Solutions du CLRS 3 *ed.*

Exercises : (23/12/957)
Problems : (0/6/158)

10 juin 2017

Table des matières

Liste des Algorithmes

1 The Role of Algorithms in Computing

1.1 Algorithms

1.1-1 *Give a real-world example that requires sorting or a real-world example that requires computing a convex hull.*

Solution : ■

1.1-2 *Other than speed, what other measures of efficiency might one use in a real-world setting?*

Solution : ■

1.1-3 *Select a data structure that you have seen previously, and discuss its strengths and limitations.*

Solution : ■

1.1-4 *How are the shortest-path and traveling-salesman problems given above similar? How are they different?*

Solution : ■

1.1-5 *Come up with a real-world problem in which only the best solution will do. Then come up with one in which a solution that is "approximately" the best is good enough.*

1.2 Algorithms as a technology

1.2-1 *Give an example of an application that requires algorithmic content at the application level, and discuss the function of the algorithms involved.*

Solution : ■

1.2-2 *Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merg sort runs in $64n\lg n$ steps. For which values of n does insertion sort beat merge sort?*

Solution : ■

1.2-3 *What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?*

Solution : ■

1.3 Problems

1-1 ***Comparison of running times***

For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

Solution : ■

2 Getting Started

2.1 Insertion Sort

2.1-1 *Using Figure 2.2 as a model, illuſtrate the operation of* Insertion-Sort *on the array* $A = \langle 31, 41, 59, 26, 41, 58 \rangle$.

Solution : ■

2.1-2 *Rewrite the* Insertion-Sort *procedure to sort into non-increasing inſtead of non-decreasing order.*

Solution :

Algorithme 1. Decr-Insertion-Sort (A)

```
1  for $j = 2$ to $A.length$
2      $key = A[j]$
3      // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.
4      $i = j - 1$
5      while $i > 0$ and $A[i] < key$
6          $A[i+1] = A[i]$
7          $i = i - 1$
8      $A[i+1] = key$
```

■

2.1-3 *Consider the* ***searching problem*** *:*

Input : *A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a value v.*

Output : *An index i such that $v = A[i]$ or the special value* nil *if v does not appear in A.*

Write pseudocode for ***linear search****, which scans through the sequence, looking for v. Using a loop, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.*

Solution :

Algorithme 2. Linear-Search (A, v)

```
1  $i = 1$
2  while $i \leq A.length$ and $A[i] \neq v$
3      $i = i + 1$
4  if $i \leq A.length$
5      return $i$
6  else
7      return nil
```

— Initialisation : $i = 1$

  $[1\,..\,i-1]$ eſt vide, donc ne contient pas v.

— Conservation

  Pour tout $1 \leq i \leq \mathrm{A}.\mathit{length}$, pour la boucle i, on a $[1\,..\,i-1]$ ne contenant pas v. Le corps de la boucle eſt executé si et seulement si $\mathrm{A}[i] \neq v$ et que $i \leq \mathrm{A}.\mathit{length}$. Dans ce cas, avant l'itération $i+1$, le sous-tableau $[1\,..\,i]$ ne contient pas v.

— Terminaison

  Il y a deux cas de terminaison :

  - quand $i = \mathrm{A}.\mathit{length}+1$, l'algorithme retourne NIL et l'invariant de boucle confirme (en subtituant i par $\mathrm{A}.\mathit{length}+1$) que le tableau $[1\,..\,\mathrm{A}.\mathit{length}]$ ne contient v ;
  - il exiſte un $1 \leq i \leq \mathrm{A}.\mathit{length}$ tel que $\mathrm{A}[i] = v$, l'invariant de boucle dit que $[1\,..\,i-1]$ ne contient pas v, ce qui eſt vraie. Dans ce cas l'algorithme retourne i.

