intra-A2013

English translation

No documentation is allowed.

The examen covers questions E0–E5 (or F0–F5) and is worth 100 points. You may write your answers in English or in French.

Answer each question in the exam booklet.

E0 Your name (1 point)

► Write your name and *code permanent* on each booklet that you submit.

E1 Exponential growth (30 points)

This exercise explores different definitions of exponential growth for a function $f\colon \mathbb{N} \mapsto [0, \infty)$. Define the following terms :

⋆ **pure function** : $f(n) = \Theta(a^n)$ with some constant $a > 1$;

⋆ **mixed function** : $f(n) = 2^{\Theta(n)}$;

⋆ **fast-growing function** : $f(n) = 2^{2^{\Theta(\lg n)}}$.

(Clearly, every pure function is also mixed, and every mixed function is fast-growing.)

a. (10 points) ► Show that not all fast-growing functions are mixed. (Construct an example.)

b. (10 points) ► Show that not all mixed functions are pure. (Give an example.)

c. (10 points) ► Show that f is mixed if and only if $f(n) = \big(\Theta(1)\big)^n$.

E2 ADT interface (15 points)

► Specify the two fundamental operations (insertion and removal) in the abstract data types of stack, FIFO queue, and priority queue. Explain how they change the set of stored elements.

E3 d-ary tree (14 points)

Consider a d-ary tree with n internal and m external nodes for some $d > 1$. (In a d-ary tree, every internal node has exactly d children.) ► Prove that $m - 1 = (d - 1)n$.

Hint: *Count the number of nodes in two ways : once by the number of children, and once by their parents.*

E4 Conquerors (20 points)

Let T be a binary tree where every internal node x has an integer key stored as $x.\mathsf{key}$. In this exercise, an internal node x is a **conqueror** if and only if $x.\mathsf{key} \leq y.\mathsf{key}$ for all internal nodes in the subtree rooted at x. (An internal node with two external children is a conqueror by default.)

a. (5 points) ▶ Argue that if the tree is in min-heap order, then every internal node is a conqueror.

b. (10 points) ▶ Give an algorithm that counts the number of conqueror nodes in a binary tree.

c. (5 points) ▶ Analyze your algorithm's running time (it should be $O(n)$ on a tree with n internal nodes).

E5 Prebuilt minimum spanning tree (20 points)

Let $G = (V, E)$ be a graph with positive edge weights $w\colon E \mapsto (0, \infty)$. Suppose that you are given a set of **fixed edges** $E' \subseteq E$ forming no cycles between themselves. A spanning tree $T = (V, E_T)$ **guided by** E' covers all vertices and includes the fixed edges : $E' \subseteq E_T \subseteq E$.

▶ Give an algorithm that finds a spanning tree $T = (V, E_T)$ with minimum weight $\sum_{e \in E_T} w(e)$, guided by a given edge set E' (input as an array of vertex pairs). Analyze your algorithm's running time.

GOOD LUCK !

intra-A2012

English translation

No documentation is allowed. The examen is worth 100 points, and you can collect up to 15 additional bonus points. You may write your answers in English or in French.

Answer each question in the exam booklet.

E0 Your name (1 point)

▶ Write your name and *code permanent* on each booklet that you submit.

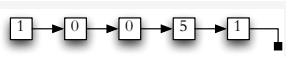
E1 Growth rates (20 points)

▶ Fill out the following table : every answer is worth 2 points. For each pair f, g, write "=" if $\Theta(f) = \Theta(g)$, "$\ll$" if $f = o(g)$, "$\gg$" if $g = o(f)$, and "???" if neither of the three applies. You do not need to justify the answers. $\lg n$ denotes the binary logarithm of n.

	$f(n)$	$g(n)$
a	$f(n) = n^2$	$g(n) = (2015n + 1)^2$
b	$f(n) = \lg n$	$g(n) = \lg(n^{2015})$
c	$f(n) = \sum_{i=0}^{n} 2^i$	$g(n) = 2^n$
d	$f(n) = \sum_{i=1}^{n} 1/i$	$g(n) = \ln n$
e	$f(n) = n!$	$g(n) = (2n)!$
f	$f(n) = n^{2015}$	$g(n) = (2n)^{2015}$
g	$f(n) = n$	$g(n) = \begin{cases} 1 & \{n = 0, 2, 4, 6, 8, \dots\} \\ n & \{n = 1, 3, 5, 7, 9, \dots\} \end{cases}$
h	$f(n) = n \ln n$	$g(n) = \lg(n!)$
i	$f(n) = 4^n$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ 4^{g(n-1)} & \{n > 0\} \end{cases}$
j	$f(n) = 2^{2^n}$	$g(n) = 4^n$

E2 Polynomials (30 points)

We want to use a linked list to represent polynomials over a single variable. For a polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_dx^d$, the list has $(d+1)$ nodes. The first node stores coefficient a_0, the second a_1, and so on.



List representing x^4+5x^3+1, a polynomial of degree 4.

a. Degree (15 points). The degree of the polynomial is the largest power of the variable with non-zero coefficient ($\max\{i\colon a_i > 0\}$), or 0 if $a_0 = 0$. ▶ Give an algorithm $\mathsf{degree}(N)$ that computes the degree of a polynomial stored in a list starting with node N (containing a_0) Note that any a_i, including a_d may be 0. The empty list (null) corresponds to $f(x) = 0$.

b. Evaluation (15 points). One uses Horner's rule to evaluate a polynomial $f(x)$ at a given value x. Write

$$f(x) = a_0+a_1x+a_2x^2+\cdots+a_dx^d = \underbrace{\Big(\Big(\big((a_d)x + a_{d-1}\big)x + a_{d-2}\Big)x + \cdots + a_0\Big)}_{(d+1)\text{ nested parentheses}}$$

For instance, $2x^3 + x^2 - x + 7 = ((2x+1)x - 1)x + 7$. In general, the evaluation is done using d additions and d multiplications : this is faster than a naïve evaluation in which one computes $x^1, x^2, \ldots, x^d$ separately.

▶ Give a *recursive* algorithm $\mathsf{eval}(N, x)$ that evaluates a polynomial (constant term stored at the list head N) by Horner's rule.

♡ **c. Addition of sparse polynomial (15 bonus points)** For a *sparse* polynomial with many 0 coefficients (such as $f(x) = x^{88} + 1$), it is better to keep only the non-zero terms. One can use thus a linked list where the nodes store pairs (a_i, i) of non-zero coefficients a_i and powers i. (The nodes are kept in a strict increasing order of powers i.)

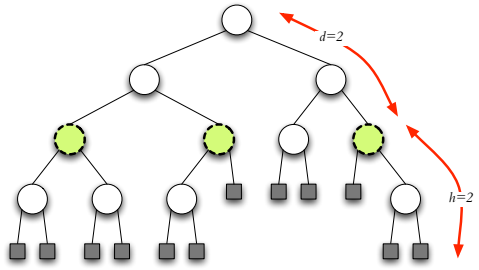
▶ Give an algorithm that computes $f(x) + g(x)$ symbolically for two polynomials f, g. The algorithm takes the two list heads as arguments, and returns the head for the list storing the result. For instance, if $f(x) = x^{88} + 1$ (2 nodes) and $g(x) = 6x^2 - x - 1$ (3 nodes), the result $f(x) + g(x) = x^{88} + 6x^2 - x$ consists of 3 nodes $(-1, 1)$, $(6, 2)$, $(1, 88)$.

