26. 11. 2010

1. (a) Podle definice napište důkaz následujícího tvrzení: $4n + 1 \in O(3n)$.
   (b) Podle definice napište důkaz následujícího tvrzení: $n - 100 \in \Theta(n + 100)$.
   (c) Podle definice napište důkaz následujícího tvrzení: $n^2 \in \omega(100n + 7777777)$.

a) $\mathcal{O}(f(n)) = \{g(n) : \exists c \in \mathbb{R}^+ : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$

$\underbrace{4n + 1}_{g(n)} \in \underbrace{O(3n)}_{f(n)} \longrightarrow 4n + 1 \leq c \cdot 3n$

pro $n_0 = 1$ $\quad 4n + 1 \leq c \cdot 3n$

$4 + 1 \leq c \cdot 3 \longrightarrow$ [illegible] $c = 2$

b) $\Theta(f(n)) = \mathcal{O}(f(n)) \cap \Omega(f(n))$

$n - 100 \in \Theta(n + 100)$

$\Theta(f(n)) = \{g(n) : \exists c > 0, \exists d > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : d \cdot f(n) \leq g(n) \leq c \cdot f(n)\}$

$\underbrace{n - 100}_{g(n)} \in \Theta(\underbrace{n + 100}_{f(n)})$

$d \cdot f(n) \leq g(n) \leq c \cdot f(n) \longrightarrow d(n + 100) \leq n - 100 \leq c(n + 100)$

pro $n_0 = 100$ $\quad d(100 + 100) \leq 100 - 100 \leq c(100 + 100)$

$0(200) \leq 0 \leq 1(200)$ $\quad d \leq 0 \rightarrow (\infty, 0\rangle$

$c > 0 \rightarrow (0, \infty)$

26. 11. 2010

c) $n^2 \in \omega(100n + 7777777)$

$\omega(f(n)) = \{g(n) : \forall c \in \mathbb{R}^+ : \exists n_0 : \forall n \geq n_0 : g \geq c\, f(n)\}$

$n^2 \geq c \cdot (100n)$

7777777 - lze zanedbat, protože jde o proměnnou konstantu

pro $n_0 = 1 \rightarrow 1^2 \geq c\,(100 \cdot 1)$

$1 \geq c \cdot 100 \rightarrow \frac{1}{100} \geq c$

26.11.2010

2.

(a) Pro funkci f určete asymptotickou časovou složitost v nejhorším případě (v závislosti na hodnotě n). Svoji odpověď zdůvodněte.

```
int f(int n) {                  // na vstupu predpokladame n >= 1
    return n == 1 ? 1 : f(n - 1) + f(n - 1);
}
```

$$T(n) \begin{cases} c & n = 1 \\ a \cdot n + b\,T\left(\frac{n}{d}\right) & n > 1 \\ n < 1 & \end{cases}$$

$b = d$: $n \log n$

$b < d$: n

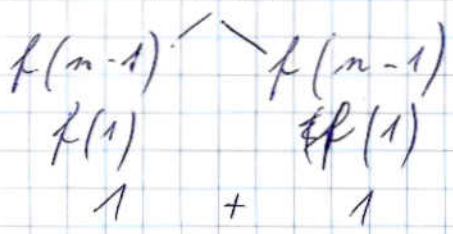
$b > d$: $n^{\log_d b}$

$$T(n) = \begin{cases} 1 \text{ pro } n = 1,\ c = 1 \\ T(n-1) + T(n-1) + a = 2\left(T(n-1) + a\right) \end{cases}$$

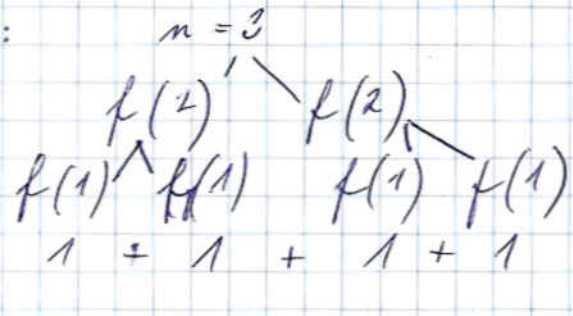
→ lze zanedbat

pro $n = 1$: $f(n) = 1$

pro $n = 2$:



pro $n = 3$:



$2^{n-1} \Rightarrow \mathcal{O}(2^n)$

výpočet

$T(n) = 2T(n-1)$

$T(1) = 1$

$T(2) = 2T(1) = 2 \cdot 1 = 2$

$T(3) = 2T(2) = 2 \cdot 2 = 4$

$T(4) = 2T(3) = 2 \cdot 4 = 8$

$T(5) = 2T(4) = 2 \cdot 8 = 16$

$\vdots$

$T(n) = 2^1 T(n-1) = 2^2 T(n-1) = 2^3 T(n-1) \dots = 2^k T(k-1)$

$n - 1 = 1 \Rightarrow n = k \Rightarrow \underline{\underline{T(n) = 2^n}}$

(b) Pro funkci g určete asymptotickou časovou složitost v nejhorším případě (v závislosti na hodnotě n). Svoji odpověď zdůvodněte.

```
int g(int n) {                    // na vstupu predpokladame n >= 1
    return n <= 1 ? 1 : g(n / 2) + g(n / 2);
}
```

$T(n) = 1$ pro $n \leq 1$ → c

$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + a = 2T\left(\frac{n}{2}\right) + a$ → lze zanedbat

$g(1) = 1$

$g(2) =$ pro $n = 2$

$g(1) \wedge g(1) = 2^{\log n} = 2^1 = 1$

$g(3) =$ pro $n = 3$

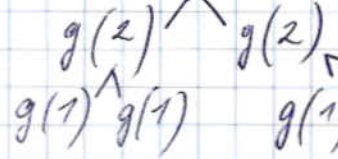
$g(1) \wedge g(1) = 2^1 = 1$

$g(4) =$ pro $n = 4$

$g(2) \wedge g(2)$

$g(1) \wedge g(1) \quad g(1) \wedge g(1) = 4 = 2^2$

$g(5)$ = pro $n = 5$



$= 4 = 2^2$

$T(n) = 2T\left(\frac{n}{2}\right)$

$T(1) = 1$ — 2^0

$T(2) = 2T(1) = 2$

$T(3) = 2T\left(\frac{3}{2}\right) = 2T(1) = 2$ — 2^1

$T(4) = 2T(2) = 2 \cdot 2 = 4$

$T(7) = 2T\left(\frac{7}{2}\right) = 2 \cdot 2 = 4$ — 2^2

$T(8) = 2T(4) = 2 \cdot 4 = 8$

$T(15) = 2T\left(\frac{15}{7}\right) = 2 \cdot 4 = 8$ — 2^3

$T(16) = 2T(8) = 2 \cdot 8 = 16$ — 2^4

$$T(n) = 2T\left(\frac{n}{2}\right) = 2 \cdot 2 \cdot T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{4}\right) + 2T\left(\frac{n}{4}\right)$$
$$= 2 \cdot 2T\left(\frac{n}{4}\right) = 2 \cdot 2T\left(\frac{n}{8}\right) + 2 \cdot 2T\left(\frac{n}{8}\right) = \dots$$
$$= 2^k T\left(\frac{n}{2^k}\right) = 2^{\log_2 n} \cdot T(1) = n$$

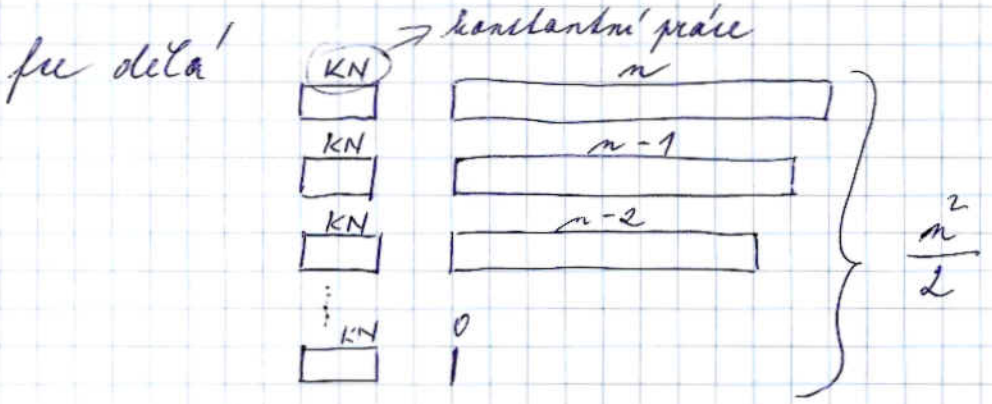
$T(1) = T\left(\frac{n}{2^k}\right)$ $\quad \frac{n}{2^k} = 1 \iff n = 2^k$