■

2.1-4 *Consider the problem of adding two n-bit binary integers, ſtored in two n-element arrays* A *and* B*. The sum of the two integers should be ſtored in binary form in an* $(n+1)$*-element array* C*. State the problem formally and write pseudocode for adding the two integers.*

Solution :

Input : Deux nombres binaires a et b sous forme de veƈteur $\mathrm{A} = \langle a_1, \ldots, a_n\rangle$ et $\mathrm{B} = \langle b_1, \ldots, b_n\rangle$ tel que pour tout $i \in [\![1, n]\!], a_i, b_i \in \{0,1\}$. Avec l'indice $i = 1$ désignant le bit le plus significant.

Output : Le veƈteur $\mathrm{C} = \langle c_1, \ldots, c_{n+1}\rangle$ qui représente $c = a + b$ en binaire.

Algorithme 3. ADD-BINARY-INTEGER (A, B)

```
1  $carry = 0$
2  for $i = n$ downto 1
3      $tmp = A[i] + B[i] + carry$
4      $C[i+1] = tmp \mod 2$
5      $carry = tmp/2$
6  $C[i] = carry$
```

■

2.2 Analyzing algorithms

2.2-1 *Express the funƈtion* $n^3/1000 - 100n^2 - 100n + 3$ *in terms of* Θ*-notation.*

Solution : $\Theta(n^3)$ ■

2.2-2 *Consider sorting n numbers ſtored in array* A *by firſt finding the smalleſt element of* A *and exchanging it with the element in* A[1]. *Then find the second smalleſt element of* A, *and exchange it with* A[2]. *Continue in this manner for the firſt* $n-1$ *elements of* A. *Write pseudocode for this algorithm, which is known as* ***selection sort***. *What loop invariant does this algorithm maintain? Why does it need to run for only the firſt* $n-1$ *elements, rather than for all n elements? Give the beſt-case and worſt-case running times of selection sort in* Θ*-notation.*

Solution :

Algorithme 4. Selection-Sort (A)

```
for $i = 1$ to $n-1$
    $min = i$
    for $j = i+1$ to $n$
        if $A[min] > A[j]$
            $min = j$
    if $min \neq j$
        swap(A, $min$, j)
```

— Invariant de boucle :

Le tableau $[1 \mathinner{..} i-1]$ eſt trié avant la $i$ème itération et tous les éléments dans $[i \mathinner{..} n]$ sont supérieurs à ceux dans $[1 \mathinner{..} i-1]$.

— À la $n-1$ itération, le tableau $[1 \mathinner{..} n-2]$ eſt trié, il ne reſte plus qu'à comparer $A[n-1]$ et $A[n]$.

— Temps d'exécution :

- Cas optimal : A déjà trié, $\Theta(n^2)$;
- Cas le plus défavorable : A trié de façon décroissante, $\Theta(n^2)$.

■

2.2-3 *Consider linear search again (see Exercise 2.1-3). How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worſt case? What are the average-case and worſt-case running times of linear search in* Θ*-notation? Juſtify your answers.*

Solution :

— Cas moyen : $\frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2}$;

— Cas le plus défavorable : $n+1$.

Donc $\Theta(n)$ dans les deux cas. ■

2.2-4 *How can we modify almoſt any algorithm to have a good beſt-case running time?*

Solution : ■

2.3 Designing algorithms

2.3-1 *Using Figure 2.4 as a model, illuſtrate the operation of merge sort on the array* $A = \langle 3, 41, 52, 26, 38, 57,$

Solution : ■

2.3-2 *Rewrite the* MERGE *procedure so that it does not use sentinels, inſtead ſtopping once either array* L *or* R *has had all its elements copied back to* A *and then copying the remainder of the other array back into* A.

