FCT/Unesp – Presidente Prudente
Projeto e Análise de Algoritmos
Prof. Danilo Medeiros Eler

Exercícios Aula 04
https://daniloeler.github.io/teaching/PAA2020/index.html

Resolução

1)
a) $T(n) = 4T(n/2) + n$
$a = 4; b = 2; f(n) = n;$

CASO 1: $f(n) = O(n^{\log_b a - E})$
$n = O(n^{\log_2 4 - E})$
$n = O(n^{2 - E})$
$n <= c\ n^{2 - E}$; $E = 1$;
$n <= c\ n$ VERDADEIRO

$T(N) = \Theta(n^{\log_b a})$
$T(N) = \Theta(n^{\log_2 4})$
$T(N) = \Theta(n^2)$

b) $T(n) = 4T(n/2) + n^2$
$a = 4; b = 2; f(n) = n^2;$

CASO 1: $f(n) = O(n^{\log_b a - E})$
$n^2 = O(n^{\log_2 4 - E})$
$n^2 = O(n^{2 - E})$
$n^2 <= c\ n^{2 - E}$; Não existe um E maior do que zero que torne verdadeiro
FALSO

CASO 2: $f(n) = \Theta\ (n^{\log_b a})$
$n^2 = \Theta\ (n^{\log_2 4})$
$n^2 = \Theta\ (n^2)$
$c1\ n^2 <= n^2 <= c2\ n^2$
VERDADEIRO

$T(N) = \Theta(n^{\log_b a} \log n)$
$T(N) = \Theta(n^{\log_2 4} \log n)$
$T(N) = \Theta(n^2 \log n)$

c) $T(n) = 4T(n/2) + n^3$
$a = 4; b = 2; f(n) = n^3;$

CASO 1: $f(n) = O(n^{\log_b a - E})$
$n^3 = O(n^{\log_2 4 - E})$
$n^3 = O(n^{2 - E})$
$n^3 <= c\ n^{2 - E}$; Não existe um E maior do que zero que torne verdadeiro
FALSO

CASO 2: $f(n) = \Theta\ (n^{\log_b a})$
$n^3 = \Theta\ (n^{\log_2 4})$
$n^3 = \Theta\ (n^2)$
$c1\ n^2 <= n^3 <= c2\ n^2$

O primeiro lado $c1\ n^2 <= n^3$ é verdadeiro, mas o segundo lado $n^3 <= c2\ n^2$ é falso

FALSO

CASO 3: $f(n) = \Omega(n^{\log_b a + E})$
$n^3 = \Omega\ (n^{\log_2 4 + E})$
$n^3 = \Omega\ (n^{2 + E})$
$n^3 >= c\ n^{2 + E}$; $E = 1$;
$n^3 >= c\ n^3$

VERDADEIRO

$af(n/b) <= cf(n)$
$4f(n/2) <= cf(n)$
$4(n/2)^3 <= cn^3$
$4(n^3/8) <= cn^3$; $c = 4/8$
$4n^3/8 <= 4n^3/8$
VERDADEIRO

$T(N) = \Theta(f(n))$
$T(N) = \Theta(n^3)$

d) $T(n) = 7T(n/2) + n^2$
$a = 7; b = 2; f(n) = n^2;$

CASO 1: $f(n) = O(n^{\log_b a - E})$
$n^2 = O(n^{\log_2 7 - E})$
$n^2 = O(n^{2.8 - E})$
$n^2 <= c\ n^{2.8 - E}$; $E = 0.8$
$n^2 <= c\ n^2$
VERDADEIRO

$T(N) = \Theta(n^{\log_b a})$
$T(N) = \Theta(n^{\log_2 7})$
$T(N) = \Theta(n^{2.8})$

e) $T(n) = T(n/2) + 1$
$a = 1$; $b = 2$; $f(n) = 1$;

CASO 1: $f(n) = O(n^{\log_b a - E})$
$1 = O(n^{\log_2 1 - E})$
$1 = O(n^{0 - E})$
$1 <= c\ n^{0 - E}$;
$1 <= c\ n^{-E}$;
$1 <= c/n^E$; Não existe um E maior do que zero que torne verdadeiro, pois um lado se manterá constante enquanto o outro tenderá a zero.
FALSO

CASO 2: $f(n) = \Theta\ (n^{\log_b a})$
$1 = \Theta\ (n^{\log_2 1})$
$1 = \Theta\ (n^0)$
$c1\ n^0 <= 1 <= c2\ n^0$
$c1\ 1 <= 1 <= c2\ 1$; $c1 = c2 = 1$

VERDADEIRO

$T(N) = \Theta(n^{\log_b a} \log n)$
$T(N) = \Theta(n^{\log_2 1} \log n)$
$T(N) = \Theta(n^0 \log n)$
$T(N) = \Theta(1 \log n)$
$T(N) = \Theta(\log n)$

f) $T(n) = 3T(n/5) + n\log n$
$a = 3$; $b = 5$; $f(n) = n\log n$;

