E1. Do třídy `List` dopište metodu `void remove(Object o)`, která odstraní zadaný prvek ze spojového seznamu. Prvky porovnávejte pomocí `equals`, dejte si pozor, abyste správně ošetřili všechny krajní případy!

```
class List {
    Node first;

    List() {}
    void add(Object o) {
        if (first == null) first = new Node(o);
        else first = new Node(o, first, first.previous);
    }
}

class Node {
    Object contents;
    Node next, previous;

    Node(Object c, Node n, Node p) {
        contents = c;
        next = n;
        previous = p;
        next.previous = this;
        previous.next = this;
    }

    Node(Object c) {
        contents = c;
        next = this;
        previous = this;
    }
}
```

E2. Napište metodu `void sort(int[] array)`, která pomocí třídění haldou (heapsort) setřídí pole `array` vzestupně. Pro reprezentaci haldy využijte přímo pole `array` (samozřejmě tak, abyste nepřišli o žádná vstupní data).

E3. Máte dvě funkce: f a g. Víte, že $f \in O(g)$. Z definice dokažte, že $f \in O(g/2)$.
Tip: rozepište si definici množiny O pro g a pro $g/2$ vedle sebe.

E4. Třída `Heap` má na sobě metody `boolean isEmpty()`, `Object removeMin()` a `int getSize()`. Napište kód, který dvě haldy `a` a `b` do jednoho vzestupně setříděného pole. Pro porovnání prvků použijte metodu `int Util.compare(Object x, Object y)`, která vrací -1, 0, 1 pokud `x` je menší, rovno nebo větší než `y`.

E5. V metodě `remove` třídy `Dictionary` je chyba. Vyznačte kde a vysvětlete proč (opravovat ji nemusíte).

```
class Dictionary {
    static final int INITIAL_CAPACITY = 97;
    Node[] array = new Node[INITIAL_CAPACITY];
    int size = 0;

    void add(Object key, Object value) {
        if (size == array.length) enlarge();

        int i = key.hashCode() % array.length;
        while (array[i] != null) i = i + 1 % array.length;
        array[i] = new Node(key, value);
        size++;
    }

    // kod metody enlarge vynechan

    void remove(Object key) {
        int i = key.hashCode() % array.length;
        while (array[i] != null && ! array[i].key.equals(key))
            i = i + 1 % array.length;
        if (array[i] == null) return;
        array[i] = null;
        size--;

        if (size < array.length / 2) shrink();
    }

    // kod metody shrink vynechan
}

class Node {
    Object key, value;

    public Node(Object k, Object v) {
        key = k; value = v;
    }
}
```

E6. Pro funkci `h` určete asymptotickou časovou složitost v nejhorším případě (v závislosti na hodnotě `n`). Svoji odpověď zdůvodněte.

```
int h(int n) { // na vstupu predpokladame n >= 1
    if (n <= 2) return 1;
    int r = 0;
    for (int i = 1; i <= n; i++) r += n;
    return r + h(n - 2);
}
```

E7. Naprogramujte funkci `void printFrequencies(String[] text)`, která ke každému řetězci v poli `text` vytiskne kolikrát se v daném poli vyskytuje. Pro implementaci použijte standardní třídu Javy `java.util.HashMap` (pravděpodobně budete potřebovat metody `void put(Object key, Object value)`, `Object get(Object key)` — při nenalezení prvku vrátí `null` — a `Iterator iterator()`).

D1. Třída `Interval` reprezentuje interval začínající v `a` a končící v `b`. Naprogramujte metodu `Set<Inteval> connect(Set<Interval> intervals, int c, int d)`, která pomocí dynamického programování ze zadané množiny intervalů vybere podmnožinu, ve které se žádné dva intervaly nepřekrývají a jejichž sjednocení tvoří interval z `c` do `d`. V případě neúspěchu vraťte `null`.

```
class Interval {
    int getA() { /* kod */ }
    int getB() { /* kod */ }
    /* zbytek tridy */
}
```

D2. Spočítejte časovou složitost v nejhorším případě (pozor: přesnou, ne jen asymptotickou) funkce `max`. Předpokládejte, že načtení hodnoty proměnné nebo konstanty trvá jeden krok, přístup do pole jeden krok plus doba potřebná pro výpočet indexu, přiřazení do proměnné jeden krok plus doba potřebná na výpočet přiřazované hodnoty, porovnání jeden krok plus doba potřebná na vyhodnocení operandů, podmíněný nebo nepodmíněný skok jeden krok a vrácení návratové hodnoty funkce jeden krok plus doba potřebná pro výpočet vracené hodnoty. Svůj výpočet rozepište tak, aby jeho struktura odpovídala struktuře kódu.

```
int max(int[] array) {
    int temp = array[0];
    for (int i = 1; i < array.length; i++)
        if (array[i] > temp) temp = array[i];
    return temp;
}
```

D3. Napište funkci `int[] buildHeap(int[] array)`, která ze setříděného pole `array` vyrobí haldu (také uloženou v poli). Snažte se, aby vaše funkce běžela co nejrychleji a aby byla co nejjednodušší.

D4. Třída `Node` reprezentuje uzel v 3,5-stromu. Dopište metodu `split`, předpokládejte, že pokus o rebalancování už proběhl.

```
class Node {
    final int[] contents = new int[5];
    final Node[] children = new Node[6];
    int occupiedValuesCount = 0;
    Node parent;

    void insert(Node child) { /* kod */ }
    void split(int additionalValue) {
        /* doplnte */
    }
}
```

D5. Odvoďte amortizovanou složitost metody `add`.

```
class Array {
    int[] contents = new int[1];
    int size = 0;

    void add(int a) {
        if (contents.length == size) {
            int[] newContents = new int[contents.length * 2];
            for (int i = 0; i < contents.length; i++)
                newContents[i] = contents[i];
            contents = newContents;
        }
        contents[size++] = a;
    }

    /* dalsi metody */
}
```

C1. Předpokládejte, že máte metodu `int findMin(int[] array, int left, int right)`, která vám v konstantním čase vrátí index prvku s nejmenší hodnotou v poli `array` mezi indexy `left` a `right` včetně. Napište metodu `void sort(int[] array)`, která v co nejmenším čase setřídí svůj parametr.

C2. Napište metodu, která ze setříděného pole vyrobí ideálně vyvážený (tzn. délka všech větví se liší maximálně o jedna) binární strom.

C3. V uzavřeném rozptylování můžete při odebrání prvku buď všechny následující prvky posunout dopředu nebo na daný index poznamenat speciální hodnotu, která značí nevyužitý index. Z výkonového hlediska mají obě varianty svoje výhody a nevýhody — zkuste je popsat.

C4. Při zvětšení kapacity rozptylovací tabulky je potřeba všechny prvky přehashovat. Jak to, že to nemá vliv na výkonovou charakteristiku operace vkládání?

C5. Třída `Interval` reprezentuje interval začínající v `a` a končící v `b`. Dá se pro hledání nepřekrývající se podmnožiny intervalů, jejichž sjednocení dává interval od `c` do `d` aplikovat nějaký hladový algoritmus? Pokud ano, ukažte jak, pokud ne, ukažte protipříklad.