$k = \log_2 n$

složitost $O(n)$

(c) 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 <= 1) return 1;
    int r = 0;
    for (int i = 1; i <= n; i++) r += n;
    return r + h(n - 1);
}
```



$KN + \frac{n^2}{2} \Rightarrow$ složitost $= n^2$ $\quad O(n^2)$

Tento příklad je lepší ukázkou. Aplikace na Master Theorem je nevhodná.

Pokus o výpočet:

$T(n) = 1 \quad , \quad n \leq 1$

$T(n) = n + T(n-1)$

$T(1) = 1 + 1 = 2$

$T(2) = 2 + T(1) = 2 + 1 = 3$

$T(n) = n + T(n-1) = n + (n-1) + T(n-2) = n + (n-1) + (n-2) + T(n-3) =$

$= \dots$

$n - k - 1 \leq 1 \qquad n - k \leq 2 \qquad k = 2 \,, \; k = n$

$n - 2 \geq k$

$T(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2 + n}{2} =$

nejsilnější $\Rightarrow O(n^2)$

26. 11. 2010

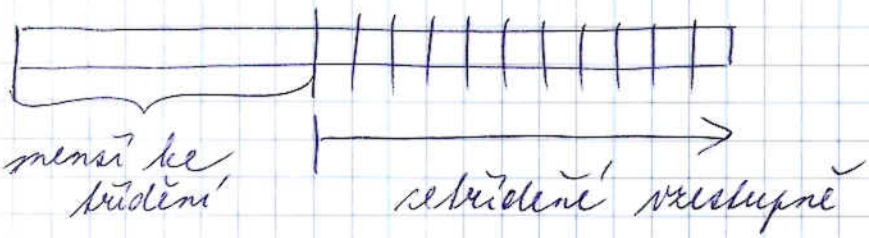
3.

(a) Metoda sort setřídí pole array zleva vzestupně. Určete invariant, který platí na konci každé iterace vnější smyčky a který se dá použít pro důkaz její správnosti.

```
void sort(int[] array) {
    for (int i = array.length - 1; i >= 0; i--) {
            int index = i;
            for (int j = 0; j < i; j++)
                    if (array[j] > array[index]) index = j;
            int helper = array[i];
            array[i] = array[index];
            array[index] = helper;
    }
}
```

array [i] ≥ MAX (arr [0] ... arr [i-1])

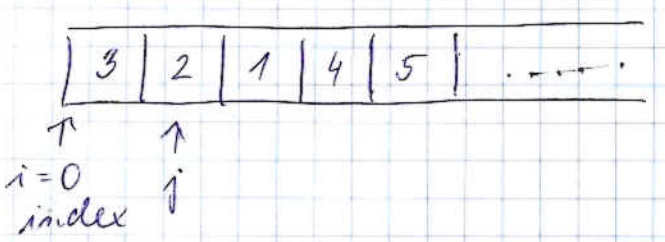
MAX (a[0] ... a[i-1]) ≤ MIN (a[i] ... a[index])

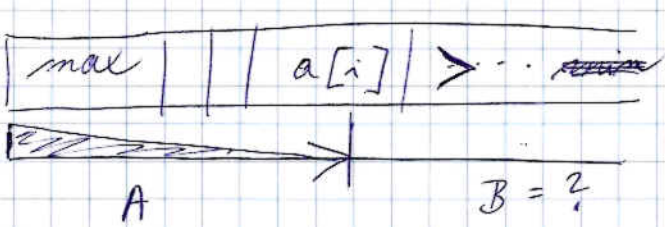


(b) Metoda sort setřídí pole array zleva sestupně. Určete invariant, který platí na konci každé iterace vnější smyčky a který se dá použít pro důkaz její správnosti.