E3 $\Theta \leftrightarrow o$ (14 points)

Let $f, g\colon \{0, 1, 2 \dots\} \mapsto (0, \infty)$ be two positive functions with $f(n) = \Theta\Big(2^{g(n)}\Big)$.

► Show that if $\lim_{n\to\infty} g(n) = \infty$, then $\lg f(n) = \big(1 + o(1)\big)g(n)$.

E4 Median of a tree (20 points)



In this exercise, a node is called *median*, if its height equals its depth.

► Give a *recursive* algorithm that lists all the median nodes in a binary tree. The algorithm must take linear time in the tree size.

Hint: *Pass the depth as argument, and return the height.*

E5 Deletion in a binary heap (15 points)

► Give an algorithm delete(H, i, n) that deletes an element in a binary heap $H[1..n]$ specified by its index i in the table. (The argument n denotes the number of elements in the heap : the table might have a larger capacicity than that.) The algorithm may rearrange the elements in the table, but $H[1..n-1]$ must contain the remaining elements after the deletion. Show all the details of the algorithm.

Hint: *For $i = 1$, the execution is identical to* deleteMin *; otherwise, inspect the parent and the children of cell i to decide between sink and swim.*

GOOD LUCK !

intra-H2012

English translation

No documentation is allowed. The examen is worth 100 points, and you can collect up to 15 additional bonus points for exercises marked by ♡. You may write your answers in English or in French.

Answer each question in the exam booklet.

E0 Your name (1 point)

▶ Write your name and *code permanent* on each booklet that you submit.

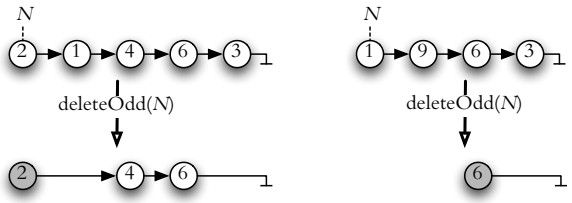
E1 Growth rates (20 points)

▶ Fill out the following table : every answer is worth 2 points. For each pair f, g, write "=" if $\Theta(f) = \Theta(g)$, "$\ll$" if $f = o(g)$, "$\gg$" if $g = o(f)$, and "???" if neither of the three applies. You do not need to justify the answers. $\lg n$ denotes the binary logarithm of n.

	$f(n)$	$g(n)$
a	$f(n) = 2n^2$	$g(n) = n^2 \lg n$
b	$f(n) = \sqrt{n}$	$g(n) = \sqrt[3]{n}$
c	$f(n) = \sum_{i=0}^{n} 2^i$	$g(n) = 2^n$
d	$f(n) = \sum_{i=1}^{n} 1/i$	$g(n) = \log_{2015} n$
e	$f(n) = n!$	$g(n) = (2n)!$
f	$f(n) = n^{2015}$	$g(n) = (2n)^{2015}$
g	$f(n) = n$	$g(n) = \begin{cases} n+2 & \{n \le 2\} \\ \frac{n \lg n}{\lg \lg n} & \{n > 2\} \end{cases}$
h	$f(n) = n \lg n$	$g(n) = \ln(n!)$
i	$f(n) = n \lg n$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ 2g(\lfloor n/2 \rfloor) + \Theta(1) & \{n > 0\} \end{cases}$
j	$f(n) = 2^{2^n}$	$g(n) = 4^n$

E2 Even-odd (20+3 points)

We have a linked list where every node N is equipped with the variables N.key (an integer key) and N.next (next node). We would like to have a procedure called deleteOdd(N) that deletes the nodes with odd keys on the list starting with N, and returns the new head of the list. The algorithm must keep the original order of the nodes with even keys. Show all the details (e.g., it is not enough to say "deletion after x :" you need to show the exact instructions).



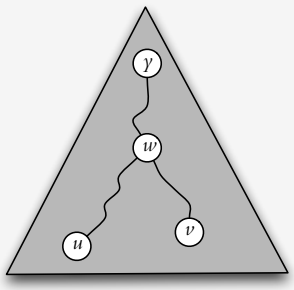
Examples of deleteOdd *: the shaded nodes indicate the return value.*

a. Iteration (10 points) ► Give an *iterative* implementation of deleteOdd.

b. Recursion (10 points) ► Give a *recursive* implementation of deleteOdd.

♡**c. Terminal recursion (3 bonus points)** You can have up to 3 bonus points for terminal recursion in **b**.

E3 Common ancestor (24 points)

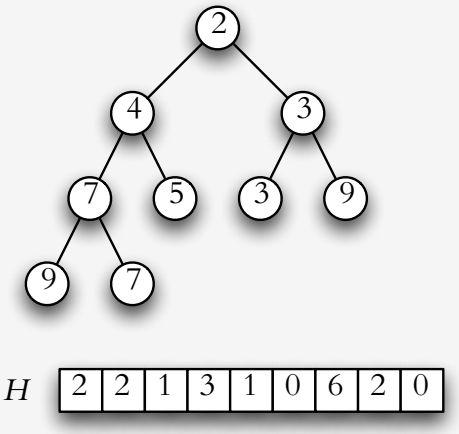


Let u and v be two nodes in a tree. A node y is their *common ancestor* if and only if u et v belong to the subtree rooted at y. The *lowest common ancestor* (LCA) is the ancestor w such that its subtree contains no other ancestor.

► Give an algorithm lca(u, v) that returns the LCA of two internal nodes u, v in a binary tree. Analyze your algorithm's running time.

Hint. Collect the ancestors of u and v in a convenient data structure, and identify the common ones.

E4 Differential heap (35 points)



We would like to implement a priority queue by using a binary heap where, with the exception of the minimal element $H[1]$, the priorities are not stored directly. Instead, we store the difference between the priorities of parent and child. At $H[1]$, we store the true priority. (The priority queue interface is not changed. This kind of implementation is useful for ajusting all priorities by the same amount in $O(1)$ — it is enough to change the priority at the root.)

a. The true priorities (10 points). ► Give an algorithm getPriority(i) that computes the true priority of an element stored at index i. $(1 \leq i \leq n)$

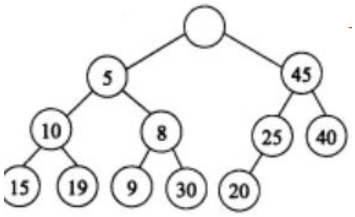
b. Insertion (25 points). ► Give an algorithm for inserting an element in a differential heap, given its true priority.

Hint. Here, you need to adapt the classic swim procedure. First, calculate the difference between parent and child at the point of insertion (last entry in the heap's array) — it may be negative, indicating that one should move up towards the root. Examine how differences between priorities should change in one iteration of swim.

