ACH2002

Aula 10

Técnicas de Desenvolvimento de Algoritmos - **Divisão e Conquista (equações de recorrência)**

(adaptados dos slides de aula da Profa. Fátima L. S. Nunes)

Aula passada

- Algoritmo de ordenação Mergesort (ordenação por **intercalação**)
  - Versão recursiva (divisão e conquista) – $\Theta(n \lg n)$
  - Versão iterativa – $\Theta(n \lg n)$

- Equações de recorrência

Paradigma Dividir e conquistar

- Chamando o tempo de **dividir** e **combinar** de D(n) e C(n), respectivamente:
- T(n) = aT(n/b) + D(n) + C(n)
- Considerando que para ***n*** suficiente pequeno T(n) = O(1) (quando paramos de dividir)

$$T(n) = \begin{cases} O(1), n \leq c \\ aT(n/b) + D(n) + C(n), caso contrário \end{cases}$$

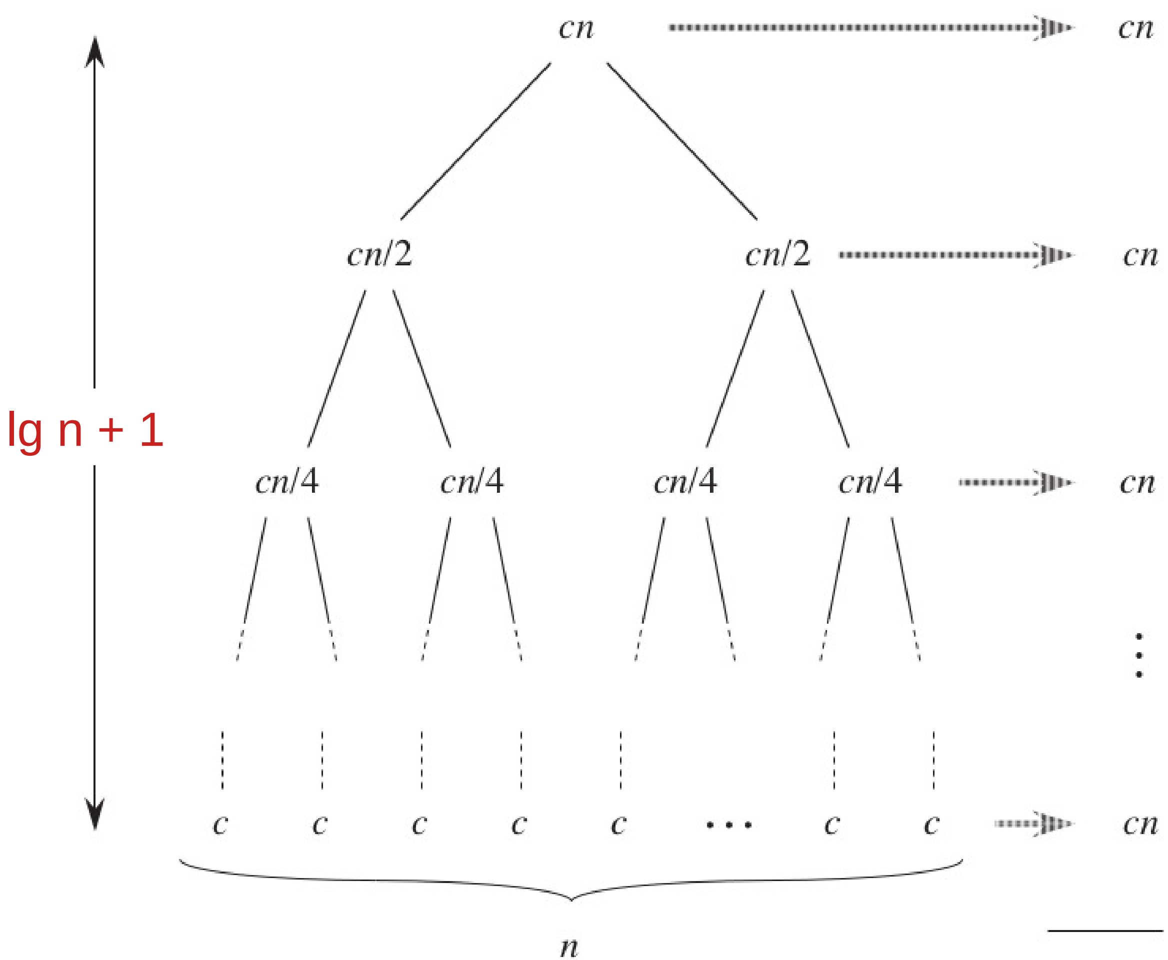
$$T(n) = \begin{cases} O(1), n \leq c \\ aT(n/b) + f(n), caso contrário \end{cases}$$

f(n)=D(n)+C(n)

