Mestre também nos fornece $O(\log_2 n)$ como solução da equação de recorrência acima, pois $\log_2 1 = 0 = d$.

Luís Felipe

19/09/23

Dividindo e conquistando - Ordenação

- Veremos o método Mergesort
- Entrada: Lista com n números
- Saída: Lista ordenada
- Ideia: Dividir a lista ao meio e ordenar recursivamente cada metade. Em seguida, fazer a fusão das sublistas ordenadas.

Luís Felipe

19/09/23

Mergesort

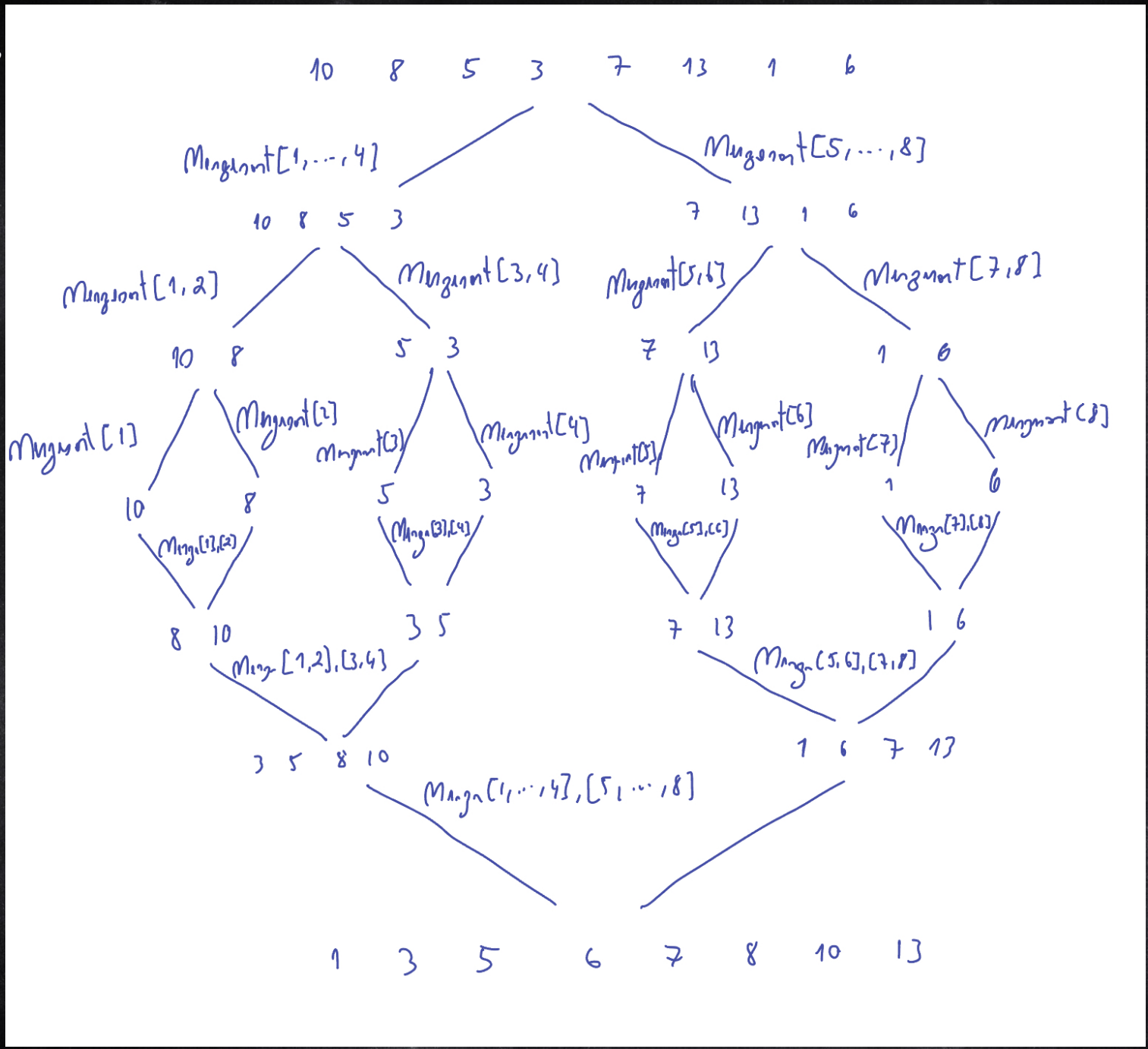
Entrada: Vetor de números $a[1 \ldots n]$
Saída: Vetor ordenado

```
função mergesort (a[1...n])
  Se n > 1 faça
    Retorne merge (mergesort (a[1...⌊n/2⌋]), mergesort ( a[⌊n/2⌋ + 1...n]))
  Senão
    Retorne a

função merge(x[1...k], y[1...ℓ]))
  Se k == 0 faça
    Retorne y[1...ℓ]
  Se ℓ == 0 faça
    Retorne x[1...k]
  Se x[1] ≤ y[1] faça
    Retorne x[1]merge(x[2...k], y[1...l])
  Senão
    Retorne y[1]merge(x[1...k], y[2...l])
```