E5 ♡ Difficult to compare. . . (12 bonus points)

In the usual implementaion of the binary heap, one insertion uses $O(\log n)$ comparisons between keys, and $O(\log n)$ assignments (to a table cell). Show how to do the insertion with $O(\log \log n)$ comparisons. (Such a solution may be useful when comparison is much more expensive than assignment — e.g., as with strings.)

I found an excellent idea: DEAP, which is a MIN-MAX heap, the left sub-tree is a min-heap, while the right one is a max-heap, and thus, if the value inserted is between the value of root and the last one in sub-right-tree, do it in the right sub-tree, otherwise move to the left-sub-tree.



intra-A2011

English translation

No documentation is allowed. The examen is worth 100 points, and you can collect up to 15 additional bonus points for exercises marked by ♡. You may write your answers in English or in French.

Answer each question in the exam booklet.

E0 Your name (1 point)

▶ Write your name and *code permanent* on each booklet that you submit.

E1 Growth rates (20 points)

▶ Fill out the following table : every answer is worth 2 points. For each pair f, g, write "=" if $\Theta(f) = \Theta(g)$, "$\ll$" if $f = o(g)$, "$\gg$" if $g = o(f)$, and "???" if neither of the three applies. You do not need to justify the answers. $\lg n$ denotes the binary logarithm of n.

	$f(n)$	$g(n)$
a	$f(n) = 2015n$	$g(n) = n^2/2015$
b	$f(n) = \sqrt{n}$	$g(n) = \sqrt[3]{n}$
c	$f(n) = \sum_{i=0}^{n} 3^i$	$g(n) = 3^n$
d	$f(n) = \sum_{i=1}^{n} n/i$	$g(n) = n \lg n$
e	$f(n) = n!$	$g(n) = (n+1)!$
f	$f(n) = 2.015^n$	$g(n) = 2.015^{n+1}$
g	$f(n) = 2.015^n$	$g(n) = 2.015^{2n}$
h	$f(n) = n \lg n$	$g(n) = \lg(n!)$
i	$f(n) = \max\{1, n + 2015 \sin n\}$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ g(n-1) + \Theta(1) & \{n > 0\} \end{cases}$
j	$f(n) = n!$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ 2ng(n-1) & \{n > 0\} \end{cases}$

a.<<
b.>>
c.=
d.=
e.<<
f.=
g.<<
h.=
i.=
j.<<

E2 *Parenthèses* (20 points)

The formal language $\mathsf{Dyck}(s)$ is the one of well-formed expressions with s types of parentheses : $\underset{\smile}{j}, \overset{\frown}{j}$ for $j = 1, 2 \ldots, s$ (like the XML language). A valid expression is derived by the rules

$$
\begin{array}{ll}
E \mapsto \varepsilon & \textit{ε is the empty string} \\
E \mapsto E\,E & \textit{more than one term side by side} \\
E \mapsto \underset{\smile}{1}\,E\,\overset{\frown}{1} & \textit{parentheses of type 1} \\
\ldots & \\
E \mapsto \underset{\smile}{s}\,E\,\overset{\frown}{s} & \textit{parentheses of type s}
\end{array}
$$

So, $\underset{\smile}{1}\,\underset{\smile}{2}\,\overset{\frown}{2}\,\underset{\smile}{3}\,\overset{\frown}{3}\,\overset{\frown}{1}\,\underset{\smile}{1}\,\overset{\frown}{1}$ is valid, but $\underset{\smile}{1}\,\underset{\smile}{2}\,\overset{\frown}{2}\,\underset{\smile}{3}\,\overset{\frown}{1}\,\overset{\frown}{3}$ is not. An *encoded expression* is an array $x[0..n-1]$ with elements $x[i] \in \{\pm 1, \pm 2, \ldots, \pm s\}$, where $+j$ encodes opening parenthesis $\underset{\smile}{j}$, and $-j$ encodes closing parenthesis $\overset{\frown}{j}$ for $j = 1, \ldots, s$.

► Give an algorithm to verify if an encoded expression is valid, in $O(n)$ time. Justify why your algorithm is correct.

E3 Queue with just a few priorities (20+5 points)

key-value，insert时二分查找？

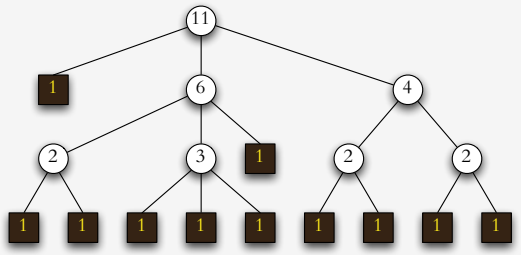
Suppose that the priorities have only a few possible values. It is then possible to design a better data structure for a priority queue than the binary heap.

► Propose a data structure implementing a priority queue for a known set of possible priorities $\{0, \ldots, m-1\}$. Show how the usual operations of the interface are implemented, and the initialization of the structure with m passed as an argument. The data structure must support $\mathsf{deleteMin}()$ and $\mathsf{insert}(x)$ in $O(m)$ worst-case time.

♡[**5 points boni**] Give a solution performing $\mathsf{deleteMin}$ and insert in $O(\log m)$ time.

In your algorithm, use the prefined function $\phi(x)$ to get the priority of x (the parameter of insert). Show only as much details as necessary. You do not need to show how common abstract data type interfaces are implemented. Rely on basic techniques on data structures discussed at the course (e.g., deleting a node on a linked list or swim/sink in a binary heap), but indicate clearly how you use them.

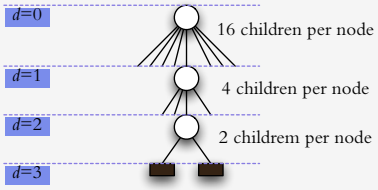
E4 Node counting (19 points)



In a rooted tree T, every internal node x stores its children in the variable x.children. Let $n[x]$ be the number of external nodes in the subtree rooted at x (including x if it is external).

a. Calculating size (10 points)

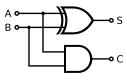
- ▶ Give a recursive definition for $n[x]$.
- ▶ Give a recursive algorithm for determining $n[x]$ for a node x (internal or external) passed as argument. The algorithm must take linear time (in the number of external nodes $n[x]$).

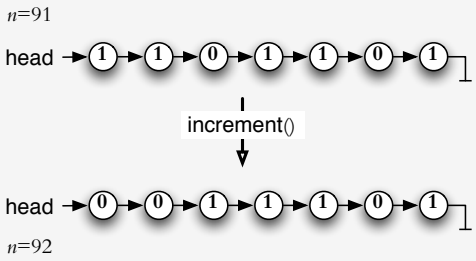


A *Dutch tree* of height h is an ordered tree where the internal nodes at each level $d = 0, 1, 2, \ldots, h-1$ have exactly $2^{2^{h-d-1}}$ children each. The external nodes are at level h.

b. Dutch tree (9 points) ▶ Show that the root of a Dutch tree with n external nodes has $\Theta(\sqrt{n})$ children.

