PAA - Aula 03
Divisão e Conquista, MergeSort, Árvore de Recorrência

Etapas do projeto de algoritmos por divisão e conquista

- Dividir: o problema original é dividido em subproblemas menores do mesmo tipo.
- Conquistar: os subproblemas são resolvidos recursivamente, sendo que os subproblemas pequenos são caso base.
- Combinar: as soluções dos subproblemas são combinadas numa solução do problema original.

importante p/ a eficiência

Exemplo/ideia do mergeSort:

v 8 5 4 1 | 3 6 2 7

- vetor fora de ordem
- dividir o vetor a ser ordenado em dois subvetores, cada um com metade do tamanho original

vl 8 5 4 1 vr 3 6 2 7

- ordenar recursivamente os dois subvetores, sendo que subvetores com 0 ou 1 elementos já estão ordenados

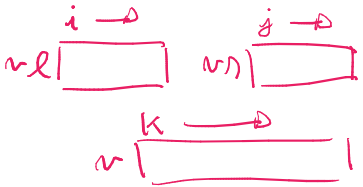
vl 1 4 5 8 vr 2 3 6 7

- intercalar (merge) os subvetores ordenados resultantes

v 1 2 3 4 5 6 7 8

Pseudocódigo do algoritmo mergeSort recursivo:

```
mergeSort (vetor v):
    se tam(v) ≤ 1: retorna
    (vl, vr) = 1ª e 2ª metade de v
    mergeSort(vl)
    mergeSort(vr)
    v = intercala(vl, vr)
```



Pseudocódigo do algoritmo intercala:

```
intercala (vetores ordenados vl e vr):
    i = j = K = 1   → convenciono vetores começando em 1
    enquanto i ≤ tam(vl) E j ≤ tam(vr):
        se vl[i] ≤ vr[j]: v[K++] = vl[i++]
        senão v[K++] = vr[j++]
    laços p/ copiar o restante de vl ou vr
    devolva v
```

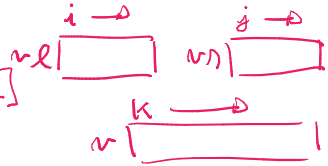
$O(tam(vl) + tam(vr))$

a++ == a = a+1

Eficiência do intercala: o número de operações é linear no tamanho dos subvetores,

- i.e., da ordem de $tam(vl) + tam(vr) = tam(v)$
- Para verificar, note que em cada iteração um elemento é colocado em v.

Corretude do intercala com invariantes de laço: no início de cada iteração



1 • $v[1..K-1]$ contém os elementos de $vl[1..i-1]$ e $vr[1..j-1]$

2 • os elementos de $v[1..K-1]$, vl, vr estão ordenados,

3 • $vl[1..i-1] \leq vr[j..n_r]$ e $vr[1..j-1] \leq vl[i..n_e]$

Prova de corretude por indução no # da iteração (ou no K):

Base: no início da iter. 1 temos $i = j = K = 1$. Portanto $v[1..K-1] = vl[1..i-1] = vr[1..j-1] = \phi$. Assim, as prop. inv. valem trivialmente

H.I.: as prop. inv. valem no início da K-ésima iteração.

Passo: na K-ésima iteração o menor elemento (x) entre $vl[i]$ e $vr[j]$ é colocado em $v[K]$

Como vl e vr estão ordenados, temos $vl[i] \leq vl[i..n_e]$ e $vr[j] \leq vr[j..n_r]$.

Já que x é o menor entre $vl[i]$ e $vr[j]$, ele é o menor dos não copiados.

Pela H.I. (1,2,3) sabemos que todos os copiados ($v[1..K-1]$) são menores que os não copiados. Assim, x é maior ou igual aos copiados. Portanto, x está sendo copiado p/a posição correta e, ao final da iter., quando K (e 'i' ou 'j') são incrementados, as prop. invariantes são restabelecidas.