Solution :

Algorithme 5. MERGE (A, p, q, r)

```
1   $n_1 = q - p + 1$
2   $n_2 = r - q$
3   Let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays
4   for $i = 1$ to $n_1$
5       $L[i] = A[p + i - 1]$
6   for $j = 1$ to $n_2$
7       $R[j] = A[q + j]$
8   $i = 1$
9   $j = 1$
10  for $k = p$ to $r$
11      if $L[i] \leq R[j]$ and $i \leq n_1$
12          $A[k] = L[i]$
13          $i = i + 1$
14      else
15          $A[k] = R[j]$
16          $j = j + 1$
```

■

2.3-3 *Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence*

$$T(n) = \begin{cases} 2 & \text{if } n = 2, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = n \lg n$.

Solution : Soit $n = 2^p$, avec $p \in \mathbb{N}^*$.

— Initialisation : $T(2) = 2 \lg 2 = 2$ eſt vraie.

— Hérédité : Soit $k < p$ et supposons que $T(2^k) = 2^k \lg 2^k = 2^k k$.
Alors $T(2^{k+1}) = 2T(2^k) + 2^{k+1} = 2^{k+1}(k+1)$.

— Conclusion : Pour tout $p \in \mathbb{N}^*$, on a bien $T(2^p) = 2^p p$.

■

2.3-4 *We can express insertion sort as a recursive procedure as follows. In order to sort $A = [1\,..\,n]$, we recursively sort $A = [1\,..\,n-1]$ and then insert $A[n]$ into the sorted array $A[1\,..\,n-1]$ Write a recurrence for the running time of this recursive version of insertion sort.*

Solution :

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n-1) + \Theta(n) & \text{otherwise} \end{cases}$$

■

2.3-5 *Referring back to the searching problem (see Exercise 2.1-3), observe that if the sequence A is sorted, we can check the midpoint of the sequence againſt and eliminate half of the sequence from further consideration. The* ***binary search*** *algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive, for binary search. Argue that the worſt-case running time of binary search is $\Theta(\lg n)$.*

Solution :

Algorithme 6. Rec-Bin-Search (A, p, q, v)

```
1  $mid = \lfloor (p+q)/2 \rfloor$
2  if $A[mid] < v$
3       Rec-Bin-Search$(A, r+1, q, v)$
4  else if $A[mid] > v$
5       Rec-Bin-Search$(A, p, mid-1, v)$
6  else if $A[mid] == v$
7       return $mid$
8  else
9       return NIL
```

Calculons le temps d'exécution au cas le plus défavorable. On a :

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n/2) + \Theta(1) & \text{otherwise} \end{cases}.$$

Supposons que $n = 2^p$, alors par la méthode d'arbre récursive, on a un arbre dégénéré de hauteur p dans lequel chaque nœud eſt étiqueté par une conſtante c. Il suffit donc de calculer

$$\begin{aligned}\sum_{i=0}^{p} c &= (p+1)c \\ &= (\lg n + 1)c \\ &= \Theta(\lg n).\end{aligned}$$

■

2.3-6 *Observe that the **while** loop of lines 5–7 of the* INSERTION-SORT *procedure in Section 2.1 uses a linear search to scan (backward) through the sorted subarray* $A[1\,..\,j-1]$. *Can we use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time of insertion sort to* $\Theta(n \lg n)$ *?*

Solution : ■

2.3-7 ★ *Describe a* $\Theta(n \lg n)$*-time algorithm that, given a set* S *of* n *integers and another integer* x*, determines whether or not there exist two elements in* S *whose sum is exactly* x*.*

Solution : ■

2.4 Problems

2-1 ***Insertion sort on small arrays in merge sort***

Although merge sort runs in $\Theta(n \lg n)$ *worst-case time and insertion sort runs in* $\Theta(n^2)$ *worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to **coarsen** the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which* $n = k$ *sublists of length* k *are sorted using insertion sort and then merged using the standard merging mechanism, where* k *is a value to be determined.*

a. *Show that insertion sort can sort the* $n = k$ *sublists, each of length* k*, in* $\Theta(nk)$ *worst-case time.*

b. *Show how to merge the sublists in* $\Theta(n \lg(n/k))$ *worst-case time.*

c. *Given that the modified algorithm runs in* $\Theta(nk + n \lg(n/k))$ *worst-case time, what is the largest value of* k *as a function of* n *for which the modified algorithm has the same running time as standard merge sort, in terms of* Θ*-notation ?*

