DSA, 5. 11. 2010, varianta A

Pokud zadání nerozumíte nebo se vám zdá nejednoznačné, zeptejte se. Pištečitelně, nečitelná řešení nebudeme uznávat.

(A) 1

(1) Podle definice napište důkaz následujícího tvrzení: $3 \in \Theta(1)$.

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

$3 \overset{?}{\in} \Theta(1)$ (3 = g, 1 = f)

$d \cdot f(n) \leq g(n) \leq c \cdot f(n)$

$d \cdot 1 \leq 3 \leq c \cdot 1$

$d \leq 3$ pro $d = \langle 0, 3 \rangle$

$c \geq 3$ pro $c = \langle 3, \infty)$

1.

Pokud se má dokázat, že $3 \epsilon \Theta(1)$, tak musí obecně platit

$$f(x) \epsilon \Theta(g(x)) \Leftrightarrow f(x) \epsilon O(g(x)) \wedge f(x) \epsilon \Omega(g(x))$$

Pak se každé tvrzení dokazuje zvlášť.

a) $3 \epsilon O(1)$

$$\exists c > 0, \exists n_0 > 0, \forall n \geq n_0, 3 \leq c \cdot 1$$

Necht $c = 10$, je jedno co zvolíš, zkrátka musí existovat c, které je kladné, nic víc nepotřebuješ. A pak daná nerovnice platí už pro libolvolné n. Tj. $3 \epsilon O(1)$.

b) $3 \epsilon \Omega(1)$

$$\exists c > 0, \exists n_0 > 0, \forall n \geq n_0, 3 \geq c \cdot 1$$

Stejný argument, stačí ukázat, že ex. kladné c, a to je např. $c = 1$. Opět pro libovolné n.

(2) Pro funkci findMin určete asymptotickou časovou složitost v nejhoršímpřípadě. Svoji odpověď zdůvodněte.

(A) 2

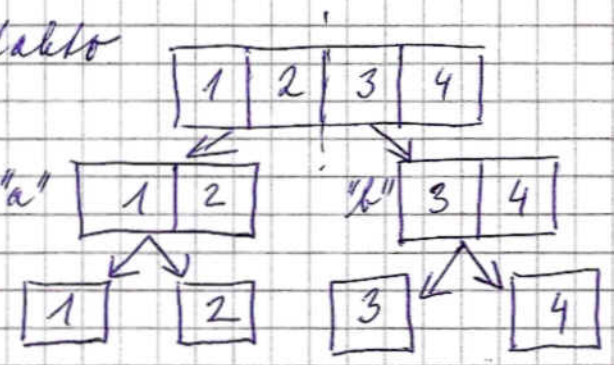
```
static int findMin(int[] array) {
        if (array.length == 1) return array[0];
        int half = array.length / 2;
        int[] a = new int[half];
        for (int i = 0; i < half; i++) a[i] = array[i];
        int[] b = new int[array.length - half];
        for (int i = half; i < array.length; i++) b[i - half] = array[i];
        int aa = findMin(a);
        int bb = findMin(b);
        return aa < bb ? aa : bb;
}
```

Zkus ten kód okomentovat.

```
static findMin(int[] arr) { // fce dostáva za argument pole
  if(arr.length == 1) return arr[0];
     // asi nějaká zarážka, pokud je délka pole jedna, vrací
     // hodnotu daného prvku pole
  int half = arr.length / 2;
     // rozdělení pole na dvě poloviny, half je uprostřed
  int[] a = new int[half];
     // vytvoření nového pole a, poloviční délky
  for (.....) a[i] = arr[i];
     // a překopírování první poloviny prvků z pole arr do pole a
  int[] b = new int[arr.len - half];
     // vytvoření nového pole b, poloviční délky
  for (.....) b[i] = arr[i];
     // a překopírování druhé poloviny prvků z pole arr do pole b

  int aa = findMin(a);
  int bb = findMin(b);
    // rekurentní volání nejprve s první polovinou původního pole (a)
    // a také pro druhou polovinu pole (b)
    // zkus si to pro pole délky 2, 4, 8 nakreslit
    // do proměnných aa, resp. bb se uloží nejnižší hodhota
    // dané podčásti levé, resp. pravé

  return aa < bb ? aa: bb;
    // vrací menší hodnotu
}
```