```
void sort(int[] array) {
    for (int i = 0; i < array.length; i++) {
            int index = i;
            for (int j = i + 1; j < array.length; j++)
                    if (array[j] > array[index]) index = j;
            int helper = array[i];
            array[i] = array[index];
            array[index] = helper;
    }
}
```

opačně než (3a)





1. podmínka = začátek (A) je setříděný sestupně

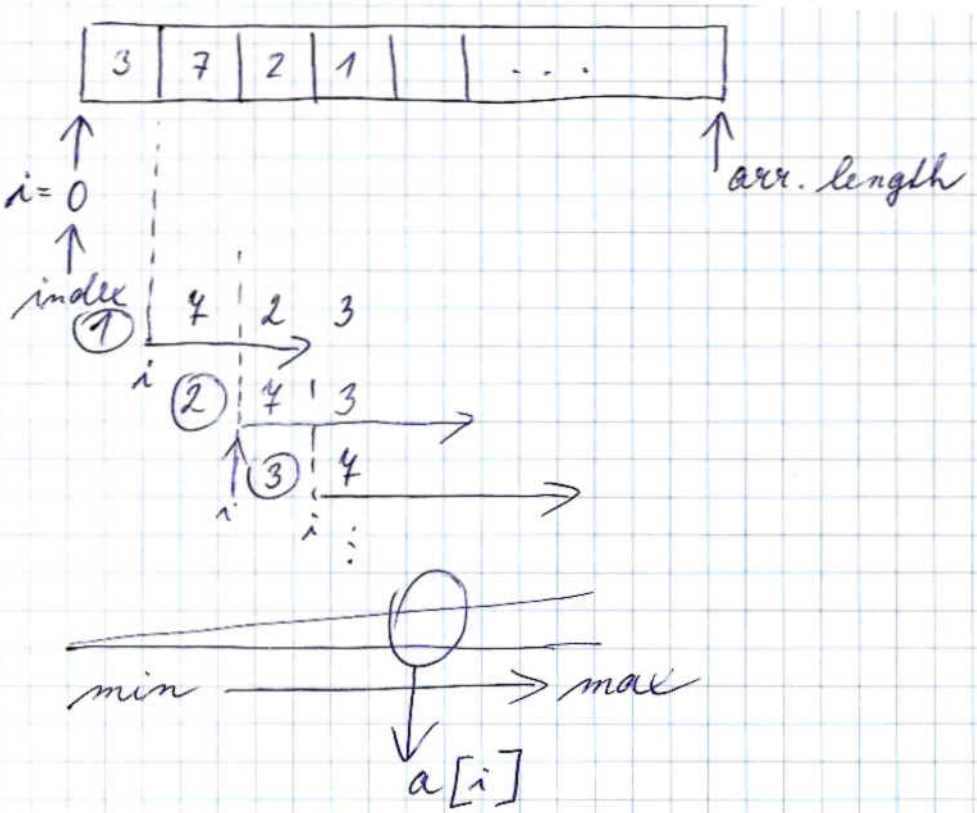
2. podmínka = v části (B) není $x > a[i]$ nebo

min z A (= a[i]) ≥ max z B

4

(c) Metoda sort setřídí pole array zleva vzestupně. Určete invariant, který platí na konci každé iterace vnější smyčky a který se dá použít pro důkaz její správnosti.

```
void sort(int[] array) {
    for (int i = 0; i < array.length; i++) {
            int index = i;
            for (int j = i + 1; j < array.length; j++)
                    if (array[j] < array[index]) index = j;
            int helper = array[i];
            array[i] = array[index];
            array[index] = helper;
    }
}
```



$a_k \leq a_{k+1}$, $k \in \langle 0, i \rangle$

$a_i \leq max\,(a_i \ldots a_{length})$

4.

(a) Do třídy List dopište metodu void remove (Object c) (odstranění daného objektu ze seznamu). Pro porovnání objektů použijte metodu equals, ošetřete i případ, kdy odstraňovaný objekt v seznamu není.

```
class List {
    final Node node;

    public List() {
            node = new Node();
            node.next = node;
            node.previous = node;
    }

