Níže uvedené úlohy představují přehled otázek, které se vyskytly v tomto nebo v minulých semestrech ve cvičení nebo v minulých semestrech u zkoušky. Mezi otázkami semestrovými a zkouškovými není žádný rozdíl, předpokládáme, že připravený posluchač dokáže zdárně zodpovědět většinu z nich.

Tento dokument je k dispozici ve variantě převážně s řešením a bez řešení.

Je to pracovní dokument a nebyl soustavně redigován, tým ALG neručí za překlepy a jazykové prohřešky, většina odpovědí a řešení je ale pravděpodobně správně :-).

-------------------------------------- RECURSION ---

1.
Určete, jakou hodnotu vypíše program po vykonání příkazu **`print(rekur(6));`**, když rekurzivní funkce **`rekur()`** je definována takto:

```
int rekur(int x) {
  if (x < 1) return 2;
  return (rekur(x-3)+rekur(x-4));
}
```

a) 6
b) 7
c) 8
d) 14
e) 16

2.
Určete, jakou hodnotu vypíše program po vykonání příkazu **`print(rekur(5));`**, když rekurzivní funkce **`rekur()`** je definována takto:

```
int rekur(int x) {
  if (x < 0) return 2;
  return (rekur(x-4)+rekur(x-2));
}
```

a) 5
b) 6
c) 8
d) 10
e) 16

3.

```
int fff (int x, int y) {
  if (x <= y) return x;
  return fff(y,x);
}
```

Uvedená funkce fff vrátí pro kladné hodnoty x a y:

a) max(x,y)
b) min(x,y)
c) vždy x
d) vždy y
e) nevrátí nic, bude volat stále sama sebe

4.

```
int ff(int x, int y) {
  if (x > 0) return ff(x-1, y)-1;
  return y;
}
```

Funkce ff provádí následující akci:

a) pro kladná x vrací 0, jinak vrací y
b) odečte x od y, pokud x je nezáporné
c) odečte y od x, pokud x je nezáporné
d) vrací –y pro kladné x, jinak vrací y
e) spočte zbytek po celočíselném dělení y%x

5.

```
int ggg(int x, int y) {
 if (x <= y) return y;
 return ggg(y,x);
}
```

Uvedená funkce ggg vrátí pro kladné hodnoty x a y:

a) max(x,y)
b) min(x,y)
c) vždy x
d) vždy y
e) nevrátí nic, bude volat stále sama sebe

6.

Určete, jakou hodnotu vypíše program po vykonání příkazu `print(rekur(5));`, když rekurzivní funkce `rekur()` je definována takto:

```
int rekur(int x) {
  if (x <= 0) return 1;
  return (rekur(x-2)+rekur(x-2));
}
```

a) 4
b) 7
c) 8
d) 15
e) 16

7.

Determine what value will be printed as result of the function call `print(recur(4));`, Recursive function `recur()` is defined as follows:

```
int recur(int x) {
  if (x < 0) return 1;
  return (recur(x-2)+recur(x-2));
}
```

a) 4
b) 6
c) 8
d) 16
e) 32

8.
Určete, jakou hodnotu vypíše program po vykonání příkazu **`print(rekur(2));`**, když rekurzivní funkce **`rekur()`** je definována takto:

```
int rekur(int x) {
  if (x < 0) return 1;
  return (rekur(x-2) + rekur(x-1));
}
```

a) 2
b) 3
c) 4
d) 5
e) 8

9.
The call of the function **`recur(2)`** produces the sequence:

```
void recur(int x) {
  if (x < 0) return;
  print(x);
  recur(x-1);
  recur(x-1);
}
```

a) 1 1 2
b) 2 1 1 0 0
c) 0 0 1 0 0 1 2
d) 0 1 0 2 0 1 0
e) 2 1 0 0 1 0 0

10.
The call of the function **`recur(2)`** produces the sequence:

```
void recur(int x) {
  if (x < 0) return;
  recur(x-1);
  recur(x-1);
  print(x);
}
```

a) 1 1 2
b) 2 1 1 0 0
c) 0 0 1 0 0 1 2
d) 0 1 0 2 0 1 0
e) 2 1 0 0 1 0 0

11.
Determine the exact number of calls of the **`xyz()`**function while perfoming the command **`print(recur(2));`**. Recursive function **`recur()`** is defined as follows:

```
int recur(int x) {
  if (x < 1) return 2;
  xyz();
  return (recur(x-1)+recur(x-2));
}
```

a) 2
b) 3
c) 5
d) 6
e) 8

12.

```
void ff(int x) {
  if (x >= 0) ff(x-2) ;
  abc(x);
  if (x >= 0) ff(x-2) ;
}
```

