Análise de Algoritmos
Recorrências e Algoritmos Recursivos
Exercícios

Nelson Cruz Sampaio Neto
nelsonneto@ufpa.br

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

8 de setembro de 2023

Exercícios

1. Encontre a solução fechada para a sequência T definida por $T(1) = 1$ e $T(n) = T(n-1) + 3$ para $n \geq 2$.

2. Encontre a solução fechada para a sequência T definida por $T(1) = 7$ e $T(n) = 2T(n-1) + 1$ para $n \geq 2$.

3. Encontre a solução fechada para a sequência T definida por $T(1) = 1$ e $T(n) = T(\frac{n}{2}) + 1$ para $n \geq 2$.

4. Defina recursivamente cada uma das sequências abaixo.

   a) 1, 3, 5, 7, 9, ...
   b) 1, 4, 9, 16, 25, ...
   c) 1, 3, 7, 15, 31, 63.

Exercício 1

- **Expandir**:
  $k = 1$: $T(n) = T(n-1) + 3$
  $k = 2$: $T(n) = [T(n-2) + 3] + 3 = T(n-2) + 6$
  $k = 3$: $T(n) = [T(n-3) + 3] + 6 = T(n-3) + 9$

- **Conjecturar**: Após k expansões, $T(n) = T(n-k) + 3k$.

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

  $$T(n) = T(n-(n-1)) + 3(n-1)$$
  $$T(n) = T(1) + 3n - 3$$
  $$T(n) = 1 + 3n - 3$$
  $$T(n) = 3n - 2$$

- **Verificar**: Provar a conjectura via indução matemática.

Exercício 1

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 3n - 2$.

  $T(1) = 3 * 1 - 2 = 1$ OK!

  Agora, queremos demonstrar que $T(n+1) = 3(n+1) - 2$.

  Da definição de T e usando a hipótese indutiva:

$$
\begin{aligned}
T(n+1) &= T(n+1-1) + 3 \\
T(n+1) &= T(n) + 3 \\
T(n+1) &= 3n - 2 + 3 \\
T(n+1) &= 3n + 3 - 2 \\
T(n+1) &= 3(n+1) - 2 \quad \text{c.q.d}
\end{aligned}
$$

Exercício 2

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1) + 1$
  $k = 2$: $T(n) = 2[2T(n-2) + 1] + 1 = 4T(n-2) + 3$
  $k = 3$: $T(n) = 4[2T(n-3) + 1] + 3 = 8T(n-3) + 7$

- **Conjecturar**: Após k expansões, $T(n) = 2^k T(n-k) + 2^k - 1$.

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

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

  $$T(n) = 2^{n-1} \cdot 7 + 2^{n-1} - 1 = 2^{n-1} \cdot 8 - 1$$

  $$T(n) = 2^{n-1} \cdot 2^3 - 1 = 2^{n+2} - 1$$

- **Verificar**: Provar a conjectura via indução matemática.

Exercício 2

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^{n+2} - 1$.

  $T(1) = 2^3 - 1 = 7$ OK!

  Agora, queremos demonstrar que $T(n+1) = 2^{n+3} - 1$.

  Da definição de T e usando a hipótese indutiva:

  $$T(n+1) = 2T(n+1-1) + 1$$
  $$T(n+1) = 2T(n) + 1$$
  $$T(n+1) = 2(2^{n+2} - 1) + 1$$
  $$T(n+1) = 2^{n+3} - 1 \quad \text{c.q.d}$$

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

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

  Logo, $T(n) = T(1) + log(n) = 1 + log(n)$.

Exercício 3

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log(n)$.

  $T(1) = 1 + log(1) = 1 + 0 = 1$ OK!

  Agora, queremos demonstrar que $T(2n) = 1 + log(2n)$.

  Da definição de T e usando a hipótese indutiva:

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

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

  $T(2n) = 1 + log(n) + 1$

  $T(2n) = 1 + log(n) + log(2)$

  $T(2n) = 1 + log(2n)$ c.q.d

Exercício 4

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

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

c) $T(1) = 1$ e
$T(n) = 2T(n-1) + 1$ para $2 \leq n \leq 6$.