    public void add(Object c) {
            Node temp = new Node();
            temp.content = c;
            temp.next = node.next;
            temp.previous = node;
            node.next.previous = temp;
            node.next = temp;
    }
}
```

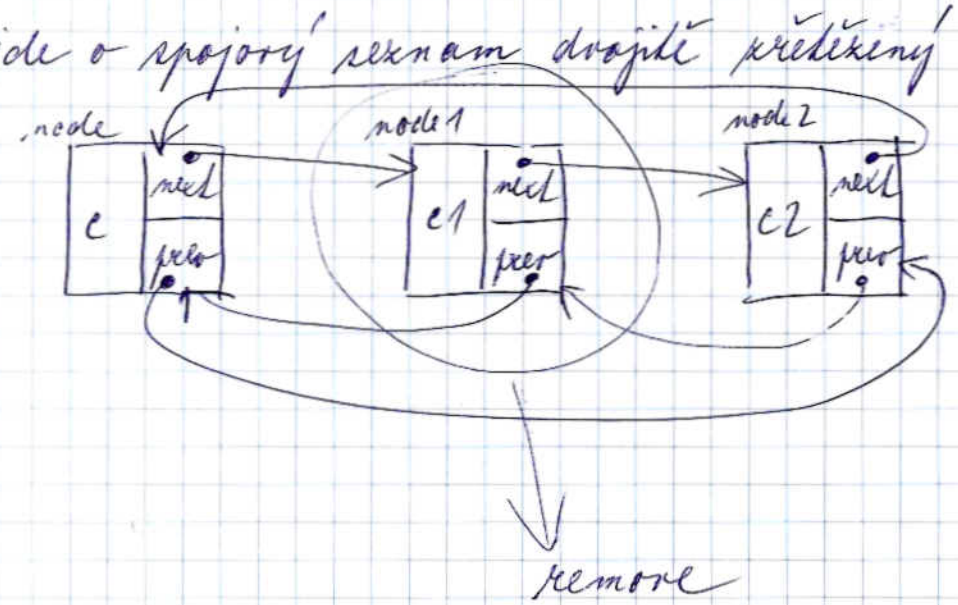
```
class Node {
        Node next, previous;
        Object content;
}
```

remove (Objekt c)

- vyhledat (c) – je tam f = return();
  - equals
  - není tam f = NULL;
- je tam => remove

hledám, kde je (c) c == node.c

jde o spojový seznam dvojitě zřetězený



remove

```
Node find (Object c) {
    Node help = node.next;
    while (help != node) {
        if (c.quals (help.c)) return help;
        else help = help.next;
    }
    return NULL;
}
void remove (Object c) {
    Node f = find (c);
        if (f == NULL) return;
        else {
            f.prev.next = f.next;
            f.next.prev = f.prev;
        }
}
```

VARIANTA 2 - dle Ing. Píšeho

```
remove (Object c) {
    Node tmp = Node.next;
        while (tmp != NODE)
            if (TMP.CONTENTS.EQUALS(c) {
                TMP.NEXT.PREV = TMP.PREV;
                TMP.PREV.NEXT = TMP.NEXT;
            }
        TMP = TMP.NEXT;
}
```

(b) Do třídy Stack dopište metodu void enlarge() (zvětšení naalokovaného vnitřního pole) tak, aby amortizovaná složitost operace push byla konstantní.

```
class Stack {
    static final int INITIAL_CAPACITY = 7;
    int[] array = new int[INITIAL_CAPACITY];
    int size = 0;

    void push(int elem) {
            if (size == array.length) enlarge();
            array[size++] = elem;
    }

// kod metody pop vynechan
}
```

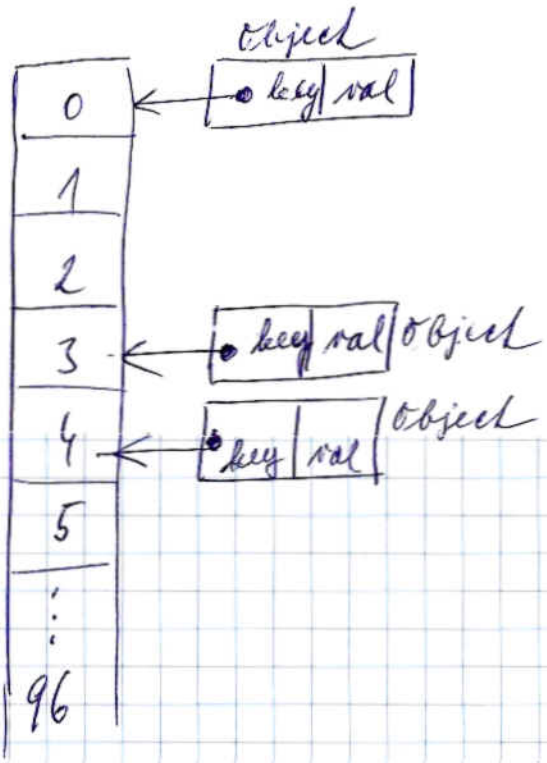
Zásobník (Stack)

KROKY:
- jde o amortizovanou složitost
- vytvořit new array1 = array.length * 2
- copy arr to arr1 => arr1[i] = arr[i]
- zrušit array