Complexidade de tempo do MergeSort (cálculo “por intuição”)

$$T(n) = \begin{cases} c & \text{if } n = 1\,, \\ 2T(n/2) + cn & \text{if } n > 1\,, \end{cases}$$

```
mergeSort (A,p,r)
se p < r
  q ← ⌊(p+r)/2⌋
  mergeSort(A, p, q)
  mergeSort(A, q+1,r)
  merge(A,p,q,r)
```



(d)

Para dividir e combinar

Para dividir e combinar

Para dividir e combinar

Para conquistar o caso base

Total: cn (lg n + 1) = O(n lg n)

Aula de hoje

- Versão recursiva da busca binária (divisão e conquista)

- Resolução de equações de recorrência (não só para quando os subproblemas são disjuntos, mas para toda solução recursiva)

Complexidade de tempo do MergeSort

```
mergeSort (A,p,r)
se p < r
  q ← ⌊(p+r)/2⌋
  mergeSort(A, p, q)
  mergeSort(A, q+1,r)
  merge(A,p,q,r)
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

E quanto é isso afinal?

Veremos técnicas para resolver equações de recorrências,

mas esta dá para resolver intuitivamente...

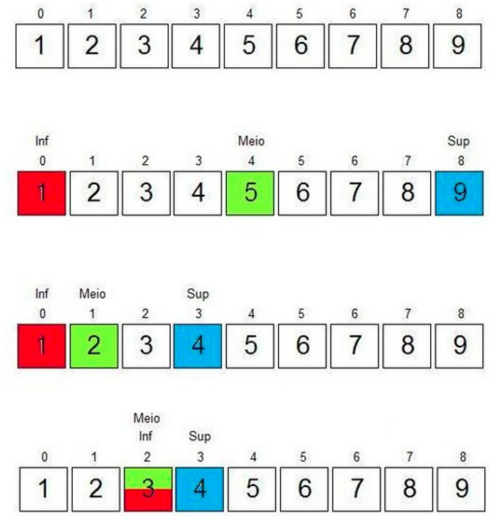
Então vamos ver agora!

Primeiro: vamos treinar escrever as equações de recorrência para os algoritmos

Como seria a versão recursiva do algoritmo de busca binária?

A solução é também do tipo "dividir e conquistar"

Busca binária - versão iterativa (para lembrarmos)



Ex: busca do nr 3
[encurtador.com.br/uvBCO0]

```
// Busca binaria em lista ordenada
int buscaBin(TIPOCHAVE ch, LISTA l)
{
    int inf, sup, meio;
    inf = 0;
    sup = l.nroElem - 1;
    while(inf <= sup)
    {
            meio = ((inf + sup) / 2);
            if(l.A[meio].chave == ch) return(meio);  // achou
            else
            {
                if(l.A[meio].chave < ch) inf = meio + 1;
                else sup = meio - 1;
            }
    }
    return(-1);
}
```

Busca binária - versão recursiva do tipo “dividir e conquistar”

```
int buscaBinaria(int valor, int[] vetor, int esq, int dir){
    int meio;
    if (esq <= dir) {
        meio = (esq + dir) / 2;
        if (valor == vetor[meio]) return meio;
        if (valor < vetor[meio]) {
            dir = meio - 1;
            return binaria(valor, vetor, esq, dir);
        } else {  /* valor > vetor[meio] */
            esq = meio + 1;
            return binaria(valor, vetor, esq, dir);
        }
    } else return -1; /* retorna -1 se o valor nao for encontrado */
}
```

Note como os subproblemas são disjuntos!

Complexidade de algoritmos recursivos

- Como calcular a complexidade temporal de algoritmos recursivos?
  - devemos considerar as chamadas para o próprio método
  - já sabemos que a abordagem *dividir e conquistar* gera T(n) como uma equação de recorrência.
  - precisamos analisar o algoritmo e definir a forma de T(n), de acordo com este conceito.

```
int buscaBinaria(int valor, int[] vetor, int esq, int dir){
   int meio;
   if (esq <= dir) {
      meio = (esq + dir) / 2;
      if (valor == vetor[meio]) return meio;
      if (valor < vetor[meio]) {
         dir = meio - 1;
         return binaria(valor, vetor, esq, dir);
      } else {  /* valor > vetor[meio] */
           esq = meio + 1;
           return binaria(valor, vetor, esq, dir);
      }
   } else return -1; /* retorna -1 se o valor nao for encontrado */
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com vetor[meio]
  - Quando n = 1, quanto vale T(n)?