Exercício 5

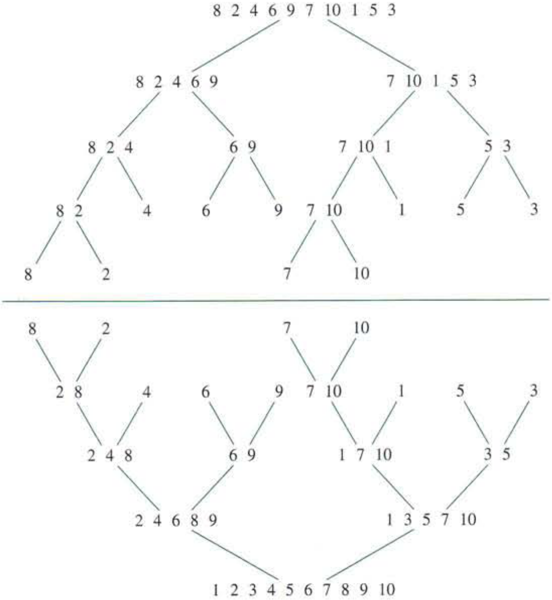
- Algoritmo para ordenação por intercalação (MERGE-SORT).

- Recebe como entrada o vetor A, a ser ordenado, e os índices do primeiro p e do último r elementos.

```
MERGE-SORT (A, p, r)
se (p < r) então
    q = (p + r) / 2
    MERGE-SORT (A, p, q)
    MERGE-SORT (A, q + 1, r)
    MERGE (A, p, q, r)
```

- Dado que a função MERGE é $\Theta(n)$, qual é a complexidade no tempo do algoritmo MERGE-SORT?

Exercício 5


8 2 4 6 9 7 10 1 5 3
8 2 4 6 9
7 10 1 5 3
8 2 4
6 9
7 10 1
5 3
8 2
4
6
9
7 10
1
5
3
8
2
7
10
8
2
7
10
2 8
4
6
9
7 10
1
5
3
2 4 8
6 9
1 7 10
3 5
2 4 6 8 9
1 3 5 7 10
1 2 3 4 5 6 7 8 9 10

Exercício 5

- Passo base: Um vetor com 1 (um) elemento já está ordenado. Isso acontece quando $p = r$.

  $T(1) = \Theta(1)$ ou

  $T(1) = 1$.

- Passo recorrente: Um vetor com mais de 1 (um) elemento.

  $T(n) = \Theta(1) + T(\frac{n}{2}) + T(\frac{n}{2}) + \Theta(n)$.

  A soma das funções $\Theta(n)$ e $\Theta(1)$ é uma função linear de n, ou seja, $\Theta(n)$. Logo,

  $T(n) = 2T(\frac{n}{2}) + n$ para $n > 1$.

Exercício 5

- **Expandir**:

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

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

  $k = 3$: $T(n) = 4[2T(\frac{n}{8}) + \frac{n}{4}] + 2n = 8T(\frac{n}{8}) + 3n$

- **Conjecturar**: $T(n) = 2^k T(\frac{n}{2^k}) + kn$.

- A expansão irá parar quando $n = 2^k$, ou seja, $k = log_2(n)$, isso porque a base da recursividade é definida para 1. Logo,

  $T(n) = n\ T(\frac{n}{n}) + nlog_2(n) = n\ T(1) + nlog_2(n)$

  $T(n) = n + nlog_2(n)$

- Com isso, podemos afirmar que a complexidade no tempo do MERGE-SORT é $O(nlog_2(n))$.

Exercício 6

- Encontre a complexidade no tempo de um algoritmo com a seguinte definição recursiva:

$T(1) = 0$ e

$T(n) = T(n-1) + c$ para $n > 1$.

Considere que c é uma constante.

