Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Faculdade de Computação
Análise de Algoritmos

Lista de Exercícios
Fundamentos

1. Sejam f, g e h funções reais positivas da variável inteira n. Com respeito às notações assintóticas, avalie as afirmativas abaixo.

I. $f = O(g)$ se e somente se $g = \omega(f)$.

II. $f = \Theta(g)$ se e somente se $f = O(g)$ e $f = \Omega(g)$.

III. $f = o(g)$ e $g = o(h)$ implicam $f = o(h)$.

IV. $nlog(n) + k = O(n)$, dado que k é uma constante positiva.

A análise permite concluir que somente

(A) a afirmativa III é verdadeira.
(B) as afirmativas I e IV são verdadeiras.
(C) as afirmativas II e III são verdadeiras.
(D) as afirmativas I, II e III são verdadeiras.
(E) as afirmativas II, III e IV são verdadeiras.

2. Um limite inferior para um problema P é uma função f, tal que a complexidade no tempo de pior caso de qualquer algoritmo que resolva P é $\Omega(f)$. Isto quer dizer que

(A) todo algoritmo que resolve P efetua $\Theta(f)$ passos.
(B) se existir um algoritmo A que resolve P com complexidade $O(f)$, então A é denominado algoritmo ótimo para P.
(C) todo algoritmo que resolve P é recursivo.
(D) todo algoritmo que resolve P é interativo.
(E) todo algoritmo que resolve P tem complexidade linear no mínimo.

3. O algoritmo para calcular um polinômio $a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$, quando $x = c$, pode ser expresso em pseudocódigo por:

```
 Polinomio(A, n, c)
power = 1
y = A[0]
para i = 1 até n faça
   power = power * c
   y = y + A[i] * power
retorne (y)
```

Seja $T(n)$ o tempo de execução do algoritmo acima descrito para as entradas $A[a_0...a_n]$, n e c. A ordem de $T(n)$ usando a notação Little-o é

(A) $T(n) = o(c)$.
(B) $T(n) = o(log(n))$.
(C) $T(n) = o(\sqrt{n})$.
(D) $T(n) = o(n)$.
(E) $T(n) = o(n^2)$.

4. Considere o algoritmo abaixo.

```
 PROC(n)
se n <= 1 então
   retorne (2)
senão
   retorne (PROC(n/2) + PROC(n/2))
fim se
```

Assinale a alternativa que indica o valor retornado pelo algoritmo considerando a entrada $n = 64$.

(A) 128.
(B) 130.
(C) 1.024.
(D) 4.096.
(E) 4.160.

5. O tempo de execução $T(n)$ de um algoritmo, em que n é o tamanho da entrada, é dado pela equação de recorrência $T(n) = 8T(n/2) + qn$ para todo $n > 1$. Dado que $T(1) = p$, e que p e q são constantes arbitrárias, a complexidade do algoritmo é

(A) $\Theta(log(n))$.
(B) $\Theta(n)$.
(C) $\Theta(nlog(n))$.
(D) $\Theta(n^2)$.
(E) $\Theta(n^3)$.

6. Resolva as relações de recorrência abaixo pelo Teorema Mestre.

(a) $T(1) = 1$.
$T(n) = 4T(n/4) + n$ para $n > 1$.

onde $a = 4$, $b = 4$, $f(n) = n$ e $n^{log_b a} = n^{log_4 4} = n$.

O caso 2 se aplica porque $f(n) = \Theta(n^{log_b a}) = \Theta(n)$.

Logo, a solução é $T(n) = \Theta(nlog(n))$.

(b) $T(1) = 1$.
$T(n) = 7T(n/2) + n^2$ para $n > 1$.

Considerando que $2 < log_2 7 < 3$:

$a = 7; b = 2; f(n) = n^2$

Caso 1: $f(n) = O(n^{log_b a - \varepsilon})$ então $T(n) = \Theta(n^{log_b a})$ para $\varepsilon > 0$

$f(n) = O(n^{log_2 7 - \varepsilon})$ = então $T(n) = \Theta(n^{log_2 7})$

Escolhendo um $\epsilon = log_2 7 - 2$, por exemplo.

(c) $T(1) = 1$.
$T(n) = 3T(n/9) + 2n$ para $n > 1$.

$a = 3; b = 9; f(n) = 2n$

Caso 3: $f(n) = \Omega(n^{\log_b a+\varepsilon})$ e $af\left({n}/{b}\right) \leq cf(n)$ para $\varepsilon > 0$ e $c < 1$ então

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

$f(n) = \Omega(n^{\log_9 3+\varepsilon}) = \Omega(n^{0.5+\varepsilon}) = \Omega(n)$ para $\varepsilon = 0.5$

$3f\left({n}/{9}\right) \leq cf(n)$

$3 \times {2n}/{9} \leq c2n$

${6n}/{9} \leq c2n$

${6}/{18} \leq c$

${1}/{3} \leq c$

Portanto, $c = {1}/{3}$ e $T(n) = \Theta(2\text{n}) = \Theta(\text{n})$

7. As definições recursivas apresentadas abaixo descrevem o tempo de execução de dois algoritmos recursivos: A e B:

$T_A(1) = 1$ e $T_A(n) = 2T_A(n-1) + 2$ para $n \geq 2$.