Pro 4 prvky se to rozpadne takto

„vyhledávání půlením"



Vstupuje pole (1,2,3,4), to je rozděleno na polovinu a=(1,2), b=(3,4), počet operací je 2+2 počet kopírování (do každého pole dva prvky). Zavolá se se int aa = findMin((1,2)). Dobře. To se tedy opět rozpadne na pole a = (1), b = (2). A opět se volá int aa = findMin ((1)), to už je jeden prvek a tak se alg. vrací s aa = 1 a volá se int bb = findMin((2)), to samé. Porovná se aa, bb a vrací se 1. A volá se int bb = findMin((3,4)). Atd.

Pro libovolné n, se rozpadne na poloviny, kde provádí n operací (přesun). Počet pater je $\log_2 n$ a v každém patře se dálá n operací tj. $\Theta(n \log n)$. Je to jak minimální tak i maximální složitost.

Nebo to můžeš rozepsat na Master Theorem.

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

$$T(n) = cn + 2T(\lceil \frac{n}{2} \rceil)$$

Nemůžu najít symbol pro horní část, to jsem tam neměl tak uvidíme. Nakresli si to!

(A) 3

MASTER THEOREM

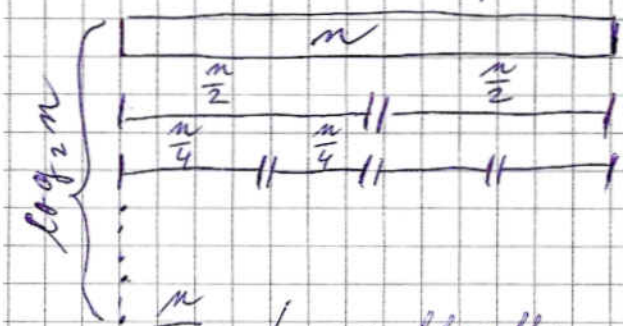
$$T(n) \begin{cases} a, & n = 1 \\ c \cdot n + AT\left(\frac{n}{B}\right) \end{cases}$$

- $A = B \quad \Theta(n \log n)$
- $A < B \quad \Theta(n)$
- $A > B \quad \Theta(n^{\log_B A})$

úvaha - kolik kroků udělám?

každá úroveň - počet operací „v patře" = k

- počet prvků = n



$\frac{k}{2} + \frac{k}{2} = k$

$\frac{k}{4} + \frac{k}{4} + \ldots = k$

$\Rightarrow n = k$

Celkem počet kroků $n \cdot \log_2 n$

$\frac{n}{2^m}$ (m = hloubka stromu)

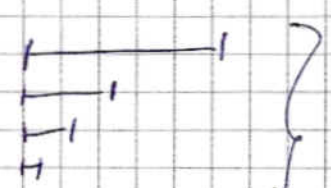
$\frac{n}{2^m} = 1 \sim n = 2^m \sim n^1 = 2^{1 \cdot \log_2 n}$

$n = 2^m \mid \cdot \log_2$

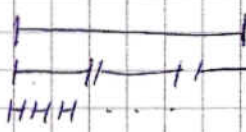
$\log_2 n = (\log_2 2)^m \sim \log_2 n = m \cdot \underbrace{\log_2 2}_{=1}$

obecně $x^n = e^{n \cdot \log x} \sim x^{n \cdot \log_x x}$

Pokud platí



} počet kroků $\log_2 n$



} $\log_3 n$ atd. ~~A,B,C~~

ŘEŠENÍ!

Časová složitost $T(n)$ funkce findMin se dá popsat rovnicí $T(n) = 2T(n/2) + cn$. Jinými slovy se v $\log n$ iteracích vykonává cn kroků, asymptotická složitost je tedy $O(n \log n)$.

3. Určete smysluplný (tzn. použitelný pro důkaz správnosti funkce) invariant smyčky ve funkci findMin, zapište jej ve formě výroku a vyznačte, ve kterém místě programu by měl platit.