Exercício 6

- **Expandir**:

  $k = 1$: $T(n) = T(n-1) + c$

  $k = 2$: $T(n) = T(n-2) + c + c = T(n) = T(n-2) + 2c$

  $k = 3$: $T(n) = T(n-3) + c + 2c = T(n) = T(n-3) + 3c$

- **Conjecturar**: $T(n) = T(n-k) + kc$.

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

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

Exercício 7

- Considere o algoritmo TESTE que recebe como parâmetro de entrada um número inteiro não-negativo n.

```
TESTE (n)
1. se (n = 0) então retorne (8)
2. senão retorne (TESTE(n - 1) + 2)
```

- Qual é o valor retornado pelo algoritmo para $n = 8$?

- Qual é a complexidade no tempo do algoritmo?

Exercício 7

- Para calcular o valor de retorno:

  $T(0) = 8$ e

  $T(n) = T(n-1) + 2$ para $n > 0$.

- Para encontrar a complexidade do algoritmo:

  $T(0) = \Theta(1)$ e

  $T(n) = T(n-1) + \Theta(1)$ para $n > 0$.

Exercício 7

- **Expandir**:

  $k = 1$: $T(n) = T(n-1) + 2$

  $k = 2$: $T(n) = [T(n-2) + 2] + 2 = T(n-2) + 4$

  $k = 3$: $T(n) = [T(n-3) + 2] + 4 = T(n-3) + 6$

- **Conjecturar**: $T(n) = T(n-k) + 2k$.

- A expansão irá parar quando $k = n$, isso porque a base da recursividade é definida para 0. Logo,

  $$T(n) = T(n-n) + 2n = T(0) + 2n = 2n + 8.$$

  Por exemplo, $T(8) = 2 \cdot 8 + 8 = 24$.

- A complexidade no tempo do algoritmo é $O(n)$.

Exercício 8

- Considere o seguinte pseudo-código:

```
Subrotina Rec( parâmetro n)
if n ≤ 1
        Stop
else
        Ative Processo X
        Rec(n/2)
```

- Seja $T(n)$ o número de ativações do processo X.
- Quantas vezes o processo X será ativado para uma entrada n?

Exercício 8

- A definição recursiva do algoritmo assume a fórmula abaixo. Note que para $n \leq 1$ o processo não é ativado.

  $T(1) = 0$ e

  $T(n) = T(\frac{n}{2}) + 1$ para $n > 1$.

- Resolvendo a relação de recorrência, tem-se: $T(n) = log_2(n)$.

  Logo, o processo será ativado $log_2(n)$ vezes para $n > 1$.

- Como a operação básica do algoritmo é a quantidade de ativações do processo X, é possível afirmar que a sua complexidade no tempo é logarítmica em n.

Exercício 9

- Dois algoritmos recursivos têm sua complexidade no tempo expressa pelas definições recursivas abaixo. Qual desses algoritmos é o mais eficiente assintoticamente?

- Algoritmo 1:

  $T(1) = 1$ e
  $T(n) = T(\frac{n}{3}) + n$ para $n > 1$.

- Algoritmo 2:

  $T(1) = 1$ e
  $T(n) = 3T(\frac{n}{4}) + n^2$ para $n > 1$.

Exercício 9

- O Algoritmo 1 é mais eficiente que o Algoritmo 2.

Considerando $T(n) = T(\frac{n}{3}) + n$,

O caso 3 se aplica porque $f(n) = \Omega(n^{log_3 1+\epsilon}) = \Omega(n)$, onde $\epsilon = 1$ e $af(n/b) = n/3 \leq cf(n) = n/3$, para $c = 1/3$ e $n \geq 0$. Logo, a solução é $T(n) = \Theta(f(n)) = \Theta(n)$.

Considerando $T(n) = 3T(\frac{n}{4}) + n^2$,

O caso 3 se aplica porque $f(n) = \Omega(n^{log_4 3+\epsilon}) = \Omega(n)$, onde $\epsilon \approx 0,2$ e $af(n/b) = 3n^2/16 \leq cf(n) = 3n^2/16$, para $c = 3/16$ e $n \geq 0$. A solução é $T(n) = \Theta(f(n)) = \Theta(n^2)$.

