Projeto e Análise de Algoritmos

Lista de Exercícios (Algoritmos de ordenação)

Flávio L. C. de Moura*

Teorema 0.1. *Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função assintoticamente positiva, e $T(n)$ definida nos inteiros não-negativos pela recorrência $T(n) = a.T(n/b) + f(n)$, onde n/b deve ser interpretado como $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$. Então $T(n)$ tem as seguintes cotas assintóticas:*

1. *Se $f(n) = O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$;*
2. *Se $f(n) = \Theta(n^{\log_b a})$, então $T(n) = \Theta(n^{\log_b a}. \lg n)$;*
3. *Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguma constante $\epsilon > 0$, e se $a.f(n/b) \leq c.f(n)$ para alguma constante $c < 1$, então para todo n suficientemente grande, temos que $T(n) = \Theta(f(n))$.*

1 Exercícios

1. Sejam $f(n), g(n)$ e $h(n)$ funções dos inteiros não-negativos nos reais positivos. Mostre que se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ então $f(n) = O(h(n))$.
2. Sejam $f(n)$, $g(n)$ e $h(n)$ funções não-negativas tais que $f(n) = O(h(n))$ e $g(n) = O(h(n))$. Prove que $f(n) + g(n) = O(h(n))$.
3. Resolva as seguintes relações de recorrência:
   (a) $T(1) = 1$, $T(n) = 3T(n/2) + n^2$, $n \geq 2$
   (b) $T(1) = 1$, $T(n) = 2T(n/2) + n$, $n \geq 2$
   (c) $T(1) \in \Theta(1)$, $T(n) = 3T(n/3 + 5) + n/2$
   (d) $T(1) = 1$, $T(n) = 2T(n-1) + 1$, $n \geq 2$
   (e) $T(1) \in \Theta(1)$, $T(n) = 9T(n/3) + n$
   (f) $T(1) \in \Theta(1)$, $T(n) = T(2n/3) + 1$
   (g) $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + 1$
   (h) $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + \sqrt{n}$
   (i) $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + \sqrt{n} \lg^2 n$
   (j) $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + n$

*flaviomoura@unb.br

(k) $T(1) \in \Theta(1)$, $T(n) = 2T(n/4) + n^2$

(l) $T(1) \in \Theta(1)$, $T(n) = 3T(n/2) + nln(n)$

(m) $T(1) \in \Theta(1)$, $T(n) = 3T(n/4) + nln(n)$

(n) $T(1) \in \Theta(1)$, $T(n) = 2T(n/2) + nln(n)$

(o) $T(1) \in \Theta(1)$, $T(n) = 2T(n/2) + n/ln(n)$

(p) $T(1) \in \Theta(1)$, $T(n) = T(n-1) + 1/n$

(q) $T(1) \in \Theta(1)$, $T(n) = T(n-1) + ln(n)$

(r) $T(1) \in \Theta(1)$, $T(n) = \sqrt{n}T(\sqrt{n}) + n$

(s) $T(n) = 8T(n/2) + \Theta(n^2)$

(t) $T(n) = 8T(n/2) + \Theta(1)$

(u) $T(n) = 7T(n/2) + \Theta(n^2)$

4. Considere o pseudocódigo a seguir:

Algorithm 1: BubbleSort($A[0..n-1]$)

```
1 for i = 0 to n − 2 do
2 |   for j = 0 to n − 2 − i do
3 |   |   if A[j + 1] < A[j] then
4 |   |   |   swap A[j] and A[j + 1];
5 |   |   end
6 |   end
7 end
```

(a) Escreva uma invariante para mostrar que este algoritmo é correto.

(b) Prove que este algoritmo é correto.

(c) Faça a análise assintótica deste algoritmo.

5. Considere o pseudocódigo a seguir:

Algorithm 2: BubbleSort2($A[0..n-1]$)

```
1 for i = 0 to n − 2 do
2 |   for j = n − 1 downto i + 1 do
3 |   |   if A[j] < A[j − 1] then
4 |   |   |   swap A[j] and A[j − 1];
5 |   |   end
6 |   end
7 end
```

(a) Escreva uma invariante para mostrar que este algoritmo é correto.

(b) Prove que este algoritmo é correto.

(c) Faça a análise assintótica deste algoritmo.

6. Considere o pseudocódigo a seguir:

Algorithm 3: SelectionSort($A[0..n-1]$)

```
1 for i = 0 to n − 2 do
2 |   min ← i;
3 |   for j = i + 1 to n − 1 do
4 |   |   if A[j] < A[min] then
5 |   |   |   min ← j;
6 |   |   end
7 |   end
8 |   swap A[i] and A[min];
9 end
```

(a) Escreva uma invariante para mostrar que este algoritmo é correto.

(b) Prove que este algoritmo é correto.

(c) Faça a análise assintótica deste algoritmo.

7. Considere o pseudocódigo a seguir:

Algorithm 4: InsertionSort($A[0..n-1]$)

```
1 for j = 1 to n − 1 do
2 |   key ← A[j];
3 |   i ← j − 1;
4 |   while i ≥ 0 and A[i] > key do
5 |   |   A[i + 1] ← A[i];
6 |   |   i ← i − 1;
7 |   end
8 |   A[i + 1] ← key;
9 end
```

(a) Escreva uma invariante para mostrar que este algoritmo é correto.

(b) Prove que este algoritmo é correto.

(c) Faça a análise assintótica deste algoritmo.

8. Mostre que, em um *heap* com n elementos e raiz $A[i]$, cada uma das subárvores com raiz em $2i$ e $2i + 1$ têm, no máximo, $2n/3$ elementos.

9. Considere o pseudocódigo a seguir:

Algorithm 5: Max-Heapify($A[1..n], i$)

```
1  l = 2i;
2  r = 2i + 1;
3  if l ≤ A.heap-size and A[l] > A[i] then
4  |  largest = l;
5  end
6  else
7  |  largest = i;
8  end
9  if r ≤ A.heap-size and A[r] > A[largest] then
10 |  largest = r;
11 end
12 if largest ≠ i then
13 |  exchange A[i] with A[largest];
14 |  Max-Heapify(A,largest);
15 end
```

(a) Faça a análise assintótica deste algoritmo.

10. Considere o pseudocódigo a seguir:

Algorithm 6: Build-Max-Heap($A[1..n]$)

```
1  A.heap-size = A.length;
2  for i = ⌊A.length/2⌋ downto 1 do
3  |  Max-Heapify(A,i);
4  end
```

(a) Faça a análise assintótica deste algoritmo.

11. Considere o pseudocódigo a seguir:

Algorithm 7: Heapsort($A[1..n]$)

```
1  Build-Max-Heap(A);
2  for i = A.length downto 2 do
3  |  exchange A[1] with A[i];
4  |  A.heap-size = A.heap-size - 1;
5  |  Max-Heapify(A,1);
6  end
```

(a) Faça a análise assintótica deste algoritmo.

12. Mostre como podemos ordenar n inteiros contidos no intervalo de 0 a n^3-1 em tempo linear, ou seja, em tempo $O(n)$.