(A) 4

```
static int findMin(int[] array) {
    int min = array[0];
    for (int i = 0; i < array.length; i++)
        if (min > array[i]) min = array[i];
    return min;
}
```

To jsem moc nevěděl, ale nakonec jsem napsal toto.

```
...
for(...) {
  if ( min > arr[i] ) min = arr[i];
  Invariant I;
}
...
```

Kde *Invariant I* musí splnovat. Hodnota min je nejmenší pro již prohledaný úsek, nebo něco jako neexituje v prohledaném úseku hodnota, která by byla menší než je hodnota min. Napsal jsem toto.

MIN (arr[0], ..., arr[j])

$$0 \leq i < arr.len \wedge \forall j \epsilon < 0, i >: min = MIN(arr[j])$$

Kde MIN je funkce (matematicky chápána), která vrací minimální hodnotu.

invariant = podmínka ve smyčce, která musí pořád platit

invariant vrací ANO-NE, ale není to boolean metoda

píše se buď na začátku nebo na konci smyčky - podle toho je nutno sestavit matemat. výraz

invariat MUSÍ být vyjádřen matemat. výrazem

(PŘ) Invariat bylo možno složit i takto

```
...
for (...) {
    Invariant I;
    if (...) ...;
} ...
```

ŘEŠENÍ

Invariant můžeme umístit před nebo za podmínku.

- V případě, že se bude nacházet před podmínkou, bude vypadat následovně:

$$min = \min_{0 \leq j \leq i-1} array[j].$$

- Za podmínkou pak bude mít tvar:

$$min = \min_{0 \leq j \leq i} array[j].$$

4. Do třídy List dopište metodu boolean isEmpty() (kontrolu prázdnosti).

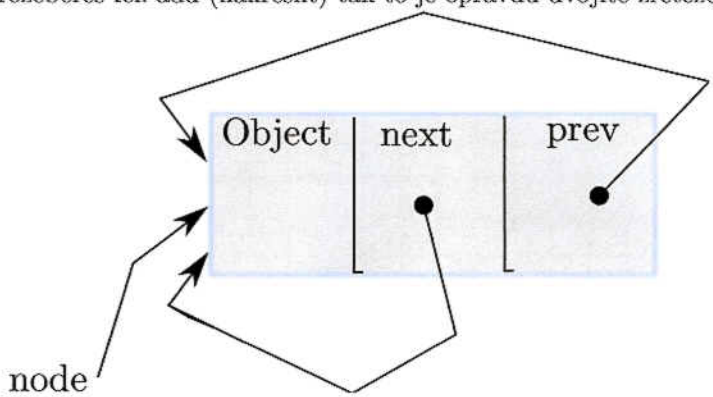
A 5

```
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;
}
```

Jedná se o spojový seznam, který je dvojitě zřetězen. Ukazatel na následující (next) a předchozí (prev) prvek.

Konstruktor List, třídy List, vytvoří jeden *dummy* prvek (viz scripta, přednášky), ve kterém není hodnota a ukazatel next a prev ukazují na node. Pokud si rozebereš fci. add (nakreslit) tak to je opravdu dvojitě zřetězený seznam.



Z obrázku to je jasné. List je prázdný, když node.next a node.prev ukazují na sebe sama (na node). Napsal jsem toto