Exercício 10

- Considere o algoritmo abaixo.

```
    void pesquisa(n) {
(1)   if (n <= 1)
(2)     'inspecione elemento' e termine
      else {
(3)     para cada um dos n elementos 'inspecione elemento';
(4)     pesquisa(n/3);
      }
    }
```

- A definição recursiva do algoritmo assume a fórmula abaixo, onde a sua operação básica é o número de inspeções.

  $T(1) = 1$ e

  $T(n) = T(\frac{n}{3}) + n$ para $n > 1$.

- O caso 3 do Teorema Mestre se aplica, logo, a complexidade no tempo do algoritmo é linear.

Exercício 10

- **Expandir**:
  $k = 1$: $T(n) = T(\frac{n}{3}) + n$
  $k = 2$: $T(n) = [T(\frac{n}{9}) + \frac{n}{3}] + n = T(\frac{n}{9}) + \frac{n}{3} + n$
  $k = 3$: $T(n) = [T(\frac{n}{27}) + \frac{n}{9}] + \frac{n}{3} + n = T(\frac{n}{27}) + \frac{n}{9} + \frac{n}{3} + n$

- **Conjecturar**:
  Após k expansões, temos $T(n) = T(\frac{n}{3^k}) + n \sum_{i=0}^{k-1} (\frac{1}{3})^i$.

- Observando a expansão, ela irá parar quando $n = 3^k$, isso porque a base da recursividade é definida para 1 (um).

- Logo, $T(n) = T(1) + n\,[\frac{1-(\frac{1}{3})^k}{1-\frac{1}{3}}] = \frac{3n}{2} - \frac{1}{2}$.

Exercício 11

- Observe a função recursiva a seguir.

```
function Prova (N : integer) : integer;
begin
  if N = 0 then Prova :=  0
  else Prova := N * 2 - 1 + Prova (N - 1);
end;
```

- Considerando-se que essa função sempre será chamada com N contendo inteiros positivos, o seu valor de retorno será:

  (a) O fatorial do valor armazenado em N.
  (b) O valor armazenado em N elevado ao quadrado.
  (c) O somatório dos N primeiros números inteiros positivos.
  (d) O somatório dos N primeiros números pares positivos.
  (e) 2 elevado ao valor armazenado em N.

Exercício 11

- Passo base: $T(0) = 0$

- **Expandir**:
  $k = 1$: $T(n) = T(n-1) + 2n - 1$
  $k = 2$: $T(n) = [T(n-2) + 2(n-1) - 1] + 2n - 1 = T(n-2) + 4n - 4$
  $k = 3$: $T(n) = [T(n-3) + 2(n-2) - 1] + 4n - 4 = T(n-3) + 6n - 9$
  $k = 4$: $T(n) = [T(n-4) + 2(n-3) - 1] + 6n - 9 = T(n-4) + 8n - 16$

- **Conjecturar**:
  Após k expansões, temos $T(n) = T(n-k) + 2kn - k^2$.

- Observando a expansão, ela irá parar quando $k = n$, isso porque a base da recursividade é definida para 0 (zero).

- Logo, $T(n) = T(0) + 2n^2 - n^2 = n^2$.

Exercício 12

- A função recursiva abaixo calcula a potência de um número.

```
int pot(int base, int exp)
{
  if (!exp)
    return 1;
  /* else */
  return (base * pot(base, exp-1));
}
```

- Qual é a complexidade no tempo da função?

Exercício 12

- Análise de complexidade no tempo, onde n é o expoente:

  $T(0) = \Theta(1)$

  $T(n) = T(n-1) + \Theta(1)$ para $n > 0$.

- Segue a mesma ideia do Fatorial recursivo: A operação básica é o número de multiplicações indicado por $T(n)$.

- A complexidade no tempo do algoritmo é $O(n)$.

- Um equívoco muito comum é definir o passo recorrente como $T(n) = c \cdot T(n-1)$, onde c é o valor da variável "base".

- Não é correto porque cada chamada da função não causa c novas chamadas de si mesma.