$T_B(1) = 1$ e $T_B(n) = T_B(n/2) + n$ para $n \geq 2$.

Assinale a alternativa correta.

(A) Os algoritmos não são eficientes.
(B) Os algoritmos são assintoticamente equivalentes.
(C) O algoritmo B tem complexidade exponencial no tempo.
(D) O algoritmo A é mais eficiente assintoticamente que o algoritmo B.
(E) O algoritmo B é mais eficiente assintoticamente que o algoritmo A.

As relações de recorrência serão resolvidas abaixo.

$T_A(1) = 1$ e $T_A(n) = 2T_A(n-1) + 2$ para $n \geq 2$.

Expandir:
$k = 1$: $T_A(n) = 2T_A(n-1) + 2$
$k = 2$: $T_A(n) = 2[2T_A(n-2) + 2] + 2 = 4T_A(n-2) + 6$
$k = 3$: $T_A(n) = 4[2T_A(n-3) + 2] + 6 = 8T_A(n-3) + 14$

Conjecturar: Após k expansões, temos $T_A(n) = 2^k T_A(n-k) + 2^{k+1} - 2$.

A expansão irá parar quando $k = n - 1$, isso porque a base da recursividade é definida para 1.

$T_A(n) = 2^{n-1} T_A(n - n + 1) + 2^{n-1+1} - 2$
$T_A(n) = 2^{n-1} T_A(1) + 2^n - 2$
$T_A(n) = 2^{n-1} + 2^n - 2$

O algoritmo A é exponencial.

$T_B(1) = 1$ e $T_B(n) = T_B(n/2) + n$ para $n \geq 2$.

Pelo caso 3 do Teorema Mestre, temos que $T_B(n) = \Theta(n)$. Dessa forma, o algoritmo B é linear e, consequentemente, mais eficiente assintoticamente que o algoritmo A.

8. Considere o algoritmo A abaixo.

```
Algoritmo A (n)
Entrada: n, inteiro, n ≥ 1.
{
    se (n = 1)
        retornar 1;
    senão
        retornar 2*A(n/2) + 1;
}
```

A complexidade no tempo de pior caso do algoritmo A é

(A) linear.
(B) logarítmica.
(C) quadrática.
(D) exponencial.
(E) $nlog(n)$.

Este algoritmo tem como redução uma chamada a $A(n/2)$, ou seja, a solução de um problema de tamanho igual a n é reduzida à solução de um problema de tamanho igual a $n/2$ mais algum tempo constante, digamos c_1, para, então, multiplicar o resultado por 2 e somar 1. A solução do problema para o caso base é resolvido em um tempo constante, digamos c_2. Assim, a relação de recorrência que descreve o comportamento do algoritmo pode ser assim escrita:

$T(1) = c_2$ e

$T(n) = T(n/2) + c_1$, onde c_1 e c_2 são constantes.

Expandir:

$k = 1$: $T(n) = T(\frac{n}{2}) + 1$

$k = 2$: $T(n) = [T((\frac{n}{4}) + 1] + 1 = T(\frac{n}{4}) + 2$

$k = 3$: $T(n) = [T((\frac{n}{8}) + 1] + 2 = T(\frac{n}{8}) + 3$

Conjecturar: Após k expansões, temos $T(n) = T(\frac{n}{2^k}) + k$.

A expansão irá parar quando $n = 2^k$, isso porque a base da recursividade é definida para 1.

$T(n) = T(\frac{n}{2^k}) + log(n) = T(1) + log(n) = 1 + log(n)$

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

O algoritmo é logarítmico em pior caso.

9. O algoritmo recursivo abaixo soma os n primeiros números naturais.

```
Algoritmo Soma (n)
Entrada: n, inteiro, n > 0.
{
   se (n = 1)
       retornar 1;
   senão
       retornar Soma (n - 1) + n;
}
```

A complexidade no tempo do algoritmo é

(A) linear.
(B) logarítmica.
(C) quadrática.
(D) exponencial.
(E) $nlog(n)$.

10. Suponha que f e g são funções reais positivas da variável inteira n. Verifique se as seguintes afirmações são verdadeiras ou falsas. Justifique sua resposta caso a afirmativa seja falsa.

(a) Se $f(n) = O(g(n))$, então $2^{f(n)} = O(2^{g(n)})$.

Falsa. Por exemplo, podemos definir $f(n) = 2n$ e $g(n) = n$.

(b) $f(n) = O((f(n))^2)$.

Falsa. Por exemplo, podemos definir $f(n) = 1/n$.

(c) Se $f(n) = O(g(n))$, então $g(n) = \Omega(f(n))$.