CASO 1: $f(n) = O(n^{\log_b a - E})$
$n\log n = O(n^{\log_5 3 - E})$
$n\log n = O(n^{0.68 - E})$
$n\log n <= c\ n^{0.68 - E}$; Não existe um E maior do que zero que torne verdadeiro
FALSO

CASO 2: $f(n) = \Theta\ (n^{\log_b a})$
$n\log n = \Theta\ (n^{\log_5 3})$
$n\log n = \Theta\ (n^{0.68})$
$c1\ n^{0.68} <= n\log n <= c2\ n^{0.68}$

O primeiro lado $c1\ n^{0.68} <= n\log n$ é verdadeiro, mas o segundo lado $n\log n <= c2\ n^{0.68}$ é falso

FALSO

CASO 3: $f(n) = \Omega(n^{\log_b a + E})$
$n\log n = \Omega(n^{\log_5 3 + E})$
$n\log n = \Omega(n^{0.68 + E})$
$n\log n >= c\, n^{0.68 + E}$; E = 0.32
$n\log n >= c\, n$

VERDADEIRO

af(n/b) <= cf(n)
3f(n/5) <= cf(n)
3(n/5 logn/5) <= cnlogn
3n/5(logn – log5) <= cnlogn ; c = 3/5
3n/5(logn – log5) <= **3n/5**logn
logn – log5 <= logn ; o lado esquerdo sempre será menor pelo fator log5
VERDADEIRO

$T(N) = \Theta(f(n))$
$T(N) = \Theta(n\log n)$

g) T(n) = 3T(n/5) + nlogn
a = 3; b = 4; f(n) = nlogn;

CASO 1: $f(n) = O(n^{\log_b a - E})$
$n\log n = O(n^{\log_4 3 - E})$
$n\log n = O(n^{0.79 - E})$
$n\log n <= c\, n^{0.79 - E}$; Não existe um E maior do que zero que torne verdadeiro
FALSO

CASO 2: $f(n) = \Theta(n^{\log_b a})$
$n\log n = \Theta(n^{\log_4 3})$
$n\log n = \Theta(n^{0.79})$
$c1\, n^{0.79} <= n\log n <= c2\, n^{0.79}$

O primeiro lado $c1\, n^{0.79} <= n\log n$ é verdadeiro, mas o segundo lado $n\log n <= c2\, n^{0.79}$ é falso

FALSO

CASO 3: $f(n) = \Omega(n^{\log_b a + E})$
$n\log n = \Omega(n^{\log_4 3 + E})$
$n\log n = \Omega(n^{0.79 + E})$
$n\log n >= c\, n^{0.79+ E}$; E = 0.21
$n\log n >= c\, n$

VERDADEIRO

$af(n/b) <= cf(n)$
$3f(n/4) <= cf(n)$
$3(n/4 \log n/4) <= cn\log n$
$3n/4(\log n – \log 4) <= cn\log n$; $c = 3/4$
$\underline{\mathbf{3n/4}}(\log n – \log 4) <= \underline{\mathbf{3n/4}}\log n$
$\log n – \log 4 <= \log n$; o lado esquerdo sempre será menor pelo fator log4
$\log n – 2 <= \log n$; o lado esquerdo sempre será menor pelo fator 2
VERDADEIRO

$T(N) = \Theta(f(n))$
$T(N) = \Theta(n\log n)$

2) O algoritmo faz quatro chamadas recursivas, quebrando o problema (tamanho do vetor) ao meio. Cada chamada recursiva tem um custo de n^2 instruções que precisam ser executadas. Assim, a recorrência que descreve esse algoritmo é: $T(n) = 4T(n/2) + n^2$

$T(n) = 4T(n/2) + n^2$
$a = 4$; $b = 2$; $f(n) = n^2$;

CASO 1: $f(n) = O(n^{\log_b a – E})$
$n^2 = O(n^{\log_2 4 – E})$
$n^2 = O(n^{2 – E})$
$n^2 <= c\, n^{2 – E}$; Não existe um E maior do que zero que torne verdadeiro
FALSO

CASO 2: $f(n) = \Theta (n^{\log_b a})$
$n^2 = \Theta (n^{\log_2 4})$
$n^2 = \Theta (n^2)$
$c1\, n^2 <= n^2 <= c2\, n^2$
VERDADEIRO

$T(N) = \Theta(n^{\log_b a} \log n)$
$T(N) = \Theta(n^{\log_2 4} \log n)$
$T(N) = \Theta(n^2 \log n)$