    T(n) = 2 comparações >>>> Se não for igual, comparo se menor
  - Quando n > 1, quanto vale T(n)? Exemplo de quando $a \neq b$
    - T(n) = T(n/2) + O(1) >>>> Sigo só um ramo da bifurcação

```
int buscaBinaria(int valor, int[] vetor, int esq, int dir){
   int meio;
   if (esq <= dir) {
      meio = (esq + dir) / 2;
      if (valor == vetor[meio]) return meio;
      if (valor < vetor[meio]) {
         dir = meio - 1;
         return binaria(valor, vetor, esq, dir);
      } else {  /* valor > vetor[meio] */
          esq = meio + 1;
          return binaria(valor, vetor, esq, dir);
      }
   } else return -1; /* retorna -1 se o valor nao for encontrado */
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com vetor[meio]
  - Quando n = 1, quanto vale T(n)?

    T(n) = 2 comparações >>>> Se não for igual, comparo se menor
  - Quando n > 1, quanto vale T(n)? ⟶ Exemplo de quando a ≠ b
    - T(n) = T(n/2) + O(1) >>>> Sigo só um ramo da bifurcação

```
int buscaBinaria(int valor, int[] vetor, int esq, int dir){
   int meio;
   if (esq <= dir) {
      meio = (esq + dir) / 2;
      if (valor == vetor[meio]) return meio;
      if (valor < vetor[meio]) {
         dir = meio - 1;
         return binaria(valor, vetor, esq, dir);
      } else {  /* valor > vetor[meio] */
         esq = meio + 1;
         return binaria(valor, vetor, esq, dir);
      }
   } else return -1; /* retorna -1 se o valor nao for encontrado */
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com vetor[meio]
  - Quando n = 1, quanto vale T(n)?

    T(n) = 2 comparações >>>> Se não for igual, comparo se menor
  - Quando n > 1, quanto vale T(n)?

    Pelo cáculo “intuitivo”: O(lg n))
    - T(n) = T(n/2) + O(1) >>>> Sigo só um ramo da bifurcação

```
int buscaBinaria(int valor, int[] vetor, int esq, int dir){
   int meio;
   if (esq <= dir) {
      meio = (esq + dir) / 2;
      if (valor == vetor[meio]) return meio;
      if (valor < vetor[meio]) {
         dir = meio - 1;
         return binaria(valor, vetor, esq, dir);
      } else {  /* valor > vetor[meio] */
          esq = meio + 1;
          return binaria(valor, vetor, esq, dir);
      }
   } else return -1; /* retorna -1 se o valor nao for encontrado */
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com vetor[meio]
  - Quando n = 1, quanto vale T(n)?

    T(n) = 2 comparações >>>> Se não for igual, comparo se menor
  - Quando n > 1, quanto vale T(n)?
    - T(n) = T(n/2) + O(1) >>>> Sigo só um ramo da bifurcação

Vamos aprender a achar a complexidade de outra forma: expandindo a recorrência

```
int buscaBinaria(int valor, int[] vetor, int esq, int dir){
   int meio;
   if (esq <= dir) {
      meio = (esq + dir) / 2;
      if (valor == vetor[meio]) return meio;
      if (valor < vetor[meio]) {
         dir = meio - 1;
         return binaria(valor, vetor, esq, dir);
      } else {  /* valor > vetor[meio] */
           esq = meio + 1;
           return binaria(valor, vetor, esq, dir);
      }
   } else return -1; /* retorna -1 se o valor nao for encontrado */
}
```

Equações de recorrência

Equações de recorrência

- Como encontrar a complexidade desta equação (resolver a equação)?

$$T(n) = \begin{cases} O(1), se\, n = 1 \\ T(n/2) + O(1), caso\, contrário \end{cases}$$

Equações de recorrência

- Como encontrar a complexidade desta equação (resolver a equação)?

$$T(n) = \begin{cases} O(1), se\, n = 1 \\ T(n/2) + O(1), caso\, contrário \end{cases}$$

- Vamos tentar expandi-la até achar a "cara" dela em função do i (nr de divisões):

$$T(n) = T(n/2) + O(1)$$
$$T(n) = T(n/4) + O(1) + O(1)$$
$$T(n) = T(n/8) + O(1) + O(1) + O(1)$$
$$\ldots$$
$$T(n) = T(n/2^i) + i * O(1)$$

Preciso achar o i que faz o parâmetro de T cair no caso base da equação de recorrência (neste exemplo n = 1), para poder parar de expandi-la e calcular finalmente seu valor

Quando (para qual i) $n/2^i = 1$?

$2^i = n \quad \Rightarrow \quad i = \log_2 n$

Equações de recorrência

$$T(n) = T(n/2) + O(1)$$
$$T(n) = T(n/4) + O(1) + O(1)$$
$$T(n) = T(n/8) + O(1) + O(1) + O(1)$$

$$T(n) = \begin{Bmatrix} O(1), se n = 1 \\ T(n/2) + O(1), caso contrário \end{Bmatrix}$$

...

$$T(n) = T(n/2^i) + i * O(1)$$

Até i = $\log_2 n$

Substituindo $T(n/2^i)$ por T(1) na última equação de T(n), temos: T(n) = T(1) i*O(1)

Mas T(1) = O(1) (definida na base da equação)

=> T(n) = (i+1) * O(1)

Usando operação f(n) * O(g(n)) = O(f(n)g(n)) da notação O, temos:

$$T(n) = O((i+1) * 1)$$

que é igual a:

$$T(n) = O(i)$$

Lembrando que i = $\log_2 n$:

$$T(n) \in O(\log_2 n)$$

Equações de recorrência

Busca sequencial recursiva

- Pior caso? valor não está no *array*
- T(n) para n = **0**? O(1) → Note que neste caso a base é n = 0 !!!!
- T(n) para n > 0? T(n-1) + O(1)

Recebe x, v e $n \geq 0$ e devolve k tal que $0 \leq k < n$ e $v[k] = x$. Se tal k não existe, devolve -1.