d. *How should we choose* k *in practice ?*

Solution : ■

2-2 ***Correctness of bubblesort***

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

Solution : ■

2-3 ***Correctness of Horner's rule***

The following code fragment implements Horner's rule for evaluating a polynomial

Solution : ■

2-4 ***Inversions***

Solution : ■

3 Growth of Functions

3.1 Asymptotic notation

3.2 Standard notations and common functions

4 Divide-and-Conquer

4.1 The maximum-subarray problem

4.1-1 *What does* FIND-MAXIMUM-SUBARRAY *return when all elements of A are negative?*

Solution : L'indice du plus grand élément du tableau. ■

4.1-2 *Write pseudocode for the brute-force method of solving the maximum-subarray problem. Your procedure should run in $\Theta(n^2)$ time.*

Solution :

Algorithme 7. BRUTE-FIND-MAX-SUBARRAY (A)

```
index = period = −1
sum = −∞
for i = 1 to A.length
    tmp-sum = 0
    for tmp-period = 1 to A.length − i + 1
        tmp-sum = tmp-sum + A[i + tmp-period − 1]
        if tmp-sum > sum
            index = i
            period = tmp-period
            sum = tmp-sum
return (index, period, sum)
```

■

4.2 Strassen's algorithm for matrix multiplication

4.3 The substitution method for solving recurrences

A Summation

Références : [1]

A.1 Summation formulas and propreties

A.1-1 *Find a simple formula for* $\sum_{k=1}^{n}(2k-1)$

Solution :

$$\begin{aligned}\sum_{k=1}^{n}(2k-1) &= n(n+1)-n \\ &= n^2.\end{aligned}$$

■

A.1-2 ⋆ *Show that* $\sum_{k=1}^{n} 1/(2k-1) = \ln(\sqrt{n}) + \mathrm{O}(1)$ *by manipulating the harmonic series.*

Solution :

$$\begin{aligned}\sum_{k=1}^{n}\frac{1}{2k-1} &= \mathrm{H}_n - \frac{1}{2}\sum_{k=1}^{n}\frac{1}{k} \\ &= \frac{1}{2}\mathrm{H}_n \\ &= \ln(\sqrt{n}) + \mathrm{O}(1).\end{aligned}$$

■

A.1-3 *Show that* $\sum_{k=0}^{\infty} k^2x^k = x(1+x)/(1-x)^3$ *for* $0 < |x| < 1$.

Solution :

$$\begin{aligned}\sum_{k=0}^{\infty} k^2x^k &= x\left(\frac{x}{(1-x)^2}\right)' \\ &= \frac{x}{(1-x)^2} + \frac{2x}{(1-x)^3} \\ &= \frac{x(1+x)}{(1-x)^3}.\end{aligned}$$

■

A.1-4 ⋆ *Show that* $\sum_{k=0}^{\infty}(k-1)/2^k = 0$.

Solution : Soit $S = \sum_{k=0}^{\infty} \frac{k-1}{2^k}$. Alors $2S = \sum_{k=0}^{\infty} \frac{k-1}{2^{k-1}} = -2 + \sum_{k=0}^{\infty} \frac{k}{2^k}$. Donc

$$\begin{aligned} 2S - S &= -2 + \sum_{k=0}^{\infty} \frac{1}{2^k} \\ &= 0. \end{aligned}$$

■

A.1-5 ★ *Evaluate the sum* $\sum_{k=1}^{\infty}(2k+1)x^{2k}$ *for* $|x| < 1$.