Verdadeira.

(d) $f(n) + O(f(n)) = \Theta(f(n))$.

Verdadeira.

11. Escreva um algoritmo (em pseudocódigo) que realize busca binária de forma iterativa e o implemente numa linguagem de programação a sua escolha. Construa um gráfico mostrando a relação valor de entrada x tempo de execução do algoritmo. Considerando uma análise assintótica em pior caso, explique se o desempenho do algoritmo é superior, inferior ou igual ao do algoritmo que implementa busca binária de forma recursiva.

O algoritmo de busca binária recebe um vetor ordenado A de tamanho n, um valor v, e um intervalo $[low..high]$ do vetor, onde o valor v será procurado. O algoritmo compara v com o ponto médio do intervalo e decide por eliminar metade do intervalo para a próxima interação. Abaixo, duas versões da busca binária: iterativa e recursiva. Cada uma delas retorna um índice i tal que $A[i] = v$, ou NIL se o intervalo não conter v. A chamada inicial deve ter os seguintes parâmetros: A, v, 1, n.

```
ITERATIVE-BINARY-SEARCH(A, v, low, high)
while (low <= high) do
     mid = (low + high)/2
     if v = A[mid]
        then return mid
     if v > A[mid]
        then low = mid + 1
        else high = mid - 1
return NIL
```

```
RECURSIVE-BINARY-SEARCH(A, v, low, high)
if low > high
   then return NIL
mid = (low + high)/2
if v = A[mid]
   then return mid
if v > A[mid]
   then return RECURSIVE-BINARY-SEARCH(A, v, mid + 1, high)
   else return RECURSIVE-BINARY-SEARCH(A, v, low, mid - 1)
```

Ambas as versões terminam com a busca mal-sucedida quando o intervalo é vazio (i.e. $low > high$, indicando que v não encontra-se no intervalo). A busca termina com sucesso quando v é encontrado no intervalo. Baseada na comparação de v com o elemento do meio no intervalo de busca, a pesquisa continua com o intervalo reduzido pela metade. Assim, a versão iterativa tem eficiência logarítmica e a recursiva tem tempo de execução igual a $T(n) = T(n/2)+\Theta(1)$, cuja solução pelo Método Mestre é $T(n) = \Theta(log(n))$. Apesar de apresentarem a mesma complexidade no tempo, o RECURSIVE-BINARY-SEARCH exige maior esforço computacional em função das chamadas recursivas.

Para as questões **12** e **13**, entregue os seguintes itens considerando o algoritmo implementado para resolver o problema computacional:

- Uma captura de tela que mostre a compilação correta na plataforma de teste;
- O cálculo da complexidade no tempo usando notação assintótica; e
- Um gráfico ilustrando a análise empírica, ou seja, a relação valor de entrada x tempo de execução.

12. Resolva o seguinte problema computacional:

Problema: Ajude a Federação (#1588)
https://www.beecrowd.com.br/judge/pt/problems/view/1588

A complexidade no tempo do algoritmo depende do número de times (N), do número de jogos (M) e do algoritmo de ordenação usado para ordenar os times. Um método simples que pode ser usado é o Bubblesort, com eficiência quadrática $\Theta(N^2)$. Nesse caso, a eficiência do algoritmo seria $O(M + N^2)$. Pode-se optar também por um método mais rápido, como o Quicksort, que entregaria uma eficiência média para o algoritmo de $O(M + NlogN)$.

13. Resolva o seguinte problema computacional de forma **recursiva**:

Problema: A Lenda de Flavious Josephus (#1030)
https://www.beecrowd.com.br/judge/pt/problems/view/1030

Uma função recursiva para resolver o problema pode ser escrita da seguinte forma:

```
int flavious(int n, int k)
{
    if(n == 1) return 0;
    return (flavious(n-1,k) + k) % n;
}
...
flavious(n, k) + 1; // Resultado final
```

A relação de recorrência relacionada com a complexidade do algoritmo pode ser definida da seguinte forma:

$$\begin{cases} T(n) = \theta(1), & \text{se } n = 1. \\ T(n) = T(n-1) + \theta(1), & \text{se } n > 1. \end{cases}$$

Resolvendo a relação de recorrência pelo método de expandir, conjecturar e verificar:

Expandir:

$$\begin{aligned} k = 1 &: T(n) = T(n-1) + 1 \\ k = 2 &: T(n) = T(n-2) + 1 + 1 \\ k = 3 &: T(n) = T(n-3) + 1 + 1 + 1 \end{aligned}$$

Conjecturar: Após k expansões: $T(n) = T(n-k) + k$. Fazendo $n - k = 1$:

$$\begin{aligned} &k = n - 1 \\ &T(n) = T(1) + n - 1 \\ &T(n) = 1 + n - 1 \\ &T(n) = n \end{aligned}$$

Portanto, a complexidade no tempo da função recursiva implementada para resolver o problema é constante para $n = 1$ e $\Theta(n)$ para $n > 1$.