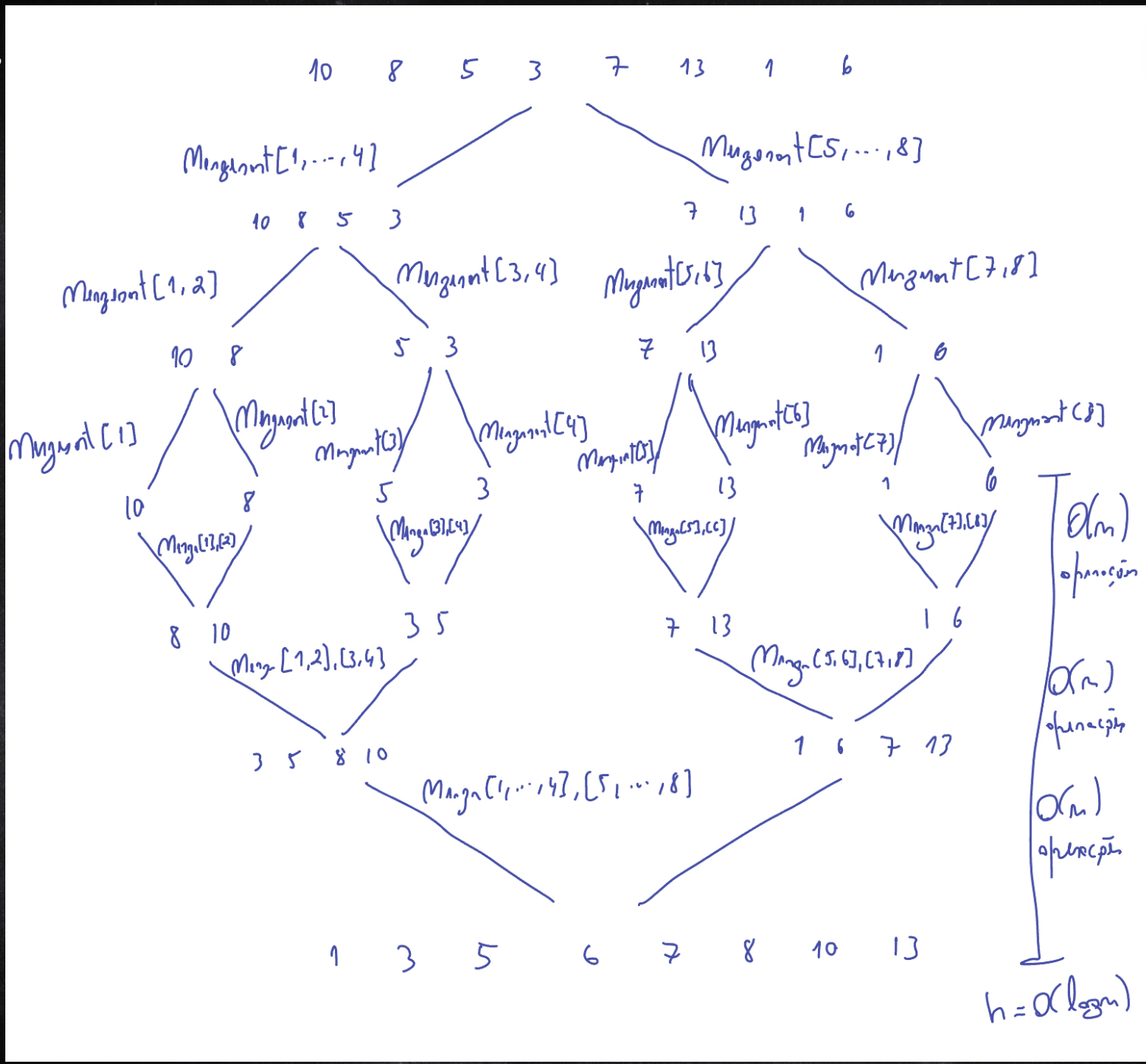
Luís Felipe

19/09/23

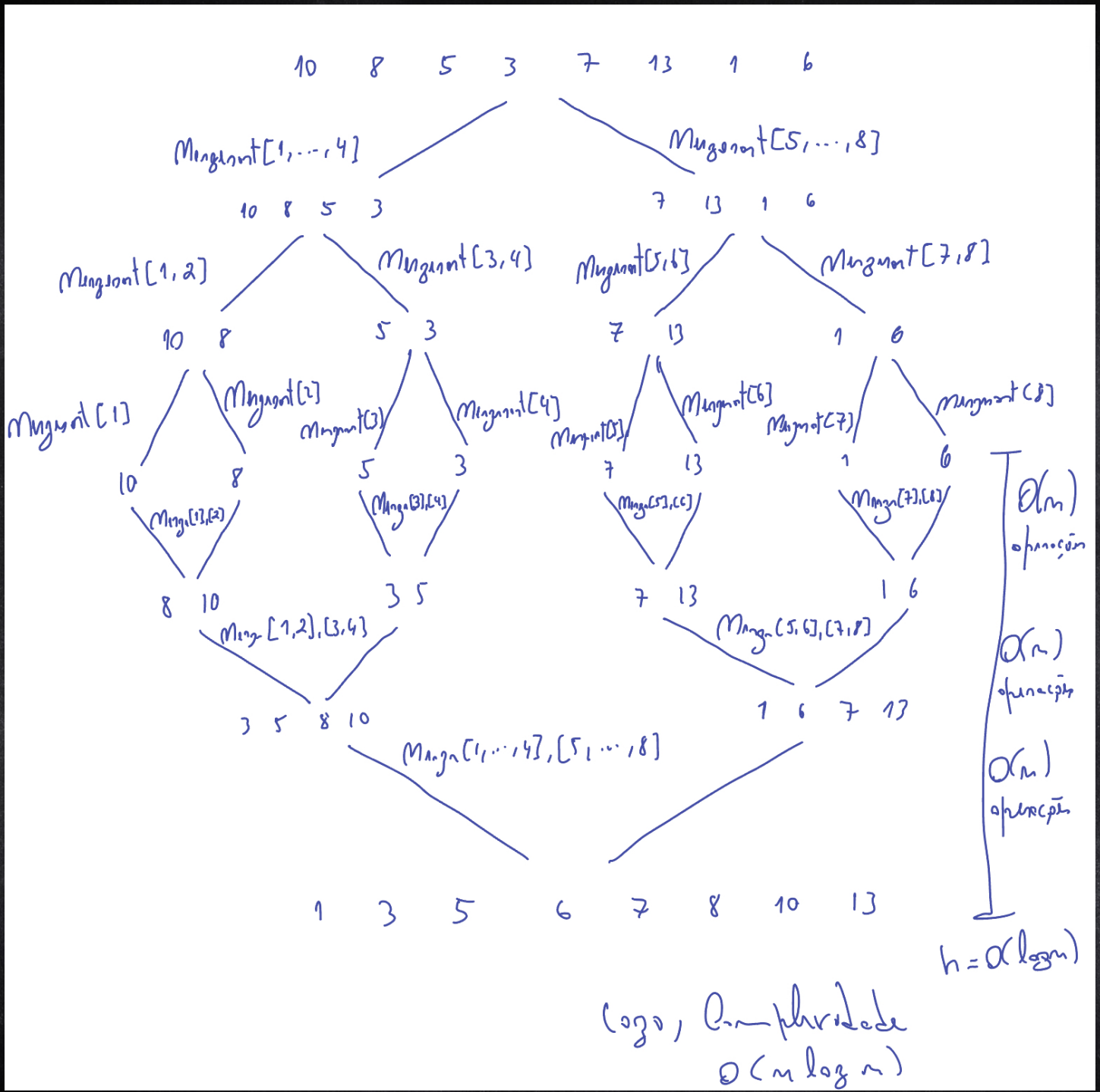


Luís Felipe

19/09/23



Luís Felipe
19/09/23



Logo, Complexidade $O(n \log n)$

Luís Felipe

19/09/23

Complexidade Mergesort

- Uma vez que sempre dividimos nosso problema ao meio e a fusão toma tempo linear, podemos escrever o tempo de execução do Mergesort como:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

- Pelo Teorema Mestre temos que a complexidade do Mergesort é $O(n \log n)$, pois $\log_2 2 = 1 = d$.

Luís Felipe

19/09/23

O Mergesort é ótimo?

- Quando a comparação entre elementos é necessária no algoritmo.
- Desta forma, a computação deste problema pode ser representada por uma árvore binária, chamada de árvore de decisão.
  - As folhas são todas as permutações possíveis para a entrada.
  - Um ramo conduzirá à ordenação final.
  - O número de comparações feitas é justamente a profundidade da árvore.
- De modo geral, $\Omega(n \log n)$ comparações são necessárias em um algoritmo de ordenação.
  - Vamos provar!