```
int BuscaR (int x, int v[], int n) {
    if (n == 0) return -1;
    if (x == v[n-1]) return n - 1;
    return BuscaR (x, v, n - 1);
}
```

Equações de recorrência

Busca sequencial recursiva

$$T(n)=\begin{cases} O(1), se\ n=0 \\ T(n-1)+O(1), caso\ contrário \end{cases}$$

$$T(n)=T(n-1)+O(1)$$

$$T(n)=T(n-2)+O(1)+O(1)$$

...

$$T(n)=T(n-i)+i*O(1)$$

```
int BuscaR (int x, int v[], int n) {
    if (n == 0) return -1;
    if (x == v[n-1]) return n - 1;
    return BuscaR (x, v, n - 1);
}
```

Quando n – i = 0 ?

=> i = n

T(n) = T(0) + n * O(1)

Mas T(0) = O(1)

T(n) = O(1) + n * O(1)

T(n) = (n+1) * O(1)

Portanto: $T(n) \in O(n)$

Equações de recorrência

- Torres de Hanói
  - O jogo **Torres de Hanói** é um quebra-cabeça que consiste em um conjunto de *N* discos de tamanhos diferentes e 3 pinos verticais, nos quais os discos devem ser encaixados.
  - Cada pino pode conter uma pilha com qualquer quantidade de discos, desde que cada disco não seja colocado sobre outro disco menor.
  - Configuração inicial: todos os discos no pino 1.
  - Objetivo: mover todos os discos para um dos outros discos, sempre obedecendo a restrição de não colocar um disco sobre outro menor.

Alguns métodos recursivos (Java)

- Torres de Hanói

n = nro

```
void hanoi(char ori, char dst, char aux, int nro) {
   if(nro == 1) {
      System.out.print("Move de " + ori);
      System.out.println(" para " + dst);
   }
   else {
      hanoi(ori, aux, dst, nro-1);
      hanoi(ori, dst, aux, 1);
      hanoi(aux, dst, ori, nro-1);
   }
}
```

Equações de recorrência

Torre de Hanói recursiva

$$T(n) = \begin{cases} O(1), \; se \; n = 1 \\ 2T(n-1) + O(1), \; caso \; contrário \end{cases}$$

$$T(n) = 2T(n-1) + O(1)$$
$$T(n) = 4T(n-2) + (2+1) * O(1)$$
$$T(n) = 8T(n-3) + (4+2+1) * O(1)$$
$$\ldots$$
$$T(n) = 2^i T(n-i) + (2^i - 1) * O(1)$$

Soma da PG

$n - i = 1 \Rightarrow i = n - 1$

$T(n) = 2^{n-1} * T(1) + (2^{n-1} - 1) * O(1)$

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

$T(n) = (2^{n-1+1} - 1) * O(1)$

$T(n) = (2^n - 1) * O(1)$

Portanto: $T(n) \in O(2^n)$

n = nro

```
void hanoi(char ori, char dst, char aux, int nro) {
   if(nro == 1) {
      System.out.print("Move de " + ori);
      System.out.println(" para " + dst);
   }
   else {
      hanoi(ori, aux, dst, nro-1);
      hanoi(ori, dst, aux, 1);
      hanoi(aux, dst, ori, nro-1);
   }
}
```

Equações de recorrência

Torre de Hanói recursiva

$$T(n)=\begin{cases} O(1),\ se\ n=1 \\ 2T(n-1)+O(1),\ caso\ contrário \end{cases}$$

$$T(n)=2T(n-1)+O(1)$$

$$T(n)=4T(n-2)+(2+1)*O(1)$$

$$T(n)=8T(n-3)+(4+2+1)*O(1)$$

...

$$T(n)=2^iT(n-i)+(2^i-1)*O(1)$$

Soma da PG

Note o que acontece com recursividade quando tenho mais de uma chamada em subproblemas não disjuntos...

n – i = 1 => i = n -1

$T(n) = 2^{n-1}*T(1) +(2^{n-1} -1)*O(1)$

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

$T(n) = (2^{n-1+1} - 1) *O(1)$

$T(n) = (2^n - 1) *O(1)$

Portanto: $T(n) \in O(2^n)$

n = nro

