Trabalho 01 - CAL

Aluno: Bruno Rafael dos Santos

Calculando a complexidade do algoritmo

Merge Sort:

- note que a função de custo para este algoritmo com uma entrada de tamanho n é definida como

$$T(n) = \begin{cases} T(n) = 2T(n/2) + n + 1, & n > 1 \\ T(n) = 1, & n \leq 1 \end{cases}$$

- portanto note que realizando i iterações da função T(n) tem-se

1. $T(n) = 2T(n/2) + n + 1$

2. $T(n) = 2\left(2T(n/4) + \frac{n}{2} + 1\right) + n + 1 = 4T(n/4) + 2n + 3$

3. $T(n) = 4\left(2T(n/8) + \frac{n}{4} + 1\right) + 2n + 3 = 8T(n/8) + 3n + 7$

$\vdots$

i. $T(n) = 2^i T(n/2^i) + i.n + (2^i - 1)$

- provando pelo primeiro princípio de indução que tal fórmula vale para $\forall n \in \mathbb{N}$ na i-ésima iteração, sendo $i \geq 2$:

1. caso base ($i = 2$):

$$T(n) = 2^2 T(n/2^2) + 2.n + (2^2 - 1)$$

$$2T(n/2) + n + 1 = 4T(n/4) + 2n + 3 \quad \text{(pela definição de T(n))}$$

$$2\left[2T(n/4) + \frac{n}{2} + 1\right] + n + 1 = 4T(n/4) + 2n + 3 \quad \text{(pela definição de T(n))}$$

$$4T(n/4) + n + 2 + n + 1 = 4T(n/4) + 2n + 3$$

$$4T(n/4) + 2n + 3 = 4T(n/4) + 2n + 3 \quad \square$$

Trabalho 01 - CAL

Aluno: Bruno Rafael dos Santos

(continuação)

hipótese:

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn + (2^k - 1)$$

② passo (K-1):

$$T(n) = 2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1).n + (2^{k+1} - 1)$$

$$2^k T\left(\frac{n}{2^k}\right) + kn + (2^k - 1) = 2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1)n + (2^{k+1} - 1) \quad \text{(pela hipótese)}$$

$$2^k \left[2T\left(\frac{n}{2^k \cdot 2}\right) + \frac{n}{2^k} + 1\right] + kn + (2^k - 1) = 2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1)n + (2^{k+1} - 1) \quad \text{(pela def. de } T(n))$$

$$2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + n + 2^k + kn + (2^k - 1) = 2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1)n + (2^{k+1} - 1)$$

$$2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1).n + 2\,2^k - 1 = 2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1)n + (2^{k+1} - 1)$$

$$2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1).n + 2^{k+1} - 1 = 2^{k+1} T\left(\frac{n}{2^{k+1}}\right) + (k+1)n + (2^{k+1} - 1) \quad \square$$

- agora que sabemos que à fórmula é válida para qualquer iteração i desde que $i \geq 2$, note que a recursão alcança o caso base quando o argumento é 1, ou seja, quando $\frac{n}{2^i} = 1$, donde vem:

$$\frac{n}{2^i} = 1 \rightarrow n = 2^i \rightarrow \log_2 n = \log_2 2^i \rightarrow i = \log_2 n$$

- portanto a recursão alcança o caso base quando $i = \log_2 n$, assim, note que, substituinob $i = \log_2 n$ na fórmula:

$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n.\log_2 n + (2^{\log_2 n} - 1)$$

$$T(n) = n.T(n/n) + n.\log_2 n + n - 1$$

$$T(n) = nT(1) + n\log_2 n + n - 1$$

$$T(n) = n + n.\log_2 n + n - 1$$

$$T(n) = n.\log_2 n + 2n - 1$$

Trabalho 01 - CAL

Aluno: Bruno Rafael dos Santos

(continuação)

- agora para determinar a classe de complexidade do algoritmo vamos mostrar que $T(n) \in O(n \log n)$, assim temos que mostrar que $\exists c \in \mathbb{R}^+$ e $\exists m \in \mathbb{N}_0$ tal que

$$T(n) \leq c . n . \log_2 n \ , \ \forall n \geq m$$

$$n . \log_2 n + 2n - 1 \leq c \, n . \log_2 n \ , \ \forall n \geq m$$

- escolhendo $c = 12$ e $m = 4$ temos que provar

$$n . \log_2 n + 2n - 1 \leq 12 . n . \log n \ , \ \forall n \geq 4$$

- temos então

caso base $(n=4)$:

$$4 \log_2 4 + 2 . 4 - 1 \leq 12 . 4 . \log_2 4$$

$$8 + 7 \leq 24 . 2$$

$$15 \leq 48 \quad \square$$

hipótese:

$$k . \log_2 k + 2k - 1 \leq 12 k . \log_2 k \Rightarrow 11 k \log_2 k \geq 2k - 1 \Rightarrow 11 k \log_2 k + 2 \geq 2k + 1$$

passo $(k+1)$:

$$(k+1) . \log_2 (k+1) + 2(k+1) - 1 \leq 12 (k+1) . \log_2 (k+1)$$

$$k \log_2 (k+1) + \log (k+1) + 2k + 1 \leq 12 k \log_2 (k+1) + 12 \log_2 (k+1)$$

$$2k + 1 \leq 11 k \log_2 (k+1) + 11 \log_2 (k+1)$$

de acordo com o caso base, $k \geq 4$, logo

$$11 k \log_2 (k+1) + 11 \log_2 (k+1) \geq 11 k \log_2 (k+1) + 11 \log_2 (4+1)$$

$$\geq 11 k \log_2 (k+1) + 11 \log_2 (5)$$

Trabalho 01 - CAL

Aluno: Bruno Rafael dos Santos

(continuação)

- e como $\log_2 5 = 2{,}32192 \geq 2$, então

$$11k \log_2(k+1) + 11 \log_2 5 \geq 11k \log_2(k+1) + 2$$

- e note também que $\log_2(k+1) \geq \log_2 k$, portanto:

$$11k \log_2(k+1) + 2 \geq 11k \log_2 k + 2$$

- e pela hipótese

$$11k \log_2(k+1) + 2 \geq 11 \log_2 k + 2 \geq 2k+1$$

- logo, pela transitividade da relação "$\geq$" temos que

$$11k \log_2(k+1) + 11 \log_2(k+1) \geq 2k+1$$

- ou seja vale a desigualdade para $k+1$, portanto está provado que

$$n \cdot \log_2 n + 2n - 1 \leq 12 \cdot n \log_2 n, \quad \forall n \geq 4$$

- e isto significa que

$$n \cdot \log_2 n + 2n - 1 \in O(n \cdot \log n)$$

- ou

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

<u>Obs.</u>: a definição de $T(n)$ no início pode ser explicada da seguinte forma:

$$T(n) = \underbrace{2T(n/2)}_{①} + \underbrace{n}_{②} + \underbrace{1}_{③}$$

$$T(1) = \underbrace{1}_{\text{uma entrada de tamanho 1 já está ordenada}}$$

① o merge sort é aplicado 1 vez em cada metade, ou seja, 2 vezes em algum vetor com metade do tamanho

② há o custo do "merge" das duas metades que foram separadas

③ custo para encontrar o meio do vetor