Luís Felipe

19/09/23

Limite inferior - ordenação

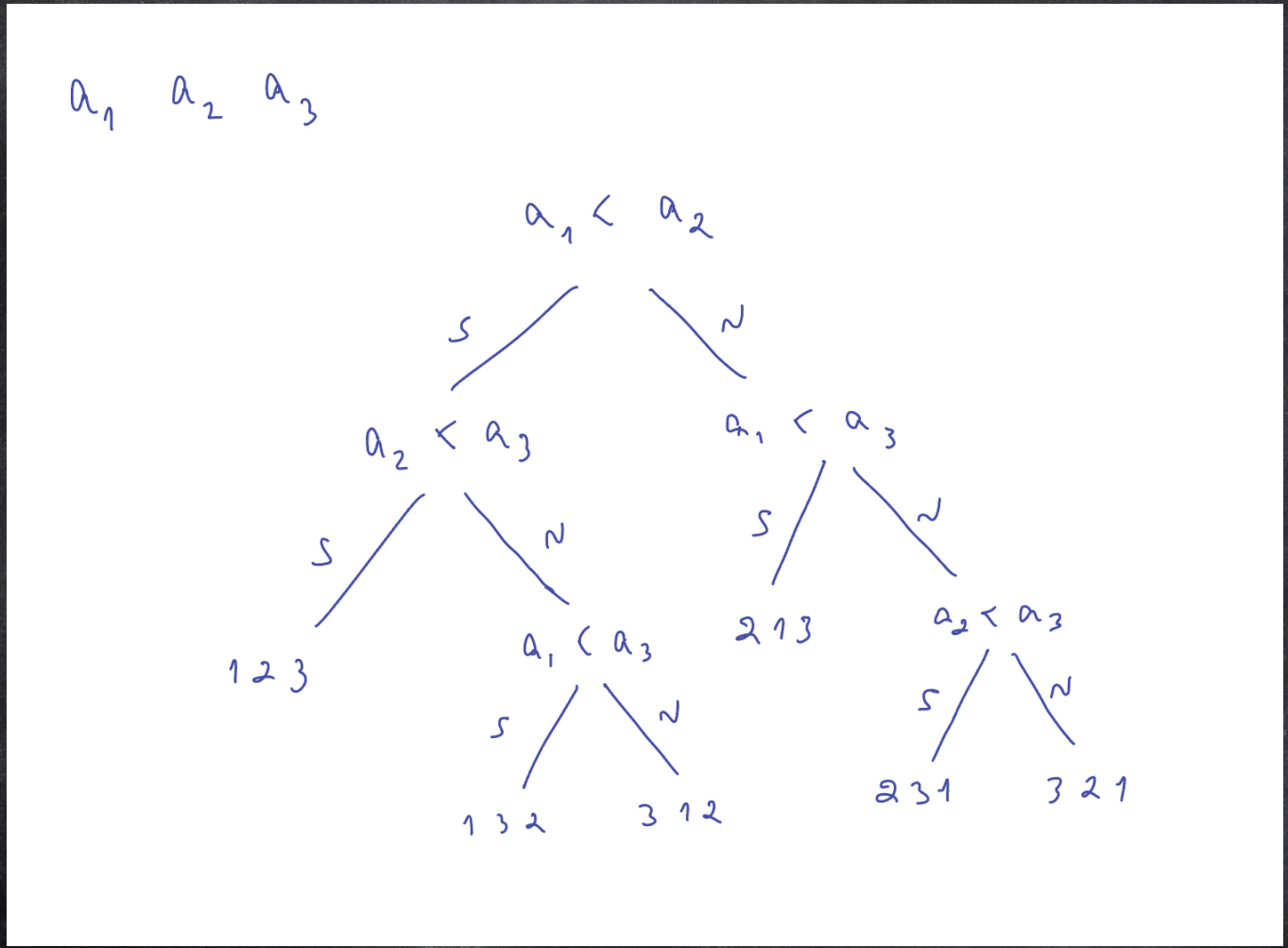
- Um algoritmo que resolva o problema para qualquer entrada tem que ser representado por uma árvore cujas folhas são todas as permutaçõ es possíveis.
  Ou seja, $n!$ folhas.
- Com n! folhas, qual a altura da árvore binária?
  - Uma árvore binária tem 2^d folhas, onde d é a altura da árvore.
  - Logo, $2^d = n!$ e assim, $d = \log_2 n!$
  - $n! \geq \frac{n}{2}^{\frac{n}{2}}$.
    - Pois cada um dos $\frac{n}{2}$ termos de $n!$ (da segunda "metade" de $1.2.\cdots.n$) possui valor maior $\frac{n}{2}$.
  - $d = \log_2 n! \geq \log_2 \frac{n}{2}^{\frac{n}{2}} \geq O(n \log n)$

Mergesort é ótimo

Luís Felipe

19/09/23

Exemplo de árvore de decisão



Luís Felipe

19/09/23

E se "quebrarmos" em outros lugares?

Quicksort (alto nível):

Dada uma tabela L, temos o seguinte procedimento recursivo:

- se $n = 0$ ou $n = 1$ então a tabela está ordenada;
- escolha qualquer elemento x em L (x é o pivô);
- separe $L - \{x\}$ em dois conjuntos de elementos: $S_1 = \{w \in L - \{x\} | w < x\}$; $S_2 = \{w \in L - \{x\} | w > x\}$. O procedimento de ordenação é chamado recursivamente para S_1 e S_2.
- L recebe as concatenações de S_1 com x e com S_2.

Luís Felipe

19/09/23

Diferença pro mergesort

Principal diferença do mergesort:

- Mergesort sempre divide o problema em subproblemas ao meio;
- Quicksort utiliza o conceito de elemento pivô para dividir o problema em subproblemas.