E5 Binary arithmetics (20+10 points)





Suppose that one uses a linked list to represent arbitrarily large non-negative integers. Each list node x is equipped thus with x.next for next node (= null denotes the end), and x.bit for storing one bit in the binary representation (taking the value 0 or 1).

The list head (head variable) gives the least significant bit. The number 0 is represented by a list with a single node x where $x.\mathsf{bit} = 0$. A positive number $n > 0$ with $2^{k-1} \leq n < 2^k$ is represented by a list of length k : $n = \sum_{i=0}^{k-1}(x_i.\mathsf{bit}) \cdot 2^i$ where x_i is the i-th node after the head (head is x_0).

- ▶ Give an algorithm implementing the increment() operation which increments (by one) the number represented by the list. (The algorithm should not create a different list for the result, but rather update the list itself.) Analyze the asymptotic growth for the worst-case running time in function of the icremented value n. Would you say that our algorithm takes a *linear* time ? Justify your response.
- ♡[**5 points boni**] Show that increment takes $O(1)$ amortized time in your implementation.
- ♡[**5 points boni**] Implement increment for *Fibonacci encoding*. In that encoding, a liste with k nodes represents the number $n = \sum_{i=0}^{k-1}(x_i.\mathsf{bit}) \cdot F(i+2)$, where $F(i)$ is the i-th Fibonacci number. (Reminder : $F(0) = 0, F(1) = 1$.) Note that the Fibonacci representation is not unique : for example, $19 = \overline{101001} = \overline{11111}$ since $19 = 13 + 5 + 1 = 8 + 5 + 3 + 2 + 1$. **Hint.** The key is to use the minimum-weight encoding which is the one minimizing $\sum_i x_i.\mathsf{bit}$. For instance, one can always replace the bit sequence $\overline{011}$ by $\overline{100}$ because $F(i) = F(i-2) + F(i-1)$.

intra-H2011

English translation

No documentation is allowed. The examen is worth 100 points, and you can collect an additonal 10 bonus points.

Answer each question in the exam booklet.

E0 Your name (1 point)

► Write your name and *code permanent* on each booklet that you submit.

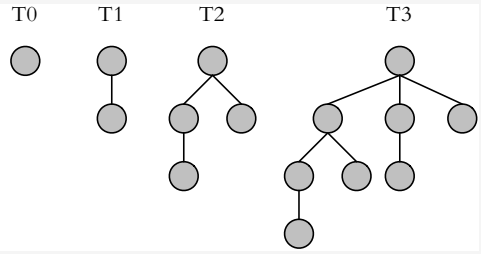
E1 Abstract types (15 points)

Cuckoo's Egg (3 points) There are three abstract data types in the following list : ► identify which ones.

> stack, linked list, ternary heap, queue, priority queue, binary tree, Justin Bieber.

Interface (12 points) The three identified abstract data types (ADTs) are generalized queues with elementary operations for insertion and removal of elements. ► Specify the two fundamental operations for each ADT, and explain how they change the set of elements in the queue.

E2 Binomial tree (20+10 points)



The **binomial trees** $T_0, T_1, T_2, \ldots$ are defined by recursion :

- ⋆ T_0 contains a single node (its root).
- ⋆ T_k contains a node, called its root, and the children $T_{k-1}, T_{k-2}, \ldots, T_0$, in this order from left to right.

a. (10 points) ► Show formally that T_k has exactly k nodes.

b. (10 points) ► Show formally that there are exactly $\binom{k}{d}$ nodes at depth d in T_k, where $\binom{k}{d} = \frac{k!}{d!(k-d)!}$ denotes the binomial coeficient (hence the name).

Tips: C(k+1)(d+1)=C(k)(d)+C(d+1)(k)=\sum_{j=d}^{k}C(d)(j)

c. (10 bonus points) Suppose that we have two trees T_k in which elements are stored at the nodes in heap order. ► Show how to merge the two trees into T_{k+1}, while preserving the heap order, in $O(k)$ time.

Hint : use induction. In **b.**, use the identities $\binom{k+1}{d+1} = \binom{k}{d} + \binom{k}{d+1} = \sum_{j=d}^{k} \binom{k}{j}$.

E3 Growth rates (20 points)

▶ Fill out the following table : every answer is worth 2 points. For each pair f, g, write "=" if $\Theta(f) = \Theta(g)$, "$\ll$" if $f = o(g)$, "$\gg$" if $g = o(f)$, and "???" if neither of the three applies. You do not need to justify the answers. $\lg n$ denotes the binary logarithm of n.

	$f(n)$	$g(n)$
a	$f(n) = 4n^2 \lg n$	$g(n) = 1 + 3n + 3n^2 + n^3$
b	$f(n) = n^{2015}$	$g(n) = n!$
c	$f(n) = \sum_{i=0}^{n} 2^i$	$g(n) = 2^n$
d	$f(n) = \sum_{i=1}^{n} 1/i$	$g(n) = \lg n$
e	$f(n) = 2.015^n$	$g(n) = 2^n$
f	$f(n) = 2.015^{\lg n}$	$g(n) = 2^{\lg n}$
g	$f(n) = n \lg n$	$g(n) = \lg(n!)$
h	$f(n) = n \lg n$	$g(n) = \begin{cases} 1 & \{n < 2\} \\ g(\lceil n/2 \rceil) + g(\lfloor n/2 \rfloor) + O(1) & \{n \geq 2\} \end{cases}$
i	$f(n) = \log_3(\log_4 n)$	$g(n) = \log_4(\log_3 n)$
j	$f(n) = \sqrt{n}$	$g(n) = \begin{cases} 1 & \{n < 2\} \\ g(n-2) + O(1) & \{n \geq 2\} \end{cases}$

E4 Q2D (29 points)

Propose a data structure for storing a set of points (in 2D), implementing the operations add, deleteMinX and deleteMinY.

The add(x, y) operation adds the point (x, y). deleteMinX deletes and returns the point with minimal X coordinate. deleteMinY deletes and returns the point with minimal Y coordinate. For instance, after add$(3, 8)$, add$(1, 12)$, add$(5, 2)$, the operations deleteMinX(), deleteMinY() return $(1, 12)$ et $(5, 2)$. Each operation must take $O(\log n)$ time at worst when the queue has n points. ▶ Show the details of the principal algorithms in the implementation. It is enough to show the details for just one of two symmetrical procedures (like deleteMinX − deleteMinY).

Hint. Use two min-heaps, and store references to indices between them. For instance, a point can be stored at the index i in heap H with fields $H[i]$.x, $H[i]$.y and $H[i]$.indice, where the last one gives the index to the same point's position in the other heap. During swim and sink, one needs to maintain the correct indexes in the other heap $H' : j \leftarrow H[i]$.indice; $H'[j]$.indice $\leftarrow i$ at every index i visited in heap H. A deleteMin in one of the heaps must be coupled with deleting the element at index $j = H[1]$.indice on the other heap : generalize the technique of deleteMin for an arbitrary index j. In order to simplify the discussion, you can assume that

the allocated tables are large enough : just make sure that $H[i].\mathsf{x} = H[i].\mathsf{y} = \infty$ for $i > n$.