```
void hanoi(char ori, char dst, char aux, int nro) {
   if(nro == 1) {
      System.out.print("Move de " + ori);
      System.out.println(" para " + dst);
   }
   else {
      hanoi(ori, aux, dst, nro-1);
      hanoi(ori, dst, aux, 1);
      hanoi(aux, dst, ori, nro-1);
   }
}
```

Equações de recorrência

- Já vimos como escrever T(n) na forma de uma equação de recorrência, a partir da análise do algoritmo.
- Também já sabemos resolver equações de recorrência para algumas casos, considerando sua expansão, isto é, obter limites assintóticos para T(n).

- O problema é:
  - É trivial expandir equações de recorrência? Mais ou menos...
  - Sempre é possível resolver equações de recorrência com expansão? NÃO!!!
  - Há outras formas para encontrar a complexidade assintótica de equações de recorrência

Equações de recorrência

- Há basicamente 3 outros métodos para resolver equações de recorrência
  - Método de substituição
  - Árvore de recursão – não veremos formalmente este (ver exemplo do Mergesort- Sugestão: livro Cormen)
  - Método mestre

Eq. recorrência - Método de substituição

- Passos a serem seguidos:
  1. Supor um limite hipotético (um chute)
  2. Usar **indução matemática** para provar que a suposição está correta

- Exemplo:

$$T(n) = \begin{cases} O(1), \; se \; n = 1 \\ 2T(n/2) + O(1), \; caso \; contrário \end{cases}$$

  - Passo 1: Supor que $T(n) \in O(n \log n)$ (parece a eq do MergeSort – sl 4)
  - O passo 2 consiste em provar que $T(n) \leq c(n \log n)$ para algum $c > 0$ e $n >= n_0$.

Eq. recorrência - Método de substituição

- Exemplo:

$$T(n)=\begin{cases} O(1), se\ n=1 \\ 2T(n/2)+O(1), caso\ contrário \end{cases}$$

  - Supor que $T(n) \in O(n \log n)$
  - O passo 2 consiste em provar que $T(n) \leq c(n \log n)$ para algum $c > 0$ e $n >= n_0$.

Eq. recorrência - Método de substituição

- Exemplo:

$$T(n)=\begin{cases} O(1), se\ n=1 \\ 2T(n/2)+O(1), caso\ contrário \end{cases}$$

  - Supor que $T(n) \in O(n \log n)$
  - O passo 2 consiste em provar que $T(n) \leq c(n \log n)$ para algum $c > 0$ e $n >= n_0$.
  - **Base da indução**: temos que provar que $T(1) \leq c\ (1 \log 1)$
  - Mas, conforme definido acima, $T(1) = O(1)$, digamos $T(1) = 1$
  - $O(1) \leq c\ (1 \log 1)$ é falso, pois $O(1)$ é pelo menos 1, e $\log 1 = 0$...

Eq. recorrência - Método de substituição

- Exemplo:

$$T(n) = \begin{cases} O(1), se\ n=1 \\ 2T(n/2) + O(1), caso\ contrário \end{cases}$$

  - Supor que $T(n) \in O(n \log n)$
  - O passo 2 consiste em provar que $T(n) \leq c(n \log n)$ para algum $c > 0$ e $n >= n_0$.
  - **Base da indução**: temos que provar que $T(1) \leq c\ (1 \log 1)$
  - Mas, conforme definido acima, T(1) = O(1), digamos T(1) = 1
  - $O(1) \leq c\ (1 \log 1)$ é falso, pois O(1) é pelo menos 1, e log 1 = 0…
  - MAS, a notação assintótica exige que provemos $T(n) \leq c(n \log n)$ para n maior ou igual que uma constante n_0 de **nossa** escolha.
  - Se escolhermos novo passo base (da indução), (ex: $n_0 = 2$), então para $n \geq 2$

$$T(2) \leq c(2 \log 2),\ isto\ é,\ T(2) \leq 2c$$

Pela equação de recorrência 1, T(1) = O(1). Logo, $T(2) = 2T(1) + O(1) = O(3) \leq 2c$

A constante ***c*** escolhida tem que valer para o passo base e para o passo indutivo…

Então vamos fazer o passo indutivo:

Eq. recorrência - Método de substituição

- Exemplo:

$$T(n)=\begin{cases} O(1),\ se\ n=1 \\ 2T(n/2)+O(1),\ caso\ contrário \end{cases}$$

  - Supor que T(n) ∈ O(n logn)
  - O passo 2 consiste em provar que T(n) ≤ c(n logn) para algum c > 0 e n >= n_0.
  - **Passo indutivo:** assumindo que T(n/2) ≤ c(n/2 log n/2) (**hipótese da indução**)

$$T(n)\leq 2(c(n/2\log n/2))+n$$
$$T(n)\leq cn(\log n/2)+n$$

  - Manipulando somente o lado direito da inequação:

$$=cn\log n-cn\log 2+n$$
$$=cn\log n-cn+n$$
$$=cn\log n-n(c-1)$$

  - Se escolhermos **c≥1 e n_0 = 1**, a parcela (n(c-1)) vai diminuir a parcela (cn log n), então podemos afirmar que: $T(n)\leq cn\log n$

E ficaria provado o passo indutivo (mas falta completar a base)

EACH USP

Voltando para nosso passo base...

- Exemplo:

$$T(n) = \begin{cases} O(1), \, se \; n = 1 \\ 2T(n/2) + O(1), \, caso \; contrário \end{cases}$$

  - Supor que T(n) ∈ O(n logn)
  - O passo 2 consiste em provar que T(n) ≤ c(n logn) para algum c > 0 e n >= n_0.
  - **Base da indução**: temos que provar que T(1) ≤ c (1 log 1)