Pergunta: Quem seria um bom pivô? (vamos escalar o melhor pivô?)

- Idealmente, queremos alguém que propicie sublistas de mesma dimensão. Mediana é a solução!!!
  - Mediana de L é um elemento x de L tal que metade dos elementos possuem valores menores ou iguais a x.
  - Simples assim? Não!!! Obter a mediana de uma lista requer $O(n^2)$ no pior caso, sem ter que ordenar a lista.

Luís Felipe

19/09/23

Pausa no Quicksort: Formas para obter mediana

- Ordene e depois obtenha o elemento do meio.
  - $O(n \log n)$ utilizando mergesort.
  - Não faz sentido para o quicksort, uma vez que ordenar é o objetivo do problema.
  - Para obter mediana não precisamos necessariamente ordenar a lista, pois só queremos o elemento que estaria no meio da lista ordenada.

- Resolva o problema mais geral: selecione o k-ésimo menor elemento. Mediana é o $n/2$-ésimo menor.
  - Para algum elemento v da lista, considere três pedaços: S_e, elementos menores do que v; S_v, elementos iguais a v; S_d, elementos maiores do que v.
  - Para saber qual é o k-ésimo elemento, dividimos o problema nesses três possíveis pedaços e resolvemos recursivamente.

Luís Felipe

19/09/23

Resolvendo k-ésimo menor elemento

S : 2 36 5 21 8 13 11 20 5 4 1

S_e : 2 4 1; S_v : 5 5; S_d : 36 21 8 13 11 20.

Se tivéssemos interessados em saber qual o 3-menor elemento de S, saberíamos que ele está em S_e, pois $3 \leq |S_e|$.

Assim, temos um critério para saber qual parte continuamos o procedimento dependendo do tamanho das sublistas.

$$\text{seleção}(S, k) = \begin{cases} \text{seleção}(S_e, k), & k \leq |S_e| \\ v, & |S_e| < k \leq |S_e| + |S_v| \\ \text{seleção}(S_d, k - |S_e| - |S_v|), & k > |S_e| + |S_v| \end{cases}$$

Uma vez escolhendo v, podemos obter S_e, S_v e S_d em $O(n)$.

Continuando a recursão, temos tamanho $\max\{|S_e|, |S_d|\}$.

Agora, como escolher v? Gostaríamos que, idealmente, $|S_e| \approx |S_d| \approx \frac{|S|}{2}$. Para assim, $T(n) = T(\frac{n}{2}) + O(n)$.

Mas assim, v seria a mediana. Ou seja, muita sorte, nada real.

Luís Felipe

19/09/23

Tipos de escolha do elemento v

Melhor caso: A escolha de v faz com que diminuamos pela metade o tamanho, $T(n) = T(\frac{n}{2}) + O(n) = O(n)$ (Aplique Teorema Mestre)

Pior caso: A escolha de v faz com que diminuamos a lista para alguma de maior tamanho possível, $T(n) = T(n-1) + O(n) = O(n^2)$ (Aplique Substituições Regressivas)

Apesar do pior caso ter complexidade quadrática, é esperado que $T(n) = O(n)$ ao escolher v aleatoriamente. (trabalho para casa! Obtenha a complexidade de caso médio. Dica: Suponha probabilidade uniforme $1/n$ e cada tipo de entrada requer n passos)

Luís Felipe

19/09/23

Voltando ao Quicksort!!

Vimos que obter mediana para pivô pode não ser uma boa alternativa, apesar de possuir um tempo esperado bom quando feito aleatoriamente.

Outros pivôs:

- Usualmente, tomamos o primeiro elemento de L como o pivô.
  - Horrível, se a entrada estiver em ordem decrescente;
  - Aceitável, se a entrada for aleatória.
- Alternativamente, tomamos um dentre três elementos: o primeiro, o último e o central.

Luís Felipe

19/09/23

Um pouco mais a baixo nível

Pergunta: Como fazer a separação de L em S_1 e S_2?