E5 Half-balanced tree (15 points)

Let $h(x)$ be the height of node x in a binary tree :

$$h(x) = \begin{cases} 0 & \text{if } x \text{ is external}\,; \\ 1 + \max_{y \in \text{children of } x} h(y) & \text{if } x \text{ is internal} \end{cases}.$$

Similarly, define

$$r(x) = \begin{cases} 0 & \text{if } x \text{ is external}\,; \\ 1 + \min_{y \in \text{children of } x} r(y) & \text{if } x \text{ is internal} \end{cases}.$$

For an external node, $h(x) = r(x) = 0$, even if it is represented by null, so $h(\mathsf{null}) = r(\mathsf{null}) = 0$. A binary tree is **half-balanced** if $h(x) \leq 2r(x)$ at every node x. ▶ Give an algorithm that checks if a binary tree is half-balanced. (Note that the algorithm has to compute h and r.) The algorithm must run in $O(n)$ on a tree with n nodes. **Hint :** do a tree traversal and compute $h(x)$ and $r(x)$ in parallel.

GODSPEED !

intra-A2010

English translation

No documentation is allowed. The examen is worth 100 points.
Answer each question in the exam booklet.

0 Your name (1 point)

► Write your name and *code permanent* on each booklet that you submit.

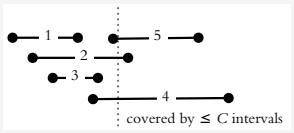
1 Everything is linear (10 points)

Consider the recurrence $T(n) \leq T(\lceil n/2 \rceil) + T(\lfloor n/3 \rfloor) + O(n)$, holding for all $n > 1$. ► Show that $T(n) = O(n)$.

2 Deque (20 points)

The abstract data type *deque* (double-ended queue) combines the stack and the queue, with operations push, pop, enqueue, dequeue. ► Show how to implement a deque by using a single table. Every operation must be done in $O(1)$. (You can ignore the dynamic table expansion/reduction in your implementation, but explain in a few words how your structure would accommodate it.)

3 Sorting intervals (25 points)



Consider a sequence of intervals $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ (stored in a table $T[1..n]$) with $x_i < y_i$, sorted by their left-hand sides :$x_1 < x_2 < x_3 < \cdots < x_n$. The intervals may have different lengths. We know that at every point $z \in \mathbb{R}$, there are at most C overlapping intervals. In other words, there are at most C intervals (x_i, y_i) with $x_i < z < y_i$.

► Give an algorithm that sorts the intervals by their right-hand side. The algorithm has to sort $T[]$ by using at most $O(C)$ additional work space, in $O(n \log C)$ time.

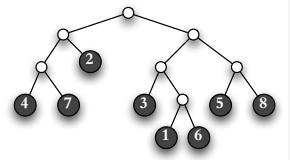
4 Growth rates (20 points)

► Fill out the following table : every answer is worth 2 points. For each pair f, g, write "=" if $\Theta(f) = \Theta(g)$, "$\ll$" if $f = o(g)$, "$\gg$" if $g = o(f)$, and "???"

if neither of the three applies. You do not need to justify the answers. $\lg n$ denotes the binary logarithm of n.

	$f(n)$	$g(n)$
a	$f(n) = \sqrt{n}$	$g(n) = n^{2/3}$
b	$f(n) = 2n^2 - \log_4 n$	$g(n) = 4^{\lg n}$
c	$f(n) = 2^{2^n}$	$g(n) = 4^n$
d	$f(n) = n!$	$g(n) = n^n$
e	$f(n) = 1.12^n$	$g(n) = 1.13^n$
f	$f(n) = \lg^* n$	$g(n) = \lg \lg n$
g	$f(n) = n \lg n$	$g(n) = \lg(n!)$
h	$f(n) = \lg n$	$g(n) = \begin{cases} 1 & \{n < 3\} \\ f(\lceil n/3 \rceil) + O(1) & \{n \geq 3\} \end{cases}$
i	$f(n) = \log_{\lg n} n$	$g(n) = \frac{\lg n}{\lg \lg n}$
j	$f(n) = \lg n$	$g(n) = \begin{cases} 1 & \{n = 1\} \\ g(n-1) + 1/n & \{n > 1\} \end{cases}$

5 Labeled tree (24 points)



You are given a binary tree in which the leaves are bijectively labeled by $\{1, 2, \ldots, n\}$. (The other nodes have no labels.)

Tree implementation : for each node x, x.left is the left child, x.right is the right child, and x.etiquette gives the label of x [when it is a leaf].

a. (12 points) ▶ Give an algorithm MINETIQUETTE(x) that returns the minimal label in the subtree of node x given as argument.

b. (12 points) We prefer to have the children of each node ordered in the sense that MINETIQUETTE is less for the left child than for the right child. For this, one needs to traverse the tree and switch the left and right children whenever necessary. ▶ Give an algorithm that orders the children at every node in $O(n)$ overall time.

GODSPEED !

intra-H2010

English translation

No documentation is allowed. The examen is worth 100 points.
Answer each question in the exam booklet.

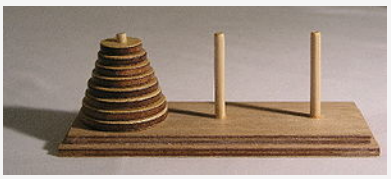
0 Your name (1 point)

Write your name and *code permanent* on each booklet that you submit.

1 Exchange (9 points)

Show the code for exchanging two elements on a linked list. The operation's argument is a reference to the node after which two successive elements are to be switched.

2 Towers of Hanoi (20 points)



The Towers of Hanoi puzzle consists of three pegs, and a set of disks of different diameters $(1, 2, \ldots, n)$. The objective is to move the stack of disks (initially in decreasing order) from one peg to another, obeying the following rules.

Rule 1. Only one disk may be moved at a time. A move consist of taking the upper disk from one peg and placing it on top of other disks (if any) on another peg.

Règle 2. A disk may be placed only on top of a smaller disk, or on an empty peg.

The solution uses the recursive procedure HANOI(i, j, k, n) that moves the top n disks from peg i to peg j using k as the auxiliary peg.

```
    Algo HANOI(i, j, k, n)
H1  if n ≠ 0
H2      HANOI(i, k, j, n − 1)
H3      move disk n from i to j
H4      HANOI(k, j, i, n − 1)
```

Let $T(n)$ be the running time of this algorithm. Show the recurrences for $T(n)$. Prove that $T(n) = O(2^n)$ using the definition of $O(\cdot)$.

3 Growth rates (20 points)

Fill out the following table : every answer is worth 2 points. For each pair f, g, write "=" if $\Theta(f) = \Theta(g)$, "$\ll$" if $f = o(g)$, "$\gg$" if $g = o(f)$, and "???" if neither of the three applies. You do not need to justify the answers.