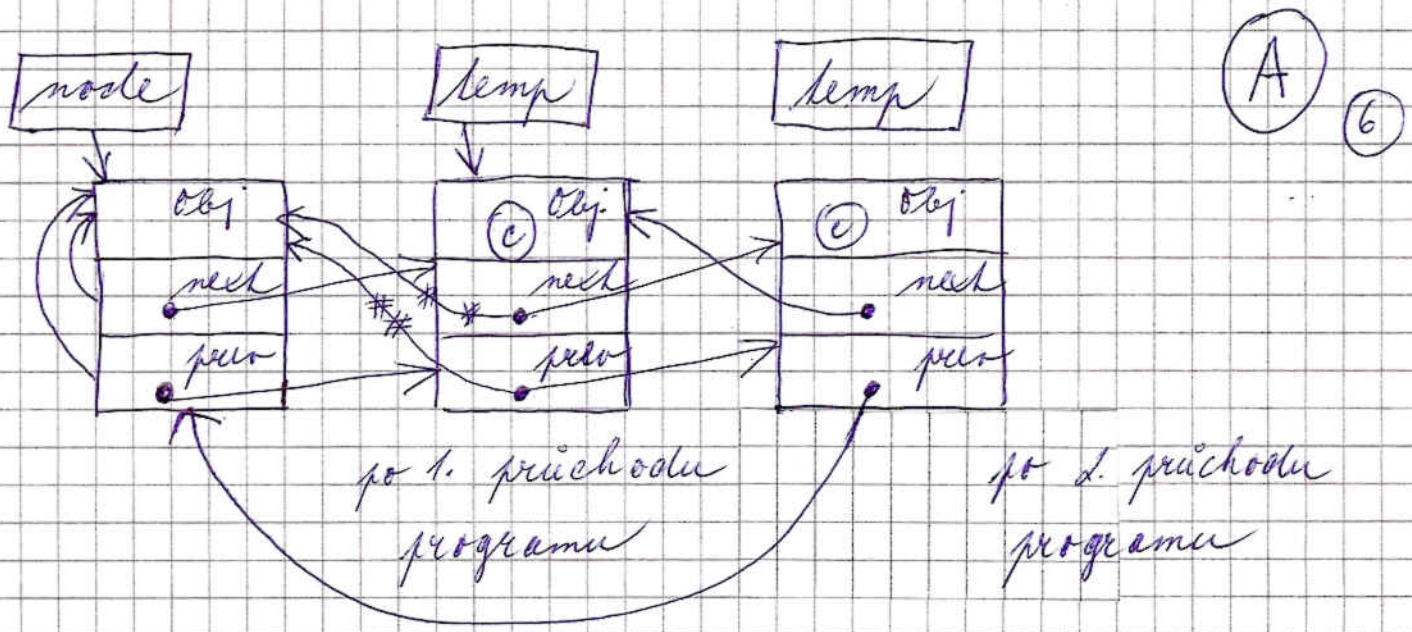
```
bool isEmpty() {
  return node == node.next && node == node.prev;
}
```

Možná, že stačí i toto

```
bool isEmpty() {
  return node.next == node.prev;
}
```

ŘEŠENÍ

```
boolean isEmpty() { return node == node.next; }
```


node
temp
temp
A
6
Obj
next
prev
C
Obj
next
prev
C
Obj
next
prev
po 1. průchodu programu
po 2. průchodu programu

(A) 4

(5) Randomizovaná funkce Object f(int a, int b) má desetiprocentní pravděpodobnost předčasného ukončení výpočtu. Pokud se tak stane, vrátí místo nějakého objektu null. Použijte ji pro implementaci funkce g s identickou funkčností a pravděpodobností předčasného ukončení bez odpovědi pouze1%. Svoji odpověď zdůvodněte.

$g() = \frac{1}{10}$ ukončení bez odpovědi $= \frac{1}{100}$

obecně - pokud mám pravděpodobnost jedné fce 10%

$\left(\frac{1}{10}\right)^n \leq \frac{1}{100}$

pokud hledám špatné jevy, pak hledám „n"

n je vždy celé číslo

(TŘ) $\frac{1}{100} = \frac{1}{10} \cdot \frac{1}{10} \Rightarrow$ fci $f()$ je nutno spustit 2x (min)

```
g() {
    f()
    f()
}
```

ŘEŠENÍ

V případě neuspěchu stačí funkci f zavolat podruhé. Tím se pravděpodobnost neuspěchu sníží na 10% z 10%, tedy na 1%.

zkouška $\left(\frac{1}{10}\right)^1 \leq \frac{1}{100}$ NE

$\left(\frac{1}{10}\right)^2 \leq \frac{1}{100} \sim \frac{1}{100} \leq \frac{1}{100}$ ANO

$g()$ má 5% špatně $f()$ 1% špatně

pak hledám n z $\left(\frac{5}{100}\right)^n \leq \frac{1}{100}$

$\left(\frac{5}{100}\right)^1 \leq \frac{1}{100} \sim$ NE $\left(\frac{5}{100}\right)^2 \leq \frac{1}{100} \sim$ ANO

DSA, 5. 11. 2010, varianta B

Pokud zadání nerozumíte nebo se vám zdá nejednoznačné, zeptejte se. Pište čitelně, nečitelná řešení nebudeme uznávat.