Note que, após a última iter. os inv. garantem que v tem todos os elementos ordenados

Pseudocódigo do algoritmo mergeSort recursivo:

mergeSort(vetor v):

- se tam(v) <= 1: devolva
- (vl, vr) = 1ª e 2ª metade de v, respectivamente
- 2 ⇒ { mergeSort(vl)
  mergeSort(vr)
- v = intercala(vl, vr)

O(n)

Corretude do mergeSort por indução:

- Pelo caso base, sabemos que nosso algoritmo devolve subvetores ordenados quando estes têm tamanho menor ou igual a 1.
- Supondo, como H.I., que nosso algoritmo ordena corretamente
  - um subvetor de tamanho n/2,
- verificamos que ele ordena um vetor de tamanho n,
  - uma vez que a função intercala funciona corretamente.

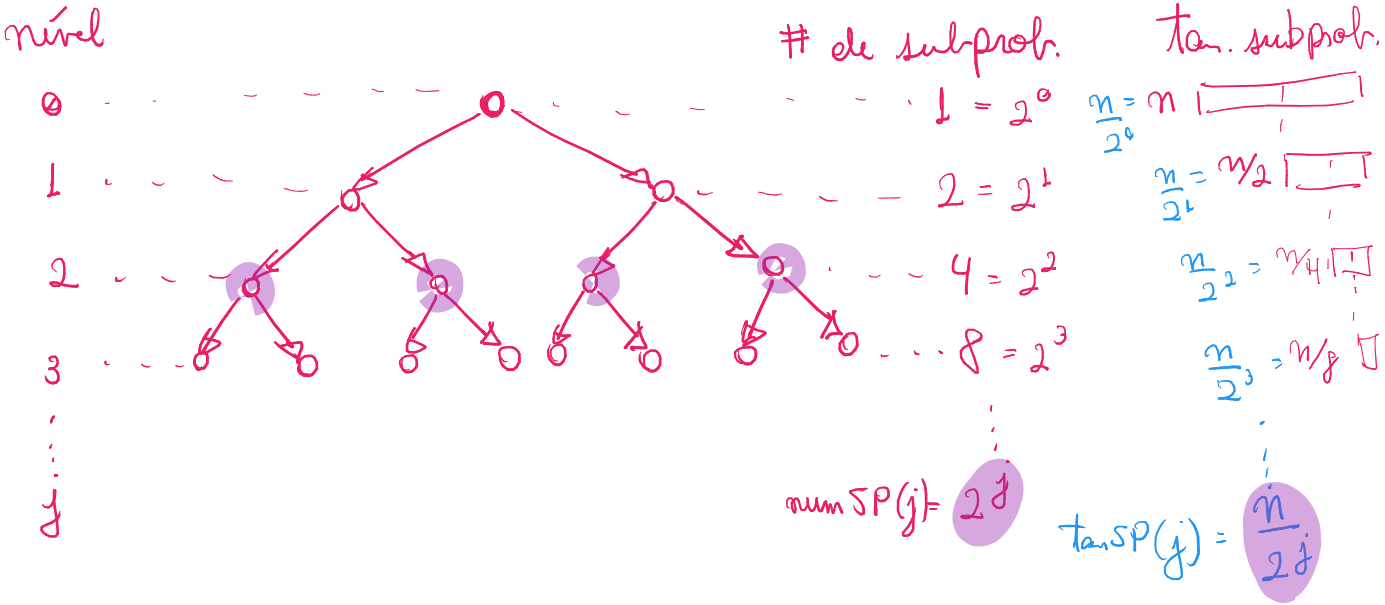
Análise de eficiência:

- Quantos são os subproblemas/chamadas recursivas? 2
- Qual o tamanho dos ~~inteiros~~ vetores nos subproblemas? $\boxed{n/2}$
- Qual o trabalho local (não recursivo)? $c.n$, sendo c uma constante e n o tamanho de v
- Assim, o tempo gasto é dado pela recorrência:
- Mas, qual é a função de tempo deste algoritmo?

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

Resolvendo a recorrência $T(n) = 2T(n/2) + cn$

- usando uma árvore (binária) de recursão.



Qual o último nível h da árvore?

- Note que, as chamadas recursivas param quando o tamanho do subproblema é 1. Portanto $\text{tam SP}(h) = \frac{n}{2^h} = 1 \Rightarrow 2^h = n \Rightarrow h = \lg n$

Como começamos a contar os níveis no nível 0, o número de níveis é $1 + \lg n$

O número de níveis da árvore é $1 + \lg n$ já que:

- no nível 0 temos n elementos no vetor,
- lg n = log_2 n é o número de vezes que podemos dividir n por 2 antes dele se tornar menor ou igual a 1 (caso base).