  - Se escolhermos novo passo base (da indução), (ex: $n_0 = 2$), então para n ≥ 2

$$T(2) \leq c(2 \log 2), \; isto \; é, \; T(2) \leq 2c$$

T(2) = 2T(1) + O(1) = O(3) ≤ 2c

A constante ***c*** escolhida tem que valer para o passo base e para o passo indutivo

Se cada O(1) levasse por exemplo tempo = 1, T(2) = 3 ≤ 2c

Escolhendo c ≥ 2 temos que T(2) = 3 ≤ 2c, ou seja, o passo base foi provado

(note que c = 2 e $n_0 = 2$ valem para o passo base e o passo indutivo).

Portanto:

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

Exercícios - 1

- Use o Método de Substituição para provar que:

  1. $T(n) = T(n-1) + n = O(n^2)$
  2. $T(n) = T(\lceil n/2 \rceil) + 1 = O(\lg n).$

     $T(n) = 2T(\lfloor n/2 \rfloor) + n = \Theta(n \lg n)$

Obs: considerem que $T(n) = O(1)$ para $n = 1$

Eq. recorrência - Método Mestre

- Fornece um processo de “livro de receitas” para resolver recorrências da forma:

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

  em que :

  $a \geq 1$;

  $b > 1$;

  $f(n)$ é uma função assintoticamente positiva.

- Método mestre exige memorização de três casos, mas a partir deles permite descobrir facilmente soluções de muitas recorrências.

Eq. recorrência - Método Mestre

- **Teorema Mestre**

Sejam a ≥ 1 e b > 1 constantes, seja f(n) uma função e seja T(n) definida sobre os inteiros não negativos pela recorrência

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

onde interpretamos n/b com o significado de $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$

Então, T(n) pode ser limitado assintoticamente como a seguir:

1. Se $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ para alguma constante ∊ > 0, então $T(n) = \Theta\left(n^{\log_b a}\right)$

2. Se $f(n) = \Theta\left(n^{\log_b a}\right)$, então $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

3. Se $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ para alguma constante ***∊ > 0***,
   e se ***af(n/b) ≤ cf(n)*** para alguma constante ***c < 1*** e para todo ***n*** suficientemente grande, então $T(n) = \Theta\left(f(n)\right)$

Eq. recorrência - Método Mestre

O que significa o Teorema Mestre?

Então, T(n) pode ser limitado assintoticamente como a seguir:

1. Se $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta\left(n^{\log_b a}\right)$

2. Se $f(n) = \Theta\left(n^{\log_b a}\right)$, então $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

3. Se $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ para alguma constante $\epsilon > 0$, e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e para todo n suficientemente grande, então $T(n) = \Theta(f(n))$

- Estamos comparando a função ***f(n)*** com a função $n^{\log_b a}$
- Intuitivamente, a solução para a recorrência é determinada pela maior das duas funções.

Caso 1: se a função $n^{\log_b a}$ for maior, a solução será $T(n) = \Theta\left(n^{\log_b a}\right)$

Caso 3: se a função ***f(n)*** for maior, a solução será $T(n) = \Theta(f(n))$

Caso 2: se 2 funções tiverem o mesmo tamanho, faz-se a multiplicação por um fator logarítmico e a solução será $T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta(f(n) \lg n)$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

$T(n) = aT(n/b) + f(n)$
onde :
$a \geq 1$; $b > 1$;
$f(n)$ é uma função assintoticamente positiva.

Exemplo 1:

$T(n) = 9T(n/3) + n$

a= ?

b= ?

f(n) = ?

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

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

onde :

$a \geq 1$; $b > 1$;

$f(n)$ é uma função assintoticamente positiva.

Exemplo 1:

$T(n) = 9T(n/3) + n$

a= 9

b= 3

f(n) = n

$f(n) = O\left(n^{\log_b a - \epsilon}\right)$, **algum** $\epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$, **algum** $\epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 1:

T(n) = 9T(n/3) + n

a= 9, b= 3, f(n) = n

Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

Como decidir SE uma das igualdades é verdadeira?
(pois pode não ser…

por conta do $\in$)

$f(n) = O\left(n^{\log_b a - \in}\right), \textbf{algum} \in > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \in}\right), \textbf{algum} \in > 0 \Rightarrow T(n) = \Theta(f(n))$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

  1. Determinar qual caso (se houver algum) se aplica

  2. Anotar a resposta

$T(n) = aT(n/b) + f(n)$
onde :
$a \geq 1$; $b > 1$;
$f(n)$ é uma função assintoticamente positiva.