B 1

1. Podle definice napište důkaz následujícího tvrzení: $100n \in o(n2)$. n^2

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

$100n \in o(n^2)$

g $\quad$ f

$g(n) \leq c \cdot f(n)$

$100n \leq c \cdot n^2 \quad / : n$

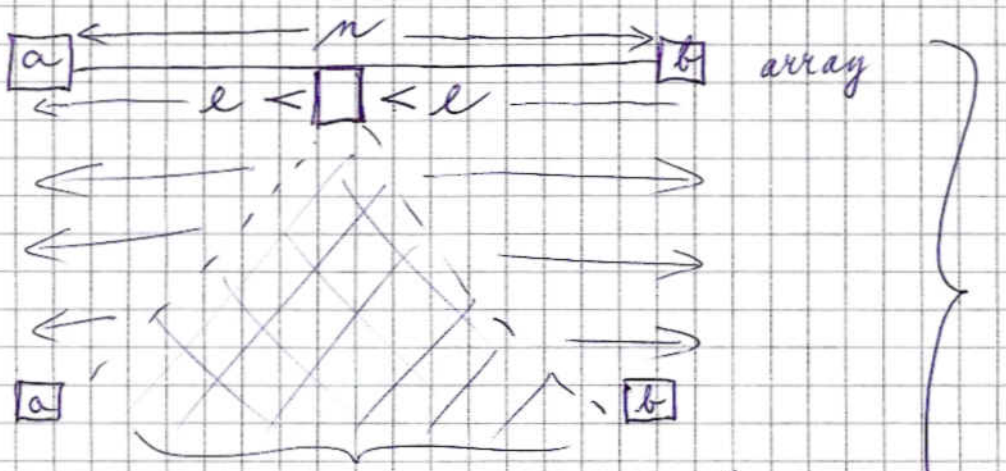
$100 \leq c \cdot n$

$\frac{100}{c} \leq n$

2. Pro funkci find určete asymptotickou časovou složitost v nejhorším případě. Svoji odpověď zdůvodněte.

B 2

```
// predpokladame, ze pole array je setridene
static int find(int e, int[] array, int a, int b) {
        if (a == b) return e == array[a] ? a : -1;    // -1 znamena nenalezeno
        else return e < array[a + (b - a) / 2] ?
                find(e, array, a, a + (b - a) / 2) :
                find(e, array, a + (b - a) / 2 + 1, b);
}
```



již se neprohledává

s každým cyklem se „n" zmenšuje

⇒ nemusíme se na něj brát ohled

nejhorší varianta = prvek v poli není

časová složitost $O(\log n)$

ŘEŠENÍ

Časová složitost $T(n)$ funkce find se dá popsat rovnicí $T(n) = T(n/2) + c$. Z této rovnice se dá odvodit, že funkce skončí po $\log n$ iteracích a v každé iteraci odvede konstantní práci, její celková časová složitost je tedy $O(\log n)$.

3. Určete smysluplný (tzn. použitelný pro důkaz správnosti funkce) invariant smyčky ve funkci sum, zapište jej ve formě výroku a vyznačte, ve kterém místě programu by měl platit.

B

3

```
static int sum(int[] array) {
        int sum = 0;
        for (int i = 0; i < array.length; i++)
                sum += array[i];
        return sum;
}
```

```
{...
    for (...) {
        sum ...;
        Invariant I;
    }
    return;
}
```

Invariant I: $sum = \sum_{k=0}^{i} (a[k];\ 0 \leq i < array.length)$

ŘEŠENÍ

Invariant můžeme umístit před nebo za řádek "sum += array[i]".

- V prvním případě můžeme invariant zapsat jako:

$$sum = \sum_{j=0}^{i-1} array[j].$$

- V druhém případě by vypadal následovně:

$$sum = \sum_{j=0}^{i} array[j].$$

4. Do třídy Queue dopište metodu boolean isEmpty() (kontrolu prázdnosti).Předpokládejte, že uživatel do fronty nikdy nevloží víc, než capacity−1prvků.