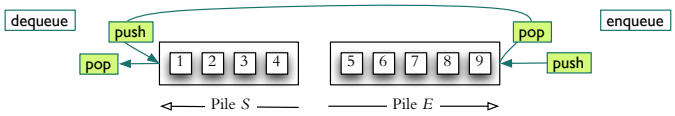
	$f(n)$	$g(n)$
a	$f(n) = \sqrt{n}$	$g(n) = n^{1/3}$
b	$f(n) = 2n + \lg n$	$g(n) = n - 1$
c	$f(n) = 4^{\lg n}$	$g(n) = \sqrt{n}$
d	$f(n) = n!$	$g(n) = n^n$
e	$f(n) = 2^n$	$g(n) = 3^n$
f	$f(n) = \lg n$	$g(n) = \log_3 n$
g	$f(n) = n^2$	$g(n) = 2^{2\log_3 n}$
h	$f(n) = \begin{cases} 1 & \{n = 0\} \\ 2 \cdot f\big(\lfloor n/2 \rfloor\big) + 1 & \{n > 0\} \end{cases}$	$g(n) = n$
i	$f(n) = \begin{cases} 1 & \{n = 0, 1\} \\ f(n-1) + O(n) & \{n > 1\} \end{cases}$	$g(n) = \frac{n^2}{\lg(n+1)}$
j	$f(n) = \begin{cases} 1 & \{n \bmod 2 = 1\} \\ \lg^* n & \{n \bmod 2 = 0\} \end{cases}$	$g(n) = \begin{cases} \lg^* n & \{n \bmod 2 = 1\} \\ 1 & \{n \bmod 2 = 0\} \end{cases}$

4 Exposure (20 points)

Let T be a binary tree. In this exercise, define the **exposure** of a node u as the *shortest* distance to a leaf within u's subtree. A leaf's exposure is 0. Design an algorithm that computes each node's exposure.

5 Two is better than one (30+10 points)

One can implement a queue by using two stacks E and S.




```
Operation enqueue(x)
1  E.push(x)
Operation dequeue()
1  if S is empty then
2       while E is not empty do S.push(E.pop())
3  return S.pop()
```

a. (15 points) Add the operation delete(k) that dequeues k times or stops earlier when the queue becomes empty. The implementation should never result in stack underflow. The operation does not return any value.

b. (15 points) What is the worst-case running time of the operations enqueue and dequeue(k) ? Show that any sequence of n opérations can be executed in $O(n)$ time ; in other words, that the amortized cost is $O(1)$. The stack operations pop and push, as well as the "is empty" test take $O(1)$ time.

c. (10 bonus points) Show a detailed implementation by storing the two stacks *together in a single array*.

GOOD LUCK !

intra-H2009

IFT2015 H09 — Examen Intra

Miklós Csűrös

16 février 2009

Aucune documentation n'est permise. L'examen vaut 100 points.
Répondez à toutes les questions dans les cahiers d'examen.

0 Votre nom (1 point)

Écrivez votre nom et code permanent sur tous les cahiers soumis.

1 Types abstraits (15 points)

Œuf de coucou (3 points) Sur la liste suivante, il y a trois types abstraits : identifiez lesquels.

> pile, liste chaînée, queue (file FIFO), file à priorités, arbre binaire, Barack Obama.

Interface (12 points) Les trois types abstraits (TA) identifiés représentent des ensembles dynamiques avec des opérations élémentaires pour ajouter et déléter des éléments. Spécifiez les deux opérations pour chacun des TAs et expliquez comment elles changent l'ensemble d'éléments représenté.

2 Grand O — petit o (24 points)

a. (6 points) Considérez la fonction $A(n) = \sqrt{3n} + \log_3(n+1) + 3$. Démontrez que $A(n) \in O(\sqrt{n})$ en utilisant la définition de $O(\cdot)$.

b. (6 points) Expliquez ce qu'on veut dire si on écrit $B(n) = \big(2 + o(1)\big)n^2$ pour une fonction $B\colon \{0, 1, \dots\} \mapsto \mathbb{R}^+$.

c. (12 points) Considérez la fonction $C(n)$ définie par

$$\begin{aligned} &C(1) = 1; \\ &C(n) = C(n-1) + 2n^2 \qquad\qquad \{n > 1\}. \end{aligned}$$

Démontrez que $C(n) = O(n^3)$ en utilisant la définition de $O(\cdot)$.

3 Le fond de la pile (30 points)

Écrivez une fonction $\mathsf{enterrer}(P, x)$ qui place l'élément x *au fond* d'une pile P. Par exemple, si P est vide au début, et on exécute la série d'opérations

$$P.\mathsf{push}(3), P.\mathsf{push}(2), \mathsf{enterrer}(P, 8), P.\mathsf{pop}(), P.\mathsf{pop}(), P.\mathsf{pop}(),$$

les trois pops devrait retourner $2, 3, 8$, dans cet ordre. La fonction enterrer ne peut utiliser que l'interface standard de la pile. (Donc vous ne pouvez pas supposer que P est implementé, par exemple, avec un tableau $T[1 \dots n]$ et faire une affectation $T[i] \leftarrow x$ de quelque sorte...) **Indice** : utilisez de la récursion.

4 Les tout petits (30 points)

Écrivez une fonction $\mathsf{petitK}(k, T, n)$ qui retourne les k éléments les plus petits dans un tableau d'entiers $T[1 \dots n]$. La fonction doit prendre un temps de $O(n \log k)$ et utiliser une espace de $O(k)$. **Indice :** utilisez une des structures de données vues au cours.

intra-H2007

IFT2015 H07 — Examen Intra

Miklós Csűrös

20 février 2007

Aucune documentation n'est permise. L'examen vaut 100 points.
Répondez à toutes les questions dans les cahiers d'examen.

0 Votre nom (1 point)

Écrivez votre nom et code permanent sur tous les cahiers soumis.

1 Échauffement (10 points)

Donnez la définition du type abstrait de données (TAD). Décrivez un TAD de votre choix vu dans le cours.

2 Le grand O (30 points)

a. (10 points) Considérez la fonction suivante :

$$f(0) = 1; \qquad f(n) = 7 \cdot f\big(\lfloor n/2 \rfloor\big) + n \quad (n > 0)$$

Démontrez que $f(n) \in O(n^3)$.

b. (10 points) Considérez la fonction suivante :

$$g(0) = g(1) = 1; \qquad g(n) = g\big(\lfloor n/3 \rfloor\big) + 3 \quad (n > 0)$$

Trouvez la meilleure borne asymptotique pour $g(n)$. Justifiez votre réponse.

c. (10 points) Démontrez que $3^{n+1} \in O(3^n)$, mais $3^{2n} \notin O(3^n)$.

3 Le bronze (10 points)

Donnez un algorithme qui trouve le troisième plus petit élément dans un tableau `A` représentant un tas binaire (min-tas). Analysez son temps de calcul asymptotique en fonction de la taille n de `A`.