Exemplo 1:

T(n) = 9T(n/3) + n

a= 9, b= 3, f(n) = n

Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

Como decidir SE uma das igualdades é verdadeira?
(pois pode não ser…

por conta do $\in$)

Testar cada caso!

$f(n) = O\left(n^{\log_b a - \in}\right)$, **algum** $\in > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \in}\right)$, **algum** $\in > 0 \Rightarrow T(n) = \Theta(f(n))$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

Exemplo 1:

T(n) = 9T(n/3) + n

a= 9, b= 3, f(n) = n

Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

- Caso 1, considerando ε =1

$n \in O\left(n^{\log_3 9-\varepsilon} = n^{2-1} = n\right)?$

$f(n) = O\left(n^{\log_b a-\epsilon}\right), \textit{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a+\epsilon}\right), \textit{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 1:

T(n) = 9T(n/3) + n

a= 9, b= 3, f(n) = n

Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

- Caso 1, considerando ε =1

$n \in O\left(n^{\log_3 9-\varepsilon} = n^{2-1} = n\right)?$

SIM!!!

Então: $T(n) = \Theta\left(n^2\right)$

$f(n) = O\left(n^{\log_b a-\epsilon}\right), \textit{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a+\epsilon}\right), \textit{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

Já encontramos! Não precisamos testar os outros casos!

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a = ?, b = ?, f(n) = ?

$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{\textit{algum}}\ \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{\textit{algum}}\ \epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a = 1, b = 3/2, f(n) = 1 $\quad n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Caso 1 , considerando ε =1: $\quad 1 \in O\left(n^{0-1} = n^{-1} = \frac{1}{n}\right)?$

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

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), algum\ \epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a = 1, b = 3/2, f(n) = 1 $\quad n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Caso 1 , considerando ε =1: $\quad 1 \in O\left(n^{0-1} = n^{-1} = \frac{1}{n}\right)?$

NÃO!!!

Caso 2: $\quad 1 \in \Theta(1)?$

$f(n) = O\left(n^{\log_b a - \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇨ $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta(f(n))$

EACH USP

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a = 1, b = 3/2, f(n) = 1 $\quad n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Caso 1 , considerando ε =1: $\quad 1 \in O\left(n^{0-1} = n^{-1} = \frac{1}{n}\right)?$

NÃO!!!

Caso 2: $\quad 1 \in \Theta(1)?$

SIM!!!

Então: $\quad T(n) = \Theta(n^0 \lg n) = \Theta(\lg n)$

$f(n) = O(n^{\log_b a - \epsilon})$, *algum* $\epsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$

$f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$

$f(n) = \Omega(n^{\log_b a + \epsilon})$, *algum* $\epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$

Poderia ter vindo direto para o caso 2

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3

b= 4

f(n) = n lg n

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

$f(n) = O\left(n^{\log_b a - \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇨ $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta(f(n))$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{log_4 3} = O\left(n^{0,793}\right)$

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$?

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

onde :

a ≥ 1; b > 1;

f(n) é uma função assintoticamente positiva.

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

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), algum\ \epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$? NÃO!!! (porque sempre leva a nlgn=O(n^k), k<1)

Caso 2: $n \lg n \in \Theta\left(n^{0,793}\right)$?

$f(n) = O\left(n^{\log_b a - \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇨ $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta\left(f(n)\right)$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$? NÃO!!! (porque sempre leva a nlgn=O(n^k), k<1)

Caso 2: $n \lg n \in \Theta\left(n^{0,793}\right)$? NÃO!!! (porque leva a nlgn=O(n^k), k<1)

Caso 3, considerando ε ≈0,2:

$n \lg n \in \Omega\left(n^{0,793+0,2}\right)$? ou seja: $n \lg n \in \Omega(n)$?