B

4

```
class Queue {
        final int capacity;
        final int[] contents;
        int first, last;

        public void push(int elem) {
                contents[first] = elem;
                first = (first + 1) % capacity;
        }

        public int pop() {
                int result = contents[last];
                last = (last + 1) % capacity;
                return result;
        }
}
```

```
public Queue(int cap) {
        capacity = cap;
        contents = new int[capacity];
}
```

konstruktor

→ modulo zajistí, že pole nepřeteče

```
boolean isEmpty() {
    return first == last; }
```

na začátku first == last == 0;

first == last

v ostatních krocích ~~již~~ jsou vkládány prvky

ŘEŠENÍ

```
boolean isEmpty() { return first == last; }
```

5. Randomizovaná funkce g má desetiprocentní pravděpodobnost vrácení špatného výsledku, randomizovaná funkce h má dvacetiprocentní pravděpodobnost vrácení špatného výsledku. Jaká je (maximální) pravděpo-dobnost, že funkce f vrátí špatný výsledek? Svoji odpověď zdůvodněte.

B

5

```
int f() { return g() + h(); }
```

$g() = \frac{1}{10}$ $\quad h() = \frac{2}{10}$

0 = špatná odpověď → hledáme

1 = dobrá odpověď

možnosti:

g()		h()		
0	$\frac{1}{10}$	0	$\frac{2}{10}$	$\frac{1}{10} \cdot \frac{2}{10} = \frac{2}{100}$
0	$\frac{1}{10}$	1	$\frac{8}{10}$	$= \frac{8}{100}$
1	$\frac{9}{10}$	0	$\frac{2}{10}$	$\frac{18}{100}$
1	$\frac{9}{10}$	1	$\frac{8}{10}$	$\frac{72}{100}$

pravděpodobnost, kdy nastane současně daný stav

Stav 1-1 nás nezajímá.

Zajímají nás pouze stavy, kdy je alespoň ~~jeden výsledek~~ jedna odpověď 0

$\frac{2}{100} + \frac{8}{100} + \frac{18}{100} = \frac{28}{100}$ = max. pravděpodobnost, že funkce f vrátí špatný výsledek

/

ŘEŠENÍ

Funkce f vrátí správný výsledek pokud funkce g i funkce h vrátí správný výsledek. Pravděpodobnost vrácení špatného výsledku funkcí f je tedy $1 - [(1 - 0,1) \cdot (1 - 0,2)]$.

DSA, 5. 11. 2010, varianta C

Pokud zadání nerozumíte nebo se vám zdá nejednoznačné, zeptejte se. Pištečitelně, nečitelná řešení nebudeme uznávat.

C

1

1. Napište důkaz následujícího tvrzení: $n \in \omega(2)$.

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

$n \overset{?}{\in} \omega(2)$

n → g, 2 → f

$g(n) \geq c \cdot f(n)$

$n \geq c \cdot 2$

$n = \langle 0^+, \infty)$

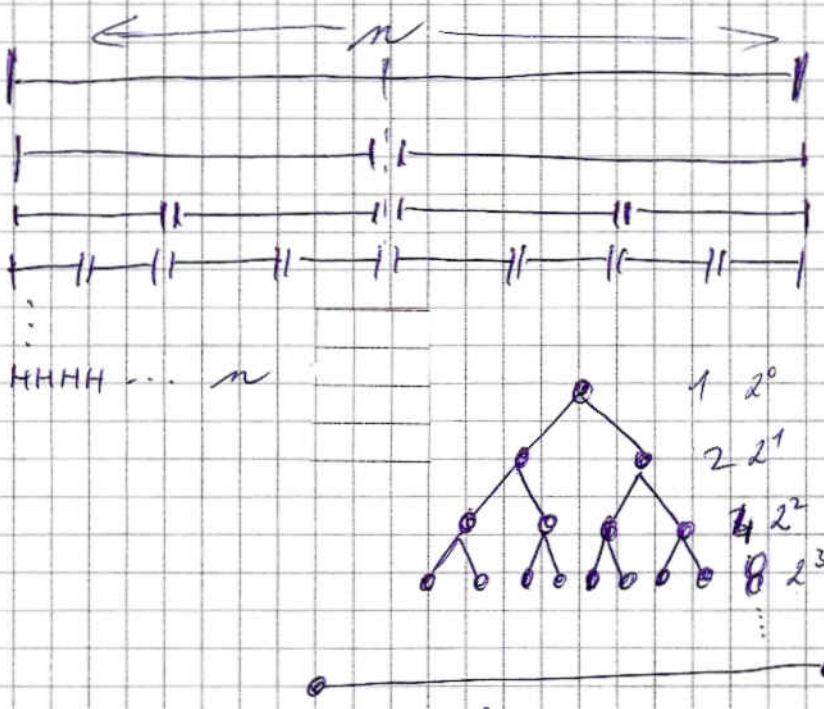
2. Pro funkci find určete asymptotickou časovou složitost v nejhorším případě. Svoji odpověď zdůvodněte.

```
static int find(int e, int[] array, int a, int b) {
        if (a == b) return e == array[a] ? a : -1;   // -1 znamena nenalezeno
        else{
                int i = find(e, array, a, a + (b - a) / 2);
                return i != -1 ? i : find(e, array, a + (b - a) / 2 + 1, b);
        }
}
```