```
int [] arr1 = new int [arr.length * 2];
    for (i=0; i < arr.leng; i++){
        arr1[i] = arr[i];
    }
    arr = arr1;
}
```

(c) Do třídy Dictionary dopište metodu Object find(Object key) (nalezení a vrácení hodnoty asociované s daným klíčem). Pro porovnání klíčů použijte metodu equals.

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

    void add(Object key, Object value) {
            // predpokladame, ze uzivatel nebude vkladat stejny klic dvakrat
            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);
    }

// kod metody enlarge vynechan
}

class Node {
    Object key, value;

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




```
Object find (Object key){
  for (i = key.hashCode(); arr.[i] != NULL ; i++) {
```

POZOR!!!
u hashCode MUSÍ být inicializace takto NE i=0;

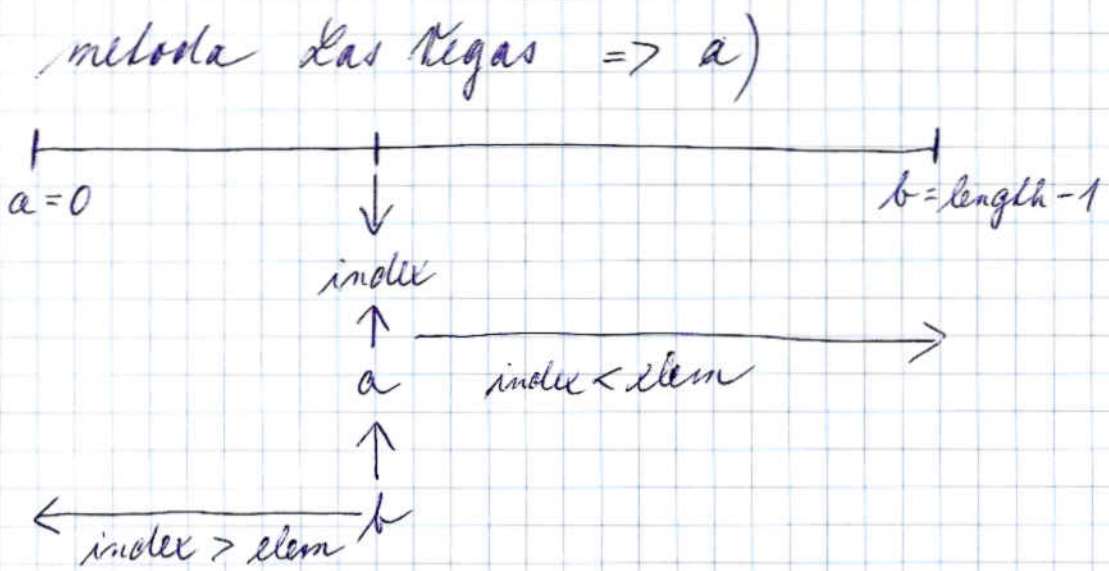
```
    if (array[i].key.equals(key)
        return arr[i].value;
  }
return NULL;          // = nenalezeno
}
```

26. 11. 2010

5.

(a) Randomizovaná funkce find vrací index prvku elem ve vzestupně se-tříděném poli array. Určete, do které z následujících dvou kategorií patří a svoji odpověď co nejexaktněji zdůvodněte:
(i.) není garantováno za jak dlouho se vrátí, ale pokud se vrátí, vrátí správný výsledek nebo
(ii.) existuje garance za jak dlouho se vrátí, ale v některých případech vrátí špatný výsledek.

```
int find(int elem, int[] array) {
    int a = 0;

    int b = array.length - 1;

