E1. Do třídy `List` dopište kód metod `remove` a `removeAll`. Dejte si pozor, abyste správně ošetřili všechny krajní případy!

```
class Node {
    int contents;
    Node next;

    Node(int c, Node n) {
        contents = c;
        next = n;
    }
}

class List {
    Node first;

    void add(int c) {
        first = new Node(c, first);
    }

    void remove(int c) {
        // odstrani jeden vyskyt daneho cisla
    }

    void removeAll(int c) {
        // odstrani vsechny vyskyty daneho cisla
    }
}
```

E2. Nakreslete, jak bude vypadat binární vyhledávací strom (varianta, ve které jsou všechny prvky v listech a ve vnitřních uzlech jsou pouze klíče umožňující navigaci po stromě) po vykonání následující posloupnosti operací: $insert(5)$, $insert(3), insert(4), remove(5), insert(7)$.

E3. Z definice dokažte: $f \in O(g) \Rightarrow f \notin \omega(g)$.

E4. V implementaci metody `merge` je chyba. Vyznačte kde a vysvětlete proč.

```
interface List {
    int removeFirst(); // odstrani prvni prvek ze seznamu a vrati ho
    void append(int c); // vlozi dany prvek na konec seznamu
    boolean isEmpty(); // vrati, zda je prazdny
}

class MyList implements List { /* kod */ }

List merge(List a, List b) { // sleje a vrati oba seznamy
    List result = new MyList();
    while ((! a.isEmpty()) && (! b.isEmpty())) {
        int aa = a.removeFirst();
        int bb = b.removeFirst();
        if (aa < bb) result.append(aa);
        else result.append(bb);
    }
    while (! a.isEmpty()) result.append(a.removeFirst());
    while (! b.isEmpty()) result.append(b.removeFirst());
    return result;
}
```

E5. Spočítejte asymptotickou časovou složitost funkce `f` v závislosti na `n`.

```
boolean f(int n) {
    // predpokladame, ze n >= 0
    if (n == 0) return true;
    if (n == 1) return false;
    return f(n - 2);
}
```

E6. Dopište kód metody `enlarge` v rozptylovací tabulce `Table`.

```
class Table {
    java.util.List[] array;
    int size;

    Table() {
        array = new java.util.ArrayList[10];
        for (int i = 0; i < array.length; i++)
            array[i] = new java.util.ArrayList();
        size = 0;
    }

    void add(Object o) {
        if (array.length < size) enlarge();
        array[o.hashCode() % array.length].add(o);
        size++;
    }

    void enlarge() {
        java.util.List[] newArray =
            new java.util.ArrayList[array.length * 2];
        for (int i = 0; i < newArray.length; i++)
            newArray[i] = new java.util.ArrayList();









        array = newArray;
    }

    /* kod dalsich metod */
}
```

E7. Funkce `Object f(Object param)` se chová jako matematická funkce, tzn. nemá žádné vedlejší efekty a pokud ji nad stejným parametrem zavoláte vícekrát, vždycky vrátí stejný objekt. Vy jste se rozhodli její běh urychlit přidáním keše — chcete si pamatovat všechny v minulosti použité parametry a k nim vrácené hodnoty, abyste při opakovaném volání funkce `f` nad stejným parametrem nemuseli návratovou hodnotu znovu počítat a mohli vrátit rovnou příslušný zapamatovaný objekt. Jakou datovou strukturu k implementaci keše použijete a proč?

D1. Stručně vysvětlete, co to je dosvědčující algoritmus (certifying algorithm). Navrhněte a popište, jak by mohl vypadat certifikát pro algoritmus quickselect, který by dovolil správnost výsledku ověřit v lineárním čase.

D2. Jak byste naimplementovali rozptylovací tabulku, která při vložení jedenáctého prvku nejstarší vložený prvek zapomene (tzn. pamatuje si maximálně deset prvků)? Naznačte svoje řešení v pseudokódu, můžete předpokládát, že standardní datové struktury už jsou naimplementovány a máte je k dispozici.

D3. V každém a, b-stromu musí platit $b \geq 2a - 1$. Proč?

D4. Předpokládejte, že při násobením matic o velikostech $n \times m$ a $m \times l$ potřebujete přesně $n \cdot m \cdot l$ operací. Ověřte si, že při násobení $(M_1 \cdot M_2) \cdot M_3$ a $M_1 \cdot (M_2 \cdot M_3)$ dostanete stejný výsledek za použití různého počtu operací a pomocí dynamického programování najděte uzávorkování minimalizující počet operací při násobení řady matic $M_1 \cdot M_2 \cdot \ldots \cdot M_n$. Postupujte v těchto krocích:

- napište, jak najít optimální řešení dané úlohy z optimálních řešení podúloh,
- poté se pokuste tuto rovnici převést do kódu, který nepoužívá tabulku, a
- teprve nakonec se zamyslete nad tím, jak kešovat řešení již spočítaných podúloh.

D5. Máte seznam objektů, které můžete porovnat pomocí funkce `compare` (vrací -1, 0, 1 podle toho, jestli je první objekt menší, roven nebo větší než druhý) a chcete zjistit, jestli daný seznam obsahuje duplicity. Jaký je nejlepší způsob, když chcete dosáhnout nejlepšího průměrného času, a jak dosáhnete nejlepšího času v nejhorším případě? Svoji odpověd stručně zdůvodněte.