C 2

ŘEŠENÍ

Časová složitost $T(n)$ funkce find se dá popsat rovnicí $T(n) = 2T(n/2) + c$. Díky exponenciálně se zvyšujícímu počtu kroků vykonaných v každé iteraci je pro asymptotickou časovou složitost podstatný jen čas strávený poslední iterací — v té se prochází každý prvek pole a vykonává se na něm konstantní práce, asymptotická časová složitost je proto $O(n)$.



$$2^0 + 2^1 + \dots + 2^k = \sum_{i=0}^{\log_2 n} 2^i = S$$

$$2^{\log_2 n}$$

$$\log_2 S = \log_2 \sum_i 2^i = \sum \log_2 2^i = \sum_{i=0}^{\log_2 n} i = \frac{\log_2 n}{2} \cdot (\log_2 n) - \log_2 n$$

$$= 2^{\log_2 S} = 2^k = n$$

$$O(n)$$

3. Určete smysluplný (tzn. použitelný pro důkaz správnosti funkce) invariant smyčky ve funkci zeros, zapište jej ve formě výroku a vyznačte, v kterém místě programu by měl platit.

C

3

```
static int zeros(int[] array) {
        int count = 0;
        for (int i = 0; i < array.length; i++)
                if (array[i] == 0) count++;
        return count;
}
```

varianta 1

```
{...
  for(...){
    Invariant J;
    if(...)}
  return; }
```

varianta 2

```
{...
  for(...){
    if(...)
  Invariant J;}
  return;}
```

Invariant J;

hledám (počet) $a[i] == 0$

(A) $\{j;\ a[j] = 0\}$

(B) $0 \leq i < array.length \wedge \forall j \in \langle 0, i \rangle : |\{j;\ a[j] = 0\}|$

(C) $count = \sum_{j=0}^{i} NULL(a[j])$

int NULL (x) — 1; x == 0
0; x != 0

ŘEŠENÍ

Invariant můžeme umístit před nebo za podmínku.

- V případě, že se bude nacházet před podmínkou, bude vypadat následovně:

$$count = |\{j : 0 \leq j < i,\ array[j] = 0\}|.$$

- Za podmínkou pak bude mít tvar:

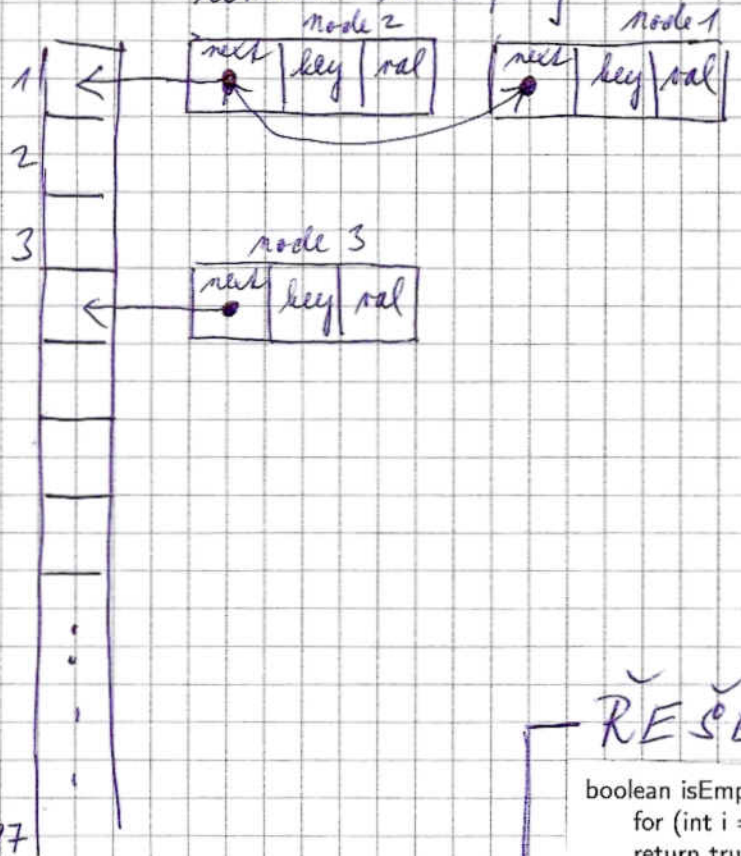
$$count = |\{j : 0 \leq j \leq i,\ array[j] = 0\}|.$$