Daná funkce ff je volána s parametrem 2: **ff(2);**. Funkce **abc(x)** je tedy celkem volána

a) 1 krát
b) 3 krát
c) 5 krát
d) 7 krát
e) 8 krát

13.

```
void gg(int x) {
  if (x < 0) return;
  abc(x);
  gg(x–1);
  gg(x–1);
}
```

Daná funkce gg je volána s parametrem 2: **`gg(2);`**. Funkce **`abc(x)`** je tedy celkem volána

a) 1 krát
b) 3 krát
c) 4 krát
d) 7 krát
e) 8 krát

14.

```
 void fff(int x) {
  if (x < 0) return;
  abc(x);
  fff(x–1);
  fff(x–2);
}
```

Daná funkce fff je volána s parametrem 2: **`fff(2);`**. Funkce **`abc(x)`** je tedy celkem volána

a) 1 krát
b) 3 krát
c) 4 krát
d) 7 krát
e) 8 krát

15.

Vypočtěte, kolik celkem času zabere jedno zavolání funkce **`rekur(4);`** za předpokladu, že provedení příkazu **`xyz();`** trvá vždy jednu milisekundu a že dobu trvání všech ostatních akcí zanedbáme.

```
void rekur(int x) {
  if (x < 1) return;
  rekur(x-1);
  xyz();
  rekur(x-1);
}
```

16.

Určete, jakou hodnotu vypíše program po vykonání příkazu **`print(rekur(4));`**, když rekurzivní funkce **`rekur()`** je definována takto:

```
int rekur(int x) {
  if (x < 1) return 2;
  return (rekur(x-1)+rekur(x-1));
}
```

Nedokážete-li výsledek přímo zapsat jako přirozené číslo, stačí jednoduchý výraz pro jeho výpočet.

17.
Funkce
```
  int ff(int x, int y) {
   if (x > 0) return ff(x-1,y)+y;
   return 0;
  }
```

a) sčítá dvě libovolná celá čísla
b) násobí dvě libovolná celá čísla
c) násobí dvě celá čísla, pokud je první nezáporné
d) vrací nulu za všech okolností
e) vrací nulu nebo y podle toho, zda x je kladné nebo ne

18.
Funkce
```
int ff(int x, int y) {
 if (x > 0) return ff(x-1, y)-1;
 return y;
}
```

a) pro kladná x vrací 0, jinak vrací y
b) odečte x od y, pokud x je nezáporné
c) odečte y od x, pokud x je nezáporné
d) vrací –y pro kladné x, jinak vrací y
e) spočte zbytek po celočíselném dělení y%x

19.
Funkce
```
int ff(int x, int y) {
 if (x < y) return ff(x+1,y);
 return x;
}
```

a) buď hned vrátí první parametr nebo jen „do nekonečna“ volá sama sebe
b) vrátí x+1
c) vrátí součet svých parametrů
d) vrátí maximální hodnotu z obou parametrů
e) neprovede ani jednu z předchozích možností

20.
Funkce
```
int ff(int x, int y) {
 if (y>0) return ff(x, y-1)+1;
 return x;
}
```

a) sečte x a y, je-li y nezáporné
b) pro kladná y vrátí y, jinak vrátí x
c) spočte rozdíl x–y, je-li y nezáporné
d) spočte rozdíl y–x, je-li y nezáporné
e) vrátí hodnotu svého většího parametru

21.
```
void ff(int x) {
```

```
  if (x > 0) ff(x-1) ;
  abc(x);
  if (x > 0) ff(x-1) ;
}
```

Daná funkce ff je volána s parametrem 2: **`ff(2);`**. Funkce **`abc(x)`** je tedy celkem volána

a) 1 krát
b) 3 krát
c) 5 krát
d) 7 krát
e) 8 krát

22.

Funkce

```
int ff(int x, int y) {
 if (x < y) return ff(x+1,y);
 return x;
}
```

a) buď hned vrátí první parametr nebo jen „do nekonečna“ volá sama sebe
b) vrátí maximální hodnotu z obou parametrů
c) vrátí součet svých parametrů
d) vrátí x+1
e) neprovede ani jednu z předchozích možností

23.

Napište rekurzivní funkci, která pro zadané číslo N vypíše řetězec skládající se z N jedniček následovaných 2N dvojkami. Např. pro N = 3 vypíše 111222222.

```
void uloha9 (int n) {
  if ( n <= 0) return;
  printf("1");
  uloha9(n-1);
  printf("22");
}
```

24.

Pomocí rekurzivní funkce vypište pro zadané N posloupnost čísel 1 2 ... N-2 N-1 N N N-1 N-2 ... 2 1.

25.