4 Quadtree (25 points)

Un *quadtree* est une structure de données basée sur un arbre quaternaire. Il sert à représenter un ensemble de points en deux dimensions. Chaque nœud u de l'arbre est associé avec un point $\mathsf{P}(u) = (\mathsf{X}(u), \mathsf{Y}(u))$. Les quatre sous-arbres contiennent points vers nord-ouest, nord-est, sud-est et sud-ouest de $\mathsf{P}(u)$. Le fragment de code suivant montre l'implantation d'un tel nœud en Java.

```
class QuadNode
{
    private QuadNode parent;
    private QuadNode NW, NE, SE, SW; // les enfants
    private double X, Y; // les coordonnés du point associé
    ...
}
```

Si un des quadrants ne contient aucun point, alors l'enfant correspondant est `null` ; tous les enfants sont `null` si u est une feuille.

Si u est le parent de v, on a toujours

$$\begin{aligned}
\mathsf{X}(v) < \mathsf{X}(u), \mathsf{Y}(v) > \mathsf{Y}(u) &\qquad \text{si } v \text{ est à nord-ouest (NW)} \\
\mathsf{X}(v) \geq \mathsf{X}(u), \mathsf{Y}(v) > \mathsf{Y}(u) &\qquad \text{si } v \text{ est à nord-est (NE)} \\
\mathsf{X}(v) \geq \mathsf{X}(u), \mathsf{Y}(v) \leq \mathsf{Y}(u) &\qquad \text{si } v \text{ est à sud-est (SE)} \\
\mathsf{X}(v) < \mathsf{X}(u), \mathsf{Y}(v) \leq \mathsf{Y}(u) &\qquad \text{si } v \text{ est à sud-ouest (SW)}
\end{aligned}$$

Figure 1 montre l'exemple d'un quadtree.

a. (13 points) Démontrez que la hauteur d'un quadtree sur n points est entre $(n-1)$ et $\log_4(3n+1) - 1$.

b. (12 points) Donnez un algorithme pour compter les points à l'est de la racine. Plus précisement, calculez le nombre de nœuds v dans l'arbre avec $\mathsf{X}(v) \geq \mathsf{X}(r)$ où r est la racine du quadtree.

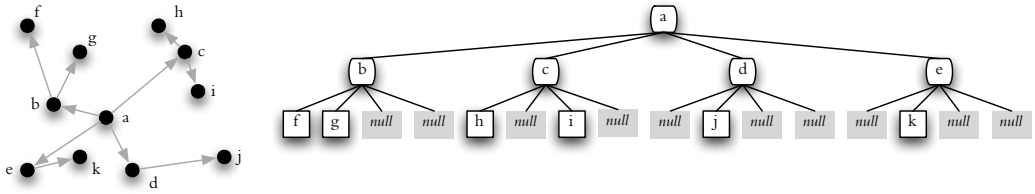


FIG. 1 – Un ensemble de points et un *quadtree* de hauteur 2.

5 Hello moto (24 points)

Vous devez écrire un logiciel pour suivre les transactions de la Bourse sur un téléphone cellulaire. Le fournisseur transmet immédiatement les transactions et le logiciel doit afficher à tout temps la liste des m transactions les plus volumineuses de la journée. (Si $v_1, v_2, \ldots$ est la liste des valeurs transmises, on veut être capable d'énumerer les m valeurs qui sont les plus grandes parmi $v_1, v_2, \ldots, v_i$, pour tout i.) Proposez une solution efficace avec l'analyse de son temps et espace de calcul dans le cas de n transactions. Comme il s'agit d'un cellulaire, il est essentiel de minimiser le temps et l'espace requis. En particulier, il est impossible de stocker la liste de toutes les transactions de la journée.

6 Asymptotique (10 points de boni)

Trouvez l'ordre correct de croissance en 2a. Plus précisement, trouvez une fonction explicite $h(n)$ telle que $f(n) \in \Theta\big(h(n)\big)$. Justifiez votre réponse.

intra-H2006

IFT2010 H06 — Examen Intra

Miklós Csűrös

21 février 2006

Aucune documentation n'est permise à l'exception d'une feuille de $8\frac{1}{2}'' \times 11''$. L'examen vaut 100 points. Le problème 6 est optionnel pour 15 points de boni. Il y a d'autre questions où vous pouvez obtenir 6 points de boni en total.

Répondez à toutes les questions dans les cahiers d'examen.

0 Votre nom (1 point)

Écrivez votre nom et code permanent sur tous les cahiers soumis.

1 Oh (12 points)

Remplissez la dernière colonne du tableau suivant. Mettez Θ pour dénoter «$f(n) \in \theta(g(n))$», Ω pour «$f(n) \in \Omega(g(n))$», O pour «$f(n) \in O(g(n))$», o pour «$f(n) \in o(g(n))$» et ω pour «$g(n) \in o(f(n))$». Chaque ligne vaut 2 points : 2 points pour la meilleure réponse ou 0 sinon (donc 0 points pour un O quand on peut mettre Θ). Vous ne devez donner aucune justification à vos réponses ici. (Comme d'habitude, $\lg n = \log_2 n$.)

	$f(n)$	$g(n)$	relation
a	$2n^2 - \lg n$	$(n+2)^2$	
b	$n+1$	$n-1$	
c	2^n	3^n	
d	$0.01\sqrt{n}$	$12(\lg n)^2$	
e	$3n^3$	$8^{\lg n}$	
f	$4^{\sqrt{n}}$	3^n	

2 Pile $\leftrightarrow$ file (15 points)

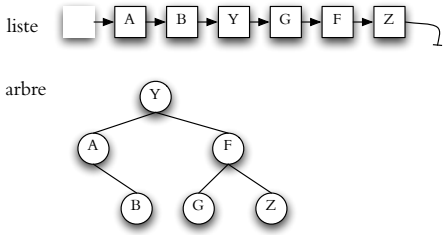
On a montré lors d'un exercice qu'on peut implanter une queue en utilisant deux piles. Donc il ne serait pas surprenant si on pouvait implanter une pile en utilisant deux queues. Mais peut-on implanter une pile avec *une* queue ? (Oui !) Montrez comment le faire. Les opérations permises sur la queue sont : enqueue, dequeue et isEmpty. Vous devez montrer l'implantation de pop, push et isEmpty pour la pile. L'implantation peut utiliser une variable simple additionnelle de n'importe quel type (si nécessaire) et une queue.

Analysez le temps de calcul des opérations quand la pile contient n éléments en supposant que toutes les opérations de la queue sont de $O(1)$.

3 Parcours (7 points)

Donnez un algorithme récursif pour l'énumération des élements dans un arbre binaire de recherche dans l'ordre *décroissant*. Comme d'habitude, vous pouvez utiliser les opérations $\mathsf{left}(x)$, $\mathsf{right}(x)$ et $\mathsf{parent}(x)$ pour trouver l'enfant gauche, enfant droit et le parent du nœud x.