4. Do třídy Dictionary dopište metodu boolean isEmpty() (kontrolu prázdnosti). Při implementaci této metody si musíte vystačit s existujícími atributy třídy Dictionary, tzn. nesmíte do ní přidávat žádné nové.

4

```
class Dictionary {
        static final int SIZE = 97;
        Node[] array = new Node[SIZE];

        public void add(Object key, Object value) {
                int h = key.hashCode() % SIZE;
                array[h] = new Node(array[h], key, value);
        }
}       // kod metod remove a find vynechan

class Node {
        Node next; Object key, value;

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

```
boolean isEmpty () {
  for (int i = 0; i < array.length; i++)
    if (array[i] != null) return false;
  return true; }
```



ŘEŠENÍ

```
boolean isEmpty() {
    for (int i = 0; i < array.length; i++) if (array[i] != null) return false;
    return true;
}
```

5. Randomizovaná funkce g má desetiprocentní pravděpodobnost vrácení špatného výsledku, randomizovaná funkce h má dvacetiprocentní pravděpodobnost vrácení špatného výsledku. Jaká je (maximální) pravděpodobnost vrácení špatného výsledku funkce f za předpokladu, že metoda i vrací ve čtvrtině případů hodnotu true? Svoji odpověď zdůvodněte.

```
int f() { return i() ? g() : h(); }
```

C 1 5

$g() = \frac{1}{10}$ $h() = \frac{2}{10}$

$\frac{1}{25}$ odpovídá = true = $\frac{1}{4}$

i		g			i		h		
0	$\frac{3}{4}$	0	$\frac{1}{10}$	$\frac{3}{4} \cdot \frac{1}{10} = \frac{3}{40}$	0	$\frac{3}{4}$	0	$\frac{2}{10}$	$= \frac{6}{40}$
0	$\frac{3}{4}$	1	$\frac{9}{10}$	$= \frac{27}{40}$	0	$\frac{3}{4}$	1	$\frac{8}{10}$	$= \frac{24}{40}$
1	$\frac{1}{4}$	0	$\frac{1}{10}$	$= \frac{1}{40}$	1	$\frac{1}{4}$	0	$\frac{2}{10}$	$= \frac{2}{40}$
1	$\frac{1}{4}$	1	$\frac{9}{10}$	$= \frac{9}{40}$	1	$\frac{1}{4}$	1	$\frac{8}{10}$	$= \frac{8}{40}$

int f() { return i() ? g() : h(); }

$\Rightarrow$ i() = 1 return g()

i() = 0 return h()

Zajímají nás ALE jenom případy

i() = 1 g() = 0 i() = 0 h() = 0

$\frac{1}{40} + \frac{6}{40} = \frac{4}{40}$

ŘEŠENÍ

Funkce g je volána ve čtvrtině případů, pravděpodobnost špatného výsledku pocházejícího od ní je tedy $1/4 \cdot 0,1$. Funkce h je volána ve třech čtvrtinách případů, pravděpodobnost špatného výsledku pocházejícího od ní je tedy $3/4 \cdot 0,2$. Celková pravděpodobnost vrácení špatného výsledku je $1/4 \cdot 0,1 + 3/4 \cdot 0,2$.