No nível j o número de subproblemas é 2^j

- e o tamanho do vetor dos subproblemas é $n/2^j$

Qual o trabalho local (não recursivo) do mergeSort?

- Uma chamada do mergeSort realiza basicamente um teste,
  - seguido de duas chamadas recursivas e uma chamada de intercala.
- Como intercala tem eficiência de tempo linear no tamanho dos vetores,
  - o trabalho não recursivo num vetor de tamanho m é $c.m$

    ↳ é uma constante

Qual o trabalho realizado no nível j?

$$\boxed{trab(j)} = numSP(j).c.tamSP(j)$$

$$= \cancel{2^j}.c.n/\cancel{2^j} = \boxed{c.n}$$

trab. é uniforme ao longo dos níveis

Por fim, o trabalho total é dado pela soma do trabalho por nível da árvore

- ao longo do número de níveis da árvore, i.e.,

$$\boxed{T(n) = \sum_{j=0}^{\lg n} trab(j) = \sum_{j=0}^{\lg n} c.n = c.n\,(1+\lg n) = cn\lg n + cn = O(n \lg n)}$$

Numa comparação rápida, para n = 10^6 e 10^9 temos:

- lg n = log_2 n
- n lg n = n log_2 n
- n^2

n	10^6	10^9
$\lg n$	20	30
$n \lg n$	2.10^7	3.10^{10}
n^2	10^{12}	10^{18}

Supondo que um computador realize 1 Giga (10^9) operações por segundo, temos:

- um algoritmo de ordenação O(n log n) leva, da ordem de, centésimos de segundo e 30 segundos, $\rightarrow 3.10^{10}/10^9 \cong 30$ seg
- um algoritmo de ordenação O(n^2) leva, da ordem de, 16 minutos e 32 anos.

10^9

10^2

↳ insertion Sort, selection Sort, bubble Sort

Animação: https://www.youtube.com/watch?v=ZZuD6iUe3Pc

(Bônus) Resolvendo a recorrência por substituição: T(n) = 2 * T(n/2) + cn

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

$$T(n/2) = 2T(n/4) + cn/2$$

$$T(n/4) = 2T(n/8) + cn/4$$

$$T(n/8) = 2T(n/16) + cn/8$$

$$T(n) = 2T(n/2) + cn = 2^1 T(n/2^1) + 1cn$$

$$= 2(2T(n/4) + cn/2) + cn$$

$$= 4T(n/4) + 2cn = 2^2 T(n/2^2) + 2cn$$

$$= 4(2T(n/8) + cn/4) + 2cn$$

$$= 8T(n/8) + 3cn = 2^3 T(n/2^3) + 3cn$$

$$= 8(2T(n/16) + cn/8) + 3cn$$

$$= 16T(n/16) + 4cn = 2^4 T(n/2^4) + 4cn$$

$$T(n) = 2^j T(n/2^j) + jcn$$

no maior j temos $\frac{n}{2^j} = 1 \Rightarrow \lg(2^j) = \lg n \Rightarrow j = \lg n$

$$\boxed{T(n) = 2^{\lg n} \cdot T(1) + \lg n \cdot c \cdot n = n \cdot c + cn \lg n}$$

$$\boxed{T(n) = 2^{\lg n} \cdot T(1) + \lg n \cdot c \cdot n = n \cdot c + c\, n \lg n}$$

Vamos verificar que a fórmula que deduzimos está correta usando indução matemática

Caso base: $T(1) = 1 \cdot c + c \cdot 1 \cdot \lg 1 = c$ ✓

H.I.: $T(n') = c n' \lg n' + c n'$ p/ $n' < n$

Passo: $\boxed{T(n)} = 2T(n/2) + cn$

pela H.I. $= 2\left(c(n/2)\lg(n/2) + c.(n/2)\right) + cn$

$= cn \lg n - cn \lg 2 + cn + cn$

$= \boxed{cn \lg n + cn}$ ✓

Portanto $T(n) = cn \lg n + cn = \Theta(n \lg n)$