Sečtěte (odečtěte, vynásobte, vydělte, umocněte) dvě nezáporná celá čísla pomocí rekurzivní (samozřejmě neefektivní) funkce.

26.

Napište rekurzivní funkci, která vypíše pouze hodnoty uložené v listech daného stromu (nebo jen ve vnitřních uzlech).

27.

Posloupnost 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 lze generovat rekurzivní funkcí zavolanou s parametrem 4.

```
  void ruler(int val) {
    if (val < 1) return;
```

```
    ruler(val-1);
    printf("%d%s",val," ");
    ruler(val-1);
  }
```

(Funkce se jmenuje ruler, neboť číselná posloupnost na jejím výstupu charakterizuje délky rysek na pravítku se stupnicí v binární soustavě.)

Zjistěte, co vypíší podobné funkce:
a)

```
  void ruler2(int val) {
    if (val < 1) return;
    printf("%d%s",val," ");
    ruler2(val-1);
    ruler2(val-1);
  }
```

b)

```
  void ruler3(int val) {
    if (val < 1) return;
    ruler3(val-1);
    ruler3(val-1);
    printf("%d%s",val," ");
  }
```

28.
Kolik znaků vypíše každá z funkcí v předchozí úloze, spustíme-li ji s prametrem 20?

29.
Sestavte rekurzivní proceduru, která vypíše všechny možnosti rozměnění stokoruny na 1, 2,5,10,20,50 korunová platidla.

30.
Ackermanova A(n, m) funkce je definována níže. Vypočtěte ručně hodnotu A(2, 2). Zjistěte, pro které dvojice n, m se dá hodnota A(n, m) vypočítat na běžném počítači (není jich příliš mnoho).

$$A(n, m) = \begin{cases} m+1 & \text{pro } n=0 \\ A(n-1,1) & \text{pro } n>0,\ m=0 \\ A(n-1,A(n,m-1)) & \text{pro } n>0,\ m>0 \end{cases}$$

31.
Schodová posloupnost
Posloupnost celých čísel nazveme schodovou, pokud absloutní hodnota rozdílu každých dvou sousedních prvků je právě 1. Prázdnou posloupnost a posloupnost s jediným prvkem považujeme také za schodové.
Ukázka 1.
a) 1 2 3 4 3 2 1
b) 1 2 1 2 1 2 3 2 3 4 3 4 5 4 5
c) 0 –1 –2 –1 –2 –1 0 1 0
d) 1 2 3 3 4 5 5
e) 8 7 5 4 3 4
Posloupnosti a), b), c) jsou schodové, posloupnosti d), e) nejsou schodové.
Ukázka 2.
Uvažujme nyní jako prvky posloupnosti pouze čísla 0 1 2 a délku posloupnosti rovnou 3. Všech schodových posloupností s těmito parametry je právě 6 a jsou to: 0 1 0, 0 1 2, 1 0 1, 1 2 1, 2 1 0, 2 1 2.

Úloha
Jsou dána celá čísla 1, 2, 3, ..., N a nezáporné celé číslo L. Napište program, jehož vstupem budou hodnoty N a L a výstupem bude seznam všech schodových posloupností délky L, které obsahují pouze hodnoty 1, 2, 3, ..., N. Použijte rekurzivní funkci.

32.
Je dána funkce fff vypsaná níže. Ve svém těle kromě sama sebe volá funkci abcd, která rekurzivní není a která proběhne v konstantním čase pro každou hodnotu svých parametrů. Vytvořte funkci ggg, která bude provádět tutéž činnost jako funkce fff, nebude však rekurzivní. Nápověda: Můžete se inspirovat nerekurzivním průchodem binárním stromem.

```
void fff(int x, int y) {
  if (x+y <= 0) return;
  fff(x-2, y-2);
  abcd(x,y);
  fff(x-2, y-2);
}
```

33.
Je dána funkce fff vypsaná níže. Ve svém těle kromě sama sebe volá funkci abcd, která rekurzivní není a která proběhne v konstantním čase pro každou hodnotu svých parametrů. Vytvořte funkci ggg, která bude provádět tutéž činnost jako funkce fff, nebude však rekurzivní. Nápověda: Můžete se inspirovat nerekurzivním průchodem binárním stromem.

```
void fff(int x, int y) {
  if (x+y <= 0) return;
  abcd(x,y);
  fff(x-2, y-2);
  fff(x-2, y-2);
}
```