    while (a < b) {
            int index = random(a,b);          // a <= index <= b
            if (array[index] == elem) return index;
            if (array[index] < elem) a = index;
            else b = index;
    }
return -1; // -1 znamena nenalezeno
}
```



(b) Randomizovaná funkce find vrací index prvku elem ve vzestupně se-tříděném poli array. Určete, do které z následujících dvou kategorií patří a svoji odpověď co nejexaktněji zdůvodněte:
(i.) není garantováno za jak dlouho se vrátí, ale pokud se vrátí, vrátí správný výsledek nebo
(ii.) existuje garance za jak dlouho se vrátí, ale v některých případech vrátí špatný výsledek.

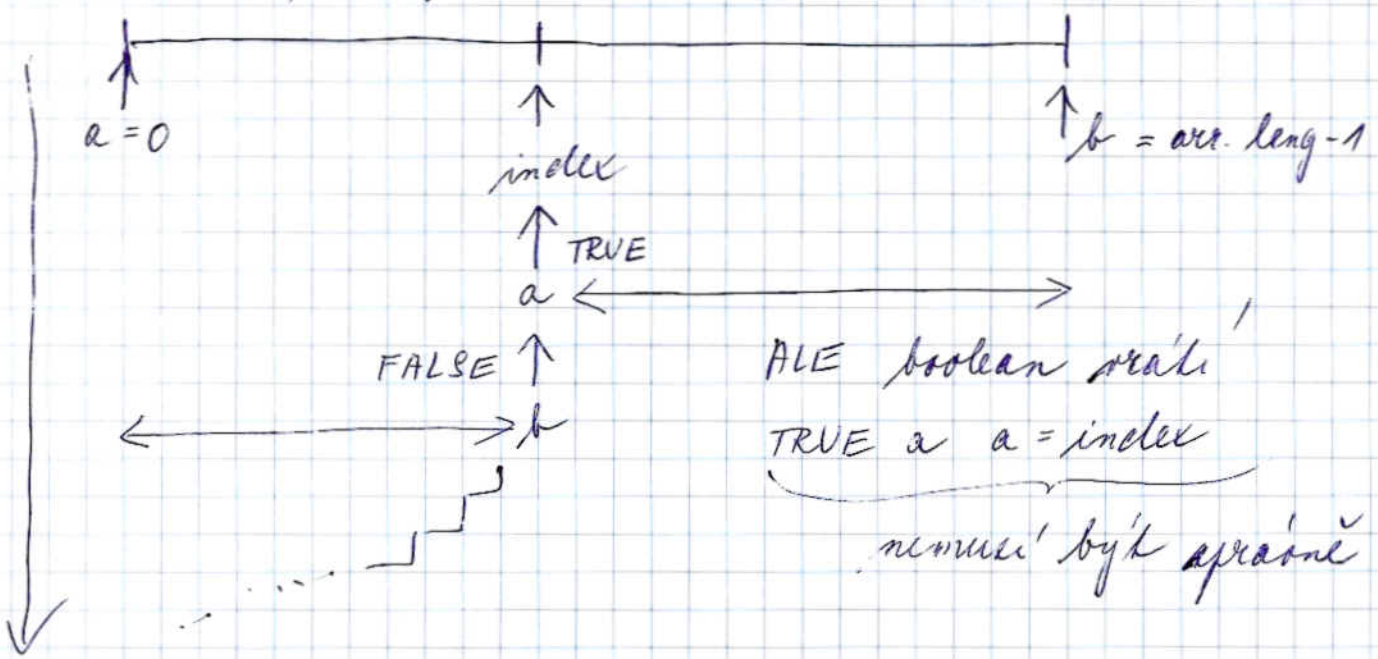
```
int find(int elem, int[] array) {
    while (true) {
            int index = random(0, array.length - 1);          // 0 <= index <= array.length - 1
            if ((array[index] < elem) &&
                    ((index == array.length - 1) || (array[index + 1] > elem)))
                    return -1;        // -1 znamena nenalezeno
            if (array[index] == elem) return index;
    }
}
```

i) <= metoda Las Vegas

(c) Randomizovaná funkce find vrací index prvku elem ve vzestupně se-tříděném poli array. Určete, do které z následujících dvou kategorií patří a svoji odpověď co nejexaktněji zdůvodněte:
(i.) není garantováno za jak dlouho se vrátí, ale pokud se vrátí, vrátí správný výsledek nebo
(ii.) existuje garance za jak dlouho se vrátí, ale v některých případech vrátí špatný výsledek.

```
int find(int elem, int[] array) {
    int a = 0;
    int b = array.length - 1;
    while (a < b) {
            int index = a + (b - a) / 2;
            if (array[index] == elem) return index;
            if (randomBoolean()) a = index;
            else b = index;
    }
    return -1;        // -1 znamena nenalezeno
}
```

správně ii) ⇐ Monte Carlo



hloubka log n = existuje garance
na jak dlouho se vrátí