3) O algoritmo faz uma chamada recursiva, quebrando o problema (tamanho do vetor) em três – passa somente um terço do vetor a cada chamada. Cada chamada recursiva necessita executar n instruções. Assim, a recorrência que descreve esse algoritmo recursivo pode ser expressa como: $T(n) = T(n/3) + n$

$T(n) = T(n/3) + n$
$a = 1$; $b = 3$; $f(n) = n$;

CASO 1: $f(n) = O(n^{\log_b a – E})$
$n = O(n^{\log_3 1 – E})$
$n = O(n^{0 – E})$
$n <= c\, n^{0 – E}$

$n <= c\, n^{-E}$;
$n <= c/n^{E}$; Não existe um E maior do que zero que torne verdadeiro, pois um lado tenderá para o infinito enquanto o outro tenderá a zero.
FALSO

CASO 2: $f(n) = \Theta\, (n^{\log_b a})$
$n = \Theta\, (n^{\log_3 1})$
$n = \Theta\, (n^{0})$
$c1\, n^{0} <= n <= c2\, n^{0}$
$c1\ 1 <= n <= c2\ 1$
$c1 <= n <= c2$

O primeiro lado $c1 <= n$ é verdadeiro, mas o segundo lado $n <= c2$ é falso
FALSO

CASO 3: $f(n) = \Omega(n^{\log_b a + E})$
$n = \Omega\, (n^{\log_3 1 + E})$
$n = \Omega\, (n^{0 + E})$
$n >= c\, n^{0 + E}$; $E = 1$;
$n >= c\, n$

VERDADEIRO

$af(n/b) <= cf(n)$
$1f(n/3) <= cf(n)$
$1(n/3) <= cn$
$n/3 <= cn$; $c = 1/3$
$n/3 <= n/3$
VERDADEIRO

$T(N) = \Theta(f(n))$
$T(N) = \Theta(n)$

4)
O algoritmo principal é composto por um algoritmo iterativo (subPrograma01) e um recursivo (subPrograma02).

O algoritmo subPrograma01 tem dois casos de execução distintos para melhor e pior caso. No melhor caso o ‘for’ mais interno é interrompido e nunca itera. Isso ocorre quando o array já está ordenado. Assim, o melhor caso tem a complexidade $\Theta(n)$.
No pior caso o ‘for’ mais interno executará sempre até o início do vetor, ou seja, até a condição $i>=0$ falhar. Isso ocorre sempre que o vetor estiver em ordem decrescente. Assim, teremos o somatório clássico (1 a n) que resulta em $\Theta(n^2)$.

O algoritmo subPrograma02 é a busca binária. Como ela é um algoritmo recursivo, podemos encontrar a relação de recorrência que a descreve para calcular a sua complexidade. A relação de recorrência desse algoritmo pode ser escrita como $T(n) = T(n/2) + 4$; sendo 4 uma constante que poderia ser escrita por uma letra 'c', por exemplo.

Utilizando o teorema mestre, calculamos da seguinte maneira:

$T(n) = T(n/2) + 4$
$a = 1; b = 2; f(n) = 4;$

CASO 1: $f(n) = O(n^{\log_b a - E})$
$4 = O(n^{\log_2 1 - E})$
$4 = O(n^{0 - E})$
$4 <= c\ n^{0 - E}$;
$4 <= c\ n^{-E}$;
$4 <= c/n^{E}$; Não existe um E maior do que zero que torne verdadeiro, pois um lado se manterá constante enquanto o outro tenderá a zero.
FALSO

CASO 2: $f(n) = \Theta\ (n^{\log_b a})$
$4 = \Theta\ (n^{\log_2 1})$
$4 = \Theta\ (n^{0})$
$c1\ n^{0} <= 4 <= c2\ n^{0}$
$c1\ 1 <= 4 <= c2\ 1$; $c1 = c2 = 4$

VERDADEIRO

$T(N) = \Theta(n^{\log_b a} \log n)$
$T(N) = \Theta(n^{\log_2 1} \log n)$
$T(N) = \Theta(n^{0} \log n)$
$T(N) = \Theta(1 \log n)$
$T(N) = \Theta(\log n)$

Esse custo $\Theta(\log n)$ deve ser observado com cuidado, pois a busca binária tem mais de um caso em que a recursão pode ser interrompida. Por isso, o custo $\Theta(\log n)$ está relacionado somente com o pior caso. O melhor caso de execução da busca binária é aquele em que o elemento central do array é o elemento de busca, assim, seria necessário um número de passos constante para terminar a busca. Então, o melhor caso do subAlgoritmo02 é $\Theta(4)$.

Finalmente, podemos proceder para a análise do algoritmo Principal.
Melhor Caso
$\Theta(n) + \Theta(4) = \Theta(n)$

Pior Caso
$\Theta(n^2) + \Theta(\log n) = \Theta(n^2)$