-- RECURSION MASTER THEOREM -----------------------------------

1.
Rekurzivní algoritmus A dělí úlohu o velikosti n na 2 stejné části, pro zisk výsledku musí každou tuto část zpracovat dvakrát. Čas potřebný na rozdělení úlohy na části a na spojení dílčích řešení je úměrný hodnotě n. Asymptotická složitost algoritmu A je popsána rekurentním vztahem

a) $T(n) = 4T(n/2) + n$
b) $T(n) = n\cdot T(n\cdot 4/2)$
c) $T(n) = T(n/2) + 4n/2$
d) $T(n) = 2T(n/4) + n$
e) $T(n) = n\cdot T(n/2) + n\cdot \log(n)$

2.
Rekurzivní algoritmus A dělí úlohu o velikosti n na 3 stejné části a pro zisk výsledku stačí, když zpracuje pouze dvě z nich. Čas potřebný na rozdělení úlohy na části a na spojení dílčích řešení je úměrný hodnotě n^2. Asymptotická složitost algoritmu A je popsána rekurentním vztahem

$T(n) = n\cdot T(n\cdot 3/2)$
$T(n) = T(n/3) + 2n/3$
$T(n) = 3T(n/2) + n^2$
$T(n) = n\cdot T(n/2) + n^2$
$T(n) = 2T(n/3) + n^2$

3.
Rekurzivní algoritmus A dělí úlohu o velikosti n na 4 stejné části, zpracuje však jen tři z nich. Čas potřebný na rozdělení úlohy na části a na spojení dílčích řešení je úměrný hodnotě n. Asymptotická složitost algoritmu A je popsána rekurentním vztahem

$T(n) = 4T(n/3) + n$
$T(n) = n \cdot T(n \cdot 4/3)$
$T(n) = 4T(3n) - n$
$T(n) = 3T(n/4) + n$
$T(n) = n \cdot T(n/3) + n \cdot \log(n)$

4.
Rekurzivní algoritmus A dělí úlohu o velikosti n na 3 stejné části, každou z nich zpracuje dvakrát. Čas potřebný na rozdělení úlohy na části a na spojení dílčích řešení je úměrný hodnotě n^2. Asymptotická složitost algoritmu A je popsána rekurentním vztahem

$T(n) = 3T(n/6) + n^2$
$T(n) = n^2 \cdot T(n/3)$
$T(n) = 6T(n/3) + n^2$
$T(n) = 6T(3n) - n^2$
$T(n) = n^2 \cdot T(n/3) - n^2$

5.
Daný rekurzivní algoritmus pracuje tak, že pro $n > 1$ data rozdělí na 4 části stejné velikosti, zpracuje 5 těchto částí (tj. jednu z nich dvakrát) a pak jejich řešení spojí. Na samotné rozdělení problému a spojení řešení menších částí potřebuje dobu úměrnou hodnotě $n^2 - n$.

a. Nakreslete první tři úrovně (kořen a dvě další) stromu rekurze.

b. Předpokládejte, že kořen stromu odpovídá činnosti algoritmu nad daty velikosti n.
Vypočtěte cenu uzlu v hloubce 2 (=ve 3. úrovni) stromu. Cena uzlu je doba, kterou algoritmus potřebuje na rozdělení dat a sloučení vyřešených podproblémů při velikosti dat, která odpovídá hloubce uzlu.

c. Vypočtěte hloubku stromu rekurze.

d. Zjistěte asymptotickou složitost daného algoritmu použitím Mistrovské věty.

6.
Předchozí Úlohu řešte dále pro případy a, b, c, d níže, postup výpočtu bude analogický:
a. Daný rekurzivní algoritmus pracuje tak, že pro $n > 1$ data rozdělí na 3 části stejné velikosti, zpracuje každou tuto část dvakrát a pak jejich řešení spojí. Na samotné rozdělení problému a spojení řešení menších částí potřebuje dobu úměrnou hodnotě $\sqrt{n} \cdot \log_2(n)$.

b. Daný rekurzivní algoritmus pracuje tak, že pro $n > 1$ data rozdělí na 6 částí stejné velikosti, zpracuje každou tuto část a pak jejich řešení spojí. Na samotné rozdělení problému a spojení řešení menších částí potřebuje dobu úměrnou hodnotě $(n + 1)^2$.

c. Daný rekurzivní algoritmus pracuje tak, že pro $n > 1$ data rozdělí na 6 částí stejné velikosti, zpracuje 3 tyto části a pak jejich řešení spojí. Na samotné rozdělení problému a spojení řešení menších částí potřebuje dobu úměrnou hodnotě $\sqrt{n} + \log_2(n)$.

d. Daný rekurzivní algoritmus pracuje tak, že pro $n > 1$ data rozdělí na 3 části stejné velikosti, zpracuje každou tuto část a pak jejich řešení spojí. Na samotné rozdělení problému a spojení řešení menších částí potřebuje dobu úměrnou hodnotě $(n - 1)^2$.