$f(n) = O\left(n^{\log_b a - \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇨ $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$, *algum* $\epsilon > 0$ ⇨ $T(n) = \Theta(f(n))$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

$$n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$$

Portanto, temos

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$? NÃO!!! (porque sempre leva a nlgn=O(n^k), k<1)

Caso 2: $n \lg n \in \Theta\left(n^{0,793}\right)$? NÃO!!! (porque leva a nlgn=O(n^k), k<1)

Caso 3, , considerando ε ≈0,2:

$n \lg n \in \Omega\left(n^{0,793+0,2}\right)$? ou seja: $n \lg n \in \Omega(n)$? SIM, mas ainda temos que provar a expressão de regularidade.

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Se $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ para alguma constante $\epsilon > 0$,
e se ***af(n/b) ≤ cf(n)*** para alguma constante ***c < 1*** e para todo ***n*** suficientemente grande, então $T(n) = \Theta(f(n))$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:

1. Determinar qual caso (se houver algum) se aplica
2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

$$n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$$

Portanto, temos

Caso 3, considerando ε ≈0,2:

$n \lg n \in \Omega\left(n^{0,793+0,2}\right)$? ou seja: $n \lg n \in \Omega(n)$? SIM, mas ainda temos que provar a expressão de regularidade af(n/b) ≤ cf(n), para alguma constante c < 1.

$$af(n/b) = 3(n/4)\lg(n/4) \le (3/4)n \lg n = cf(n) \quad para \ \ c = 3/4$$

Então: $T(n) = \Theta(n \lg n)$

Se $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ para alguma constante $\epsilon > 0$,
e se ***af(n/b) ≤ cf(n)*** para alguma constante ***c < 1*** e para todo ***n*** suficientemente grande, então $T(n) = \Theta(f(n))$

Eq. recorrência - Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

Exemplo 4:

T(n) = 2T(n/2) + n lg n

a= 2, b= 2, f(n) = n lg n

$$n^{\log_b a} = n^{\log_2 2} = O(n^1)$$

Portanto, temos

Caso 1, considerando ε = 0,1: $n \lg n \in O(n^{1-0,1})$? NÃO!!!

Caso 2: $n \lg n \in \Theta(n)$? NÃO!!!

Caso 3, considerando ε = 0,1: $n \lg n \in \Omega(n^{1+0,1})$? NÃO!!!

Na verdade, (n lgn n)/n é menor que n^{ε} para qualquer ε > 0 !!!

Logo, o método mestre não pode ser usado para esta recorrência...

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

$f(n) = O(n^{\log_b a - \epsilon})$, ***algum*** $\epsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$

$f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$

$f(n) = \Omega(n^{\log_b a + \epsilon})$, ***algum*** $\epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$

Teorema Mestre

- Exemplos de aplicação:
  - (Caso 1) $T(n) = 4T(n/2) + n \log(n)$, $T(1) = 1$.
  - (Caso 2) $T(n) = 2T(n/2) + n$, $T(1) = 1$.
  - (Caso 3) $T(n) = T(n/2) + n \log(n)$, $T(1) = 1$.

- Exemplos nos quais não se aplica:
  - $T(n) = T(n - 1) + n \log(n)$, $T(1) = 1$.
  - $T(n) = T(n - a) + T(a) + n$, $T(b) = 1$.
    (para inteiros $a \geq 1$, $b \leq a$, a e b inteiros)
  - $T(n) = T(\alpha n) + T((1 - \alpha)n) + n$, $T(1) = 1$.
    (para $0 < \alpha < 1$)
  - $T(n) = T(n - 1) + \log(n)$, $T(1) = 1$.
  - $T(n) = 2\,T(n/2) + n \log(n)$, $T(1) = 1$.

Exercícios - 1

- Use o Método de Substituição para provar que:

1. $T(n) = T(n-1) + n = O(n^2)$
2. $T(n) = T(\lceil n/2 \rceil) + 1 = O(\lg n)$.

   $T(n) = 2T(\lfloor n/2 \rfloor) + n = \Theta(n \lg n)$

Obs: considerem que $T(n) = O(1)$ para $n = 1$

Exercícios - 2

- Use o Teorema Mestre para encontrar solução para as seguintes recorrências:

1. $T(n) = 4T(n/2) + n \log(n)$
2. $T(n) = 2T(n/2) + n$
3. $T(n) = T(n/2) + n \log(n)$
4. $T(n) = T(n/2) + \Theta(1)$
5. $T(n) = 4T(n/2) + n$
6. $T(n) = 4T(n/2) + n^2$
7. $T(n) = 4T(n/2) + n^3$

EACH USP

Referências

- **Mergesort iterativo**: Paulo Feofiloff. Algoritmos em linguagem C. Cap 9.
- **Recorrências**: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002 (Seções 4.3 a 4.5)

Novo calendário

- P1: 27/11 (cai matéria dada até 16/11)
- P2: 18/12 (cai matéria dada a partir de 23/11, mas tem que saber fazer análise de complexidade - algoritmos de ordenação)
- PSub: 21/12
- PRec: fevereiro (aguardando calendário de 2024)
- EP 2: 20/12 (disparado em 13/11 (parte 1) e 30/11 (parte 2))

EACH USP

“Reposição” de aulas

- 8 aulas foram perdidas durante a paralisação
- Tivemos 5 plantões gravados:
  - O resumo de cada um vale 1 presença
- As 3 demais serão compensadas com atividades solicitadas nas próximas aulas