Solution :

$$\begin{aligned} \sum_{k=1}^{\infty}(2k+1)x^{2k} &= \left(\sum_{k=1}^{\infty} x^{2k+1}\right)' \\ &= \left(\frac{x}{1-x^2} - x\right)' \\ &= \frac{3x^2}{1-x^2} + \frac{2x^4}{(1-x^2)^2} \end{aligned}$$

■

A.1-6 *Prove that* $\sum_{k=1}^{n} O(f_k(i)) = O\left(\sum_{k=1}^{n} f_k(i)\right)$ *by using the linearity proprety of summations.*

Solution : Soit pour tout $k \in [\![1,n]\!], g_k \in O(f_k)$, autrement dit $g_k \leq c_k f_k$ est vérifié à partir d'un rang N_k avec $c_k > 0$. On a donc $\sum_{k=1}^{n} g_k \leq \sum_{k=1}^{n} c f_k$ vérifié à partir d'un rang $N = \max(N_1, \ldots, N_n)$ et $c = \max(c_1, \ldots, c_n)$. D'où $\sum_{k=1}^{n} g_k \in O(\sum_{k=1}^{n} f_k)$. ■

A.1-7 *Evaluate the product* $\prod_{k=1}^{n} 2 \cdot 4^k$.

Solution : Soit $P = \prod_{k=1}^{n} 2 \cdot 4^k = \prod_{k=1}^{n} 2^{2k+1}$. On a

$$\begin{aligned} \lg P &= \sum_{k=1}^{n} 2k + 1 \\ &= n(n+2) \end{aligned}$$

Finalement, $P = 2^{n(n+2)}$. ■

A.1-8 ★ *Evaluate the product* $\prod_{k=2}^{n}(1 - 1/k^2)$.

Solution :

$$\begin{aligned}\prod_{k=2}^{n}(1-\frac{1}{k^2}) &= \prod_{k=2}^{n}\frac{k-1}{k}\cdot\frac{k+1}{k}\\ &= \frac{1}{2}\cdot\frac{n+1}{n}\\ &= \frac{n+1}{2n}.\end{aligned}$$

■

A.2 Bounding summations

A.2-1 *Show that $\sum_{k=1}^{n} 1/k^2$ is bounded above by a constant.*

Solution :

$$\begin{aligned}\sum_{k=1}^{n}\frac{1}{k^2} &= 1+\sum_{k=2}^{n}\frac{1}{k^2}\\ &\le 1+\int_1^n \frac{1}{x^2}\,\mathrm{d}x\\ &= 1+(1-\frac{1}{n})\\ &= 2-\frac{1}{n}\\ &\le 2.\end{aligned}$$

■

A.2-2 *Find an asymptotic upper bound on the summation* $\sum\limits_{k=0}^{\lfloor \lg n\rfloor}\lceil n/2^k\rceil$.

A.2-3 *Show that the nth harmonic number is $\Omega(\lg n)$ by splitting the summation.*

A.2-4 *Approximate $\sum_{k=1}^{n} k^3$ with an integral.*

Solution : On a

$$\int_0^n x^3\,\mathrm{d}x \le \sum_{k=1}^{n} k^3 \le \int_1^{n+1} x^3\,\mathrm{d}x$$

ce qui donne

$$\frac{n^4}{4} \le \sum_{k=1}^{n} k^3 \le \frac{(n+1)^4-1}{4}.$$

Ainsi, $\sum_{k=1}^{n} k^3 = \Theta(n^4)$. ■

A.2-5 *Why didn't we use the integral approximation (A.12) directly on $\sum_{k=1}^{n} 1/k$ to obtain an upper bound on the nth harmonic number ?*

Solution : Car la primitive de $1/x$ n'eſt pas définie en 0. ■

A.3 Problems

A-1 *Bounding summations*

Give asymptotically tight bounds on the following summations. Assume that $r > 0$ and $s > 0$ are conſtants.

a. $\sum_{k=1}^{n} k^r$.

b. $\sum_{k=1}^{n} \lg^s k$.

c. $\sum_{k=1}^{n} k^r \lg^s k$

Solution : ■

Références

[1] Jānis Lazovskis. University of Illinois at Chicago – MCS 401 Homework 1 grader solutions.