4 Liste $\leftrightarrow$ arbre (35 points)



On veut implanter un TAD liste en se reliant à un arbre binaire balancé comme l'arbre AVL ou l'arbre rouge et noir. Comme d'habitude, la valeur stockée à un nœud x est dénotée par $\mathsf{val}(x)$, mais les nœuds ne sont pas organisés par $\mathsf{val}(x)$ mais plutôt par leur ordre dans la liste. Le nœud x qui correspond au k-ème élément sur la liste est le k-ème dans le parcours infixe. Pour ceci, on maintient une variable $\mathsf{taille}(x)$ associée à chaque nœud x qui est le nombre de nœuds dans le sous-arbre enraciné à x (y incluant x). Dans ce qui suit, vous devez développer les algorithmes pour l'implantation de la liste TAD. Votre code peut utiliser les opérations usuelles de $\mathsf{left}(x)$, $\mathsf{right}(x)$ et $\mathsf{parent}(x)$ pour naviguer dans l'arbre. Un enfant ou parent manquant est dénoté par null. Pour simplifier la code, on pose $\mathsf{taille}(\mathsf{null}) = 0$.

a (5 points) Donnez un algorithme récursif qui calcule taille() pour tous les nœuds dans le sous-arbre de x.

```
CALCULER-TAILLE(x) // x est un nœud
G1  if x est une feuille then taille(x) ← 1 ;
G2  else // ici vient votre code pour le cas où x n'est pas une feuille
G3  return taille(x)
```

b (5 points) Donnez un algorithme qui retourne le k-ème nœud dans le parcours infixe, en utilisant les champs taille. L'idée est d'implanter une fonction NŒUD-K(x, k) qui trouve le k-ème élément dans le parcours infixe du sous-arbre de x. (1 point boni pour un algorithme non-récursif — le patron ici est pour une solution récursive.)

```
NŒUD-K(x, k) // x est un nœud, k est un entier
K1  if k ≤ 0 ou k > taille(x) then erreur !
K2  if k = ☐ then return x
K3  if k < ☐ then return NŒUD-K(left(x), ☐)
K4  if k > ☐ then return NŒUD-K(right(x), ☐)
```

c (20 points) Expliquez comment on peut mettre à jour les valeurs taille après insertion ou deletion d'un nœud. Pour ceci, notez que quel que soit l'arbre qu'on utilise (AVL ou RN) une telle opération est executée en deux phases : insertion/suppression d'un nœud sans maintenir l'équilibre, suivi par des tests et des rotations. Montrez ce qui ce passe dans la première phase (insertion ou deletion dans un arbre binaire de recherche), et comment les tailles des nœuds doivent changer. Ensuite, montrez comment et à quels nœuds taille doit changer lors d'une rotation gauche ou droite. Démontrez que les opérations suivantes de l'ADT liste peuvent être implantées avec un temps d'exécution de $O(\log n)$ (sur n éléments) : Kth(k) [retourne le k-ème élément de la liste] insert(k, x) [insère x en position k de la liste] et delete(k) [supprime le k-ème élément de la liste].

d (5 points) Est-ce qu'il y a des avantages à cette implantation par rapport à d'autres solutions qu'on a vues (tableau et liste chaînée avec pointeurs) ? (Justifiez votre réponse.)

5 Arbre rouge-rouge-noir (30 points)

5.1 Coloriage et rrang (15 points)

Un *arbre rouge-rouge-noir* (RRN) est un arbre binaire de recherce avec des feuilles null comme les arbres rouge et noir, dans lequel les nœuds sont équipés d'un rrang. (Ne pas confondre avec le rang des arbres RN.) Les rrangs des nœuds satisfont les propriétés suivantes.

1. Pour chaque nœud x excepté la racine,

$$\mathsf{rrang}(x) \leq \mathsf{rrang}\big(\mathsf{parent}(x)\big) \leq \mathsf{rrang}(x) + 1.$$

2. Pour chaque nœud x avec un arrière-grand-parent $y = \mathsf{parent}(\mathsf{parent}(\mathsf{parent}(x)))$,

$$\mathsf{rrang}(x) < \mathsf{rrang}(y).$$

3. Pour chaque feuille x on a $\mathsf{rrang}(x) = 0$ et $\mathsf{rrang}\big(\mathsf{parent}(x)\big) = 1$.

On colorie les nœuds soit rouge soit noir : le nœud x est rouge si et seulement s'il n'est pas la racine et $\mathsf{rrang}(x) = \mathsf{rrang}\big(\mathsf{parent}(x)\big)$. Tous les autres nœuds sont noirs.

Complétez l'énoncé du théorème suivant et démontrez qu'il est vrai.

Théorème 1. *Le coloriage d'un arbre rouge-rouge-noir est tel que*

(i) chaque feuille est colorié par ☐.

(ii) si le parent d'un nœud ☐ *est* ☐*, alors son grand-parent (s'il existe) est* ☐.

(iii) chaque chemin reliant un nœud à une feuille dans son sous-arbre contient le même nombre de nœuds ☐.

5.2 La hauteur d'un arbre rouge-rouge-noir (15 points)

On veut démontrer que la hauteur d'un arbre rouge-rouge-noir avec n nœuds internes est $O(\log n)$. Il n'est pas trop difficile de démontrer le théorème suivant.

Théorème 2. *Pour tout nœud x d'un arbe rouge-rouge-noir, le nombre de nœuds internes dans le sous-arbre enraciné à x est au moins $2^{\mathsf{rrang}(x)} - 1$.*

Vous pouvez utiliser ce théorème sans preuve en **b** ici. Pour 5 points de boni, donnez une preuve formelle au théorème.

a (7 points) Démontrez que

$$\mathsf{h}(x) \leq 3 \cdot \mathsf{rrang}(x) \qquad (\star)$$

pour chaque nœud x. (Vous avez besoin du théorème 1 dans la preuve.)

b (8 points) Utilisez le théorème 2 et l'équation ($\star$) pour démontrer que la hauteur d'un arbre RRN avec n nœuds internes est borné par $c\lfloor \lg(n+1) \rfloor$ — remplacez c par un nombre aussi petit que possible. Comparez ce résultat avec les bornes sur la hauteur des arbres AVL et RN.

6 Et maintenant, quelque chose de complètement différent (15 points de boni)

Les chaînes Fibonacci f_n sont des mots sur l'alphabet $\{\mathtt{1}, \mathtt{0}\}$ définis par induction de la manière suivante. Pour $n = 0$, $f_0 = \mathtt{1}$. Pour $n > 0$, f_n est dérivée de f_{n-1} en remplaçant chaque `1` par `0` et chaque `0` par `01`. (On remplace tous les caractères de f_{n-1} en même temps.) On a donc $f_0 = \mathtt{1}$, $f_1 = \mathtt{0}$, $f_2 = \mathtt{01}$, $f_3 = \mathtt{010}$, $f_4 = \mathtt{01001}$, $f_5 = \mathtt{01001010}$, $f_6 = \mathtt{0100101001001}$, ...

Démontrez que $f_n = f_{n-1} \cdot f_{n-2}$ pour chaque $n > 1$ (le symbol $\cdot$ dénote la concaténation).