OBS.: Considere todos elementos distintos.

1. Escolha o pivô x;
2. Afaste o pivô de L (ponha x após a última posição de L)
3. Utilize dois ponteiros i e j:
   - i é inicializado apontando para o primeiro elemento de L;
   - j é inicializado apontando para o último elemento de L.
4. Incrementamos a posição de i enquanto os elementos de L são menores que pivô x;
5. Decrementamos a posição de j enquanto os elementos de L são maiores que pivô x.

Luís Felipe

19/09/23

E quando pararmos?

- Ao sair do "enquanto", duas situaçõ es podem ocorrer:
  i) Se $i < j$, os elementos de L devem ser trocados e prosseguimos;
  ii) Se $i > j$, a posição está determinada. Troque o elemento com índice i com o pivô x.

- Consequência: Todos os elementos a esquerda de x serão menores do que x e todos os elementos a direita de x serão maiores do que x.

Luís Felipe

19/09/23

Vamos passar uma vez?

Exemplo:

40 37 95 42 23 51 27

40 37 95 42 23 51 27

40 37 95 42 23 51 27

pivô

27 37 95 42 23 51 40

troquei 27 com 40

27 37 95 42 23 51 40

i j

27 37 95 42 23 51 40

i j

27 37 95 42 23 51 40

i j

27 37 23 42 95 51 40

j i

27 37 23 40 95 51 42

troquei 40 com 42

27 37 23 40 95 51 42

Faça o mesmo em cada parte

Luís Felipe

19/09/23

Algoritmo quicksort

```
procedimento quicksort(ini,fim)
    se fim-ini < 2 então
        se fim - ini = 1 então
            se L[ini].chave > L[fim].chave então
                trocar(L[ini], L[fim]);
    senão PIVO(ini, fim, mediana)
        trocar(L[mediana], L[fim])
        i:= ini j:= fim-1 key:=L[fim].chave
        enquanto j >= i faça
            enquanto L[i].chave < key faça
                i := i+1
            enquanto L[j].chave > key faça
                j := j-1
            se j >= i então
                trocar(L[i], L[j])
                i := i+1; j:= j-1
        trocar(L[i], L[fim])
        quicksort(ini, i-1)
        quicksort(i+1, fim)
quicksort(1,n)
```

Luís Felipe

19/09/23

Análises do Algoritmo

Como visto inicialmente, o Quicksort e o Mergesort têm bastante semelhança. Assim:

- O tempo de execução do quicksort para uma lista de tamanho n é $T(n)$;
- $T(n)$ é o resultado da soma dos tempos da execução das duas chamadas recursivas.

Considerando o pivô escolhido aleatoriamente, temos $T(0) = T(1) = 1$, e:

$$T(n) = T(i) + T(n - i - 1) + cn$$

i é o número de elementos de S_1 e c é uma constante.

Luís Felipe

19/09/23

Análise de pior caso

- Pior caso ocorre quando o pivô é o menor elemento.
  Assim, $i = 0$, e como $T(0) = 1$:
  $T(n) = T(n-1) + cn,\ n > 1$

  Podemos resolver esta recorrência pelo Método das Substituições Regressivas:

$$\begin{aligned} T(n) &= T(n-1) + cn \\ &= T(n-2) + c(n-1) + cn \\ &= T(1) + c\sum_{i=2}^{n} i \\ &= O(n^2) \end{aligned}$$

Luís Felipe

19/09/23

Melhor caso

- Melhor caso ocorre quando o pivô é a mediana sem elementos repetidos:
  $T(0) = 1$;
  $T(n) = 2T(\frac{n}{2}) + cn,\ n > 1$

  Exatamente a mesma recorrência obtida no algoritmo mergesort, ou seja:

$$T(n) \quad = \quad O(n \log n)$$

OBS.: Relembre o Teorema Mestre.

Luís Felipe

19/09/23

Caso Médio

- Assumindo que cada possível tamanho das sublistas tem igual probabilidade, essa probabilidade é: $1/n$.

$$\begin{aligned} T(n) &= \frac{2}{n}\sum_{j=0}^{n-1} T(j) + cn \\ nT(n) &= 2\sum_{j=0}^{n-1} T(j) + cn^2 \end{aligned}$$

Ou ainda:

$$(n-1)T(n-1) = 2\sum_{j=0}^{n-2} T(j) + c(n-1)^2$$

Efetuando a diferença das duas últimas expressões:

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c$$

Ou ainda:

$$nT(n) = (n+1)T(n-1) + 2cn$$

Luís Felipe

19/09/23

Caso Médio (cont.)

$$nT(n) = (n+1)T(n-1) + 2cn$$

Agora sim, temos uma fórmula recursiva de $T(n)$ em função de $T(n-1)$. dividimos por $n(n+1)$ e repetindo sucessivamente:

$$\begin{aligned}
\frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + \frac{2c}{n+1} \\
\frac{T(n-1)}{n} &= \frac{T(n-2)}{n-1} + \frac{2c}{n} \\
\frac{T(n-2)}{n-1} &= \frac{T(n-2)}{n-2} + \frac{2c}{n-1} \\
&\vdots \\
\frac{T(2)}{3} &= \frac{T(1)}{2} + \frac{2c}{3}
\end{aligned}$$

Ou seja: $\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c\sum_{i=3}^{n+1}\frac{1}{j}$. Dessa forma: $\frac{T(n)}{n+1} = O(\log n)$

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

Luís Felipe

19/09/23

Mergesort ou Quicksort?

Mergesort:

- Complexidade de melhor caso: $O(n \log n)$
- Complexidade de caso médio: $O(n \log n)$
- Complexidade de pior caso: $O(n \log n)$

Quicksort:

- Complexidade de melhor caso: $O(n \log n)$
- Complexidade de caso médio: $O(n \log n)$
- Complexidade de pior caso: $O(n^2)$

Pergunta: Então por que o quicksort recebeu o nome de "o rápido"?

- Quicksort não precisa de um vetor auxiliar durante sua execução, diferentemente do mergesort.
- Na prática, o quicksort se torna mais eficiente.

Luís Felipe

19/09/23

Dividindo e conquistando - Matrizes

- Há algumas aulas vimos um algoritmo de multiplicação de matrizes com complexidade $O(n^3)$.
- Vamos ver o Algoritmo de Strassen, com complexidade $O(n^{2.81})$.
- Ideia:
  - subdividir cada matriz em 4 blocos de tamanho $\frac{n}{2}$,
  - cada bloco é considerado um elemento,
  - As matrizes são multiplicadas, e
  - O resultado é obtido de maneira recursiva.

Luís Felipe

19/09/23

Estratégia I

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Esse processo pode ser descrito pela seguinte equação:

$$T(n) = 8\,T\left(\frac{n}{2}\right) + O(n^2)$$

Pelo Teorema Mestre, a complexidade é $O(n^3)$

Luís Felipe

19/09/23

Será que dá para melhorar?

- Assim como no problema da multiplicação de dois inteiros, a estratégia é reduzir o número de multiplicações
- Strassen conseguiu reduzir o número de multiplicações de 8 para 7 de maneira intrigante.

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix} \text{ onde}$$

$$\begin{aligned} P_1 &= A(F - H) \\ P_2 &= (A + B)H \\ P_3 &= (C + D)E \\ P_4 &= D(G - E) \end{aligned}$$

$$\begin{aligned} P_5 &= (A + D)(E + H) \\ P_6 &= (B - D)(G + H) \\ P_7 &= (A - C)(E + F) \end{aligned}$$

Luís Felipe

19/09/23

Complexidade do Algoritmo de Strassen

- Podemos escrever a seguinte equação:
  $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$
- Pelo Teorema mestre, a complexidade do algoritmo de Strassen é $O(n^{\log_2 7}